

Steen Andreasen

# **MORPHX IT**

**Введение в язык программирования в Ахapta X++  
и среду разработки MorphX**

## **MORPHX IT**

### **Введение в язык программирования в Ахapta X++ и среду разработки MorphX**

Copyright © 2007 Steen Andreasen, [www.steenandreasen.com](http://www.steenandreasen.com)

Редактор: Steen Andreasen

Вып. редактор: Steen Andreasen

Обложка: Poul Cappelen og Ulla Bjulver

Фотограф: Ulla Bjulver

Перевод: Михаил Ржевский и Иван Кашперук

Denmark 2007

ISBN: 87-991161-4-6

1. edition

Все права зарегистрированы. Автор предоставил код только для использования читателями данного издания. Вы обладаете правом ограниченного использования кода из этой публикации только с ссылкой на автора, код не подлежит распространению, размещению online, в интернете, продаже или коммерческой эксплуатации. Никакая часть из этой публикации не может быть использована или переиздана любым способом без предварительного письменного разрешения правообладателя за исключением кратких обзоров в статьях или обзорах. Любое другое использование без письменного разрешения запрещено, согласно Датскому закону об авторском праве.

В случае обнаружения любых погрешностей, пожалуйста, сообщите автору по следующему адресу электронной почты:

[ahaptabook@steenandreasen.com](mailto:ahaptabook@steenandreasen.com)

### **Торговые марки**

Все термины, упомянутые в этой книге, являются торговыми марками и соответственно капитализированы. [steenandreasen.com](http://steenandreasen.com) не несет ответственности за эту информацию. Использование термина в этой книге не должно расцениваться, как воздействие на законность любой торговой марки.

### **Обратите внимание**

Не пытайтесь использовать код, приведенный в качестве примеров, при тестировании на рабочих станциях. Информация в этой книге приведена только в ознакомительных целях. Автор и сайт [steenandreasen.com](http://steenandreasen.com) не несут никакой ответственности за потерю данных или любой другой вред, полученный в результате использования информации из этой книги.

*“Спасибо моей дорогой жене Улле (Ulla), и моему сыну Оливеру (Oliver),  
кто создавал со мной и поддерживал меня во время написания этой  
книги.”*

S. A.

## Благодарности

**Выражаю благодарность** всем, кто прямо или косвенно внесли вклад в содержание этой книги, внося вдохновляющие комментарии и предложения.

**Отдельная благодарность** Ларису Холму (Lars Holm) за его вклад в Приложение Свойств. Полю Капелену (Poul Cappelen) и Улле Бюлвер (Ulla Bjulver) [www.photo-art.dk](http://www.photo-art.dk) за дизайн. А так же Jens Thrane, Christian Beck, Erik Pedersen, Lars Kjærsgaard, Jim Long, Hanne Paarup, Eric Fisher [www.unitederp.com](http://www.unitederp.com), Craig Brown [www.edenbrook.co.uk](http://www.edenbrook.co.uk), Daryl Spires [www.avionsystems.co.uk](http://www.avionsystems.co.uk), кто читал, редактировал рукопись и более всего поощрял меня к написанию этой книги.

## Рецензия на это издание

"Стин Андресен (Steen Andreasen) - это превосходный программист по Ахapta и технический специалист, а так же терпеливый преподаватель. В этой книге Стин Андресен тщательно разработал для Вас, увлекательное путешествие в мир разработки Ахapta.

Я настоятельно рекомендую эту книгу, как *должную быть* у разработчика и опытного и новичка, который хочет сделать карьеру в Ахapta Programming.

Выражаю вам большую благодарность, Стин Андресен, за ваши усилия по предложению такой хорошо структурированной информации для открытой публикации."

С теплыми пожеланиями,

Harish Mohanbabu  
Microsoft Dynamics Ax - MVP  
<http://www.harishm.com/>

## От переводчиков

Много лет моей сознательной жизни прошли в изучении продуктов Microsoft. Иногда вспоминаются времена первых операционных систем Windows 3.1, Windows' 95. Разработка в среде СУБД SQL прошла наиболее плотно для меня. Теперь разработка Ахартa.

Счастливый случай свел меня со Стином. Я сразу же, не раздумывая сел за перевод книги. Надеюсь, что мой перевод поможет начинающим разработчикам в освоении продукта. Уверен, что вам будет интересно окунуться в этот мир под названием Ахартa. Там много интересного и неизведанного. Теперь правда это уже называется Dynamics Ax.

Если у вас возникнет необходимость в контакте со мной, то прошу [MikeR.Ru@gmail.ru](mailto:MikeR.Ru@gmail.ru)

Так же можете ознакомиться с моими публикациями на <http://miker-developer.blogspot.com>

**Ржевский Михаил**

Я занимаюсь разработкой в Dynamics AX чуть более трех лет. За это время очень часто сталкивался с ситуацией, когда молодые разработчики отказываются от работы с этой ERP системой по причине отсутствия хорошей документации по разработке в среде MorphX. Поэтому такие книги, как эта, не только помогают самим разработчикам, но и повышают популярность продуктов Dynamics в целом, за что я особенно благодарен автору.

Посетив мой блог по адресу <http://kashperuk.blogspot.com>, Вы найдете полезные инструменты, советы и статьи по разработке в Dynamics AX.

**Кашперук Иван**

# Содержание

<b>ПРЕДИСЛОВИЕ .....</b>	<b>15</b>
<b>ВВЕДЕНИЕ.....</b>	<b>17</b>
Почему эта книга важна .....	18
Структура книги.....	18
<b>1    ВВЕДЕНИЕ В MORPHX.....</b>	<b>19</b>
<b>1.1    Репозиторий Прикладных Объектов (АОТ) .....</b>	<b>19</b>
Слои .....	20
Свойства .....	23
Add-ins.....	24
Редактор .....	24
Отладчик.....	27
Окно сообщения компилятора.....	29
Импорт и Экспорт .....	30
Сравнение объектов.....	32
Обновление кода .....	33
Поиск.....	34
Infolog .....	35
Мусорная корзина .....	37
Настройки пользователя .....	37
<b>1.2    Проект .....</b>	<b>41</b>
Изменения в проекте .....	41
Типы проектов .....	42
<b>1.3    Резюме .....</b>	<b>42</b>
<b>2    ВВЕДЕНИЕ В X++ .....</b>	<b>43</b>
<b>2.1    Переменные.....</b>	<b>44</b>
<b>2.2    Операторы .....</b>	<b>48</b>
Операторы присваивания .....	48
Операторы отношения .....	49
Бинарные операторы .....	50
<b>2.3    Условные операторы и операторы цикла.....</b>	<b>51</b>
Циклы .....	51
Условные операторы.....	53
Исключительные ситуации .....	56
Вспомогательные операторы .....	58
<b>2.4    Запросы к базе данных .....</b>	<b>59</b>
<b>2.5    Функции .....</b>	<b>66</b>

2.6	Резюме .....	67
<b>3</b>	<b>СЛОВАРЬ ДАННЫХ [DATA DICTIONARY] .....</b>	<b>69</b>
3.1	Таблицы.....	69
	Компания .....	70
	Прикладные таблицы .....	71
	Системные таблицы .....	77
	Поля .....	78
	Группы полей .....	81
	Индексы .....	83
	Связи [Отношения] .....	84
	Действия при удалении [Delete Actions] .....	86
	Методы .....	88
3.2	Карты соответствия (Maps) .....	94
3.3	Представления .....	97
3.4	Расширенные типы данных .....	98
	Расширенный тип данных из нескольких элементов .....	101
3.5	Перечислимые типы .....	102
3.6	Функциональные ключи.....	104
3.7	Лицензионные коды.....	105
3.8	Конфигурационные ключи .....	105
3.9	Ключи контроля доступа.....	107
3.10	Табличные коллекции .....	109
3.11	Расширенные возможности использования таблиц.....	110
	Использование системных классов .....	110
	Внешние базы данных.....	111
3.12	Резюме .....	113
<b>4</b>	<b>МАКРОСЫ .....</b>	<b>115</b>
4.1	Команды макроса .....	115
4.2	Определение констант.....	116
4.3	Создание макроса.....	117
4.4	Резюме .....	119
<b>5</b>	<b>КЛАССЫ .....</b>	<b>121</b>



<b>5.1</b>	<b>Основы класса .....</b>	<b>121</b>
	Методы .....	122
	Компоненты класса.....	124
	Модификаторы.....	127
	Передача значений.....	133
<b>5.2</b>	<b>AOS .....</b>	<b>137</b>
	Установка места выполнения .....	138
	Объекты для оптимизации.....	139
<b>5.3</b>	<b>Структура класса Runbase .....</b>	<b>139</b>
	Использование структуры класса Runbase .....	140
	Диалог .....	144
<b>5.4</b>	<b>Фундаментальные классы .....</b>	<b>149</b>
	ClassFactory.....	149
	Global .....	150
	Info.....	150
<b>5.5</b>	<b>Системные классы .....</b>	<b>150</b>
	Object .....	151
	Изменения во время исполнения .....	151
	Args .....	151
	Базовые классы .....	152
	Оптимизация операции записи.....	153
	Обработка файлов.....	153
<b>5.6</b>	<b>Специальное использование классов .....</b>	<b>154</b>
	Использование COM .....	154
	X++ Compiler.....	155
<b>5.7</b>	<b>Резюме .....</b>	<b>157</b>
<b>6</b>	<b>ФОРМЫ .....</b>	<b>159</b>
<b>6.1</b>	<b>Создание форм .....</b>	<b>159</b>
<b>6.2</b>	<b>Запрос формы.....</b>	<b>162</b>
	Объединение источников данных .....	162
	Установка доступа .....	166
<b>6.3</b>	<b>Дизайн .....</b>	<b>168</b>
	Создание дизайна.....	168
	Поля в дизайне .....	171
	Модификаторы Display и Edit.....	177
<b>6.4</b>	<b>Методы на форме .....</b>	<b>180</b>
	Методы формы .....	180
	Методы источника данных формы.....	184
	Методы полей источника данных.....	188
	Методы полей формы .....	190
	Последовательность выполнения методов формы .....	190
	Переопределение запроса формы .....	193
	Изменение источника данных из X++ .....	195

	Построение форм выпадающих списков .....	198
	Форма диалога .....	200
<b>6.5</b>	<b>Специальные формы .....</b>	<b>202</b>
	Вызов определенных методов формы .....	202
	Переопределение методов.....	203
	Общие изменения формы.....	206
	Цвета.....	208
<b>6.6</b>	<b>Резюме .....</b>	<b>209</b>
<b>7</b>	<b>ОТЧЕТЫ.....</b>	<b>211</b>
<b>7.1</b>	<b>Мастер отчета .....</b>	<b>211</b>
<b>7.2</b>	<b>Создание отчета.....</b>	<b>211</b>
<b>7.3</b>	<b>Запрос отчета .....</b>	<b>214</b>
<b>7.4</b>	<b>Шаблоны .....</b>	<b>217</b>
	Шаблон Отчета .....	217
	Шаблон секции .....	220
<b>7.5</b>	<b>Дизайн .....</b>	<b>220</b>
	Создание дизайна.....	220
	Авто дизайн .....	223
	Генерируемый дизайн .....	226
	Поля в дизайне .....	226
<b>7.6</b>	<b>Методы Отчета .....</b>	<b>228</b>
	Структура RunBase отчета.....	231
	Динамические отчеты.....	236
	Последовательность методов отчета.....	238
<b>7.7</b>	<b>Специальные Отчеты .....</b>	<b>242</b>
	Выполнение отчета из X++ .....	242
	Использование временных таблиц .....	243
	Подкраска рядов .....	245
	Печать отчета, используя Microsoft Word .....	247
<b>7.8</b>	<b>Резюме .....</b>	<b>250</b>
<b>8</b>	<b>ЗАПРОСЫ .....</b>	<b>251</b>
<b>8.1</b>	<b>Создание запросов.....</b>	<b>252</b>
	Запрос AOT .....	252
	X++ Запрос .....	259
<b>8.2</b>	<b>Запросы в формах и отчетах .....</b>	<b>261</b>
<b>8.3</b>	<b>Резюме .....</b>	<b>262</b>

<b>9</b>	<b>ЗАДАНИЯ .....</b>	<b>263</b>
9.1	Создание заданий.....	263
9.2	Резюме.....	264
<b>10</b>	<b>МЕНЮ И ПУНКТЫ МЕНЮ .....</b>	<b>265</b>
10.1	Пункты меню .....	265
10.2	Меню.....	267
	Определение названия объекта в АОТ из меню .....	267
10.3	Резюме.....	268
<b>11</b>	<b>РЕСУРСЫ .....</b>	<b>269</b>
11.1	Использование ресурсов.....	269
11.2	Резюме.....	272
<b>12</b>	<b>ПРИЛОЖЕНИЕ. СВОЙСТВА ОБЪЕКТОВ .....</b>	<b>273</b>
12.1	Свойства объектов Словаря данных.....	273
	Таблицы, Табличные карты соответствия и Представления .....	273
	Поля таблицы, карты соответствия .....	274
	Поля Табличных представлений.....	276
	Группа полей таблиц, карт соответствия, представлений.....	277
	Индекс таблицы .....	277
	Отношение на таблице .....	277
	Поле отношения на таблице.....	277
	Действие при удалении таблицы (DeleteAction) .....	278
	Mapping табличной карты соответствия.....	278
	Поле Mapping карты соответствия.....	278
	Расширенный тип данных.....	278
	Перечислимый тип.....	281
	Элемент перечислимого типа.....	281
	Лицензионные коды.....	282
	Конфигурационные ключи, ключи контроля доступа .....	282
12.2	Свойства форм .....	283
	Источник данных формы.....	283
	Поля источника данных формы .....	284
	Групповые элементы формы (Контейнеры).....	284
	Дизайн формы.....	289
	Управляющие элементы .....	291
12.3	Свойства отчетов. ....	307
	Отчет .....	308
	Дизайн отчета .....	308
	Авто дизайн .....	310
	Элементы секций отчета.....	310

	Шаблон секции .....	313
	Группа секций.....	313
	Элементы секций отчета.....	313
	Группа полей .....	323
<b>12.4</b>	<b>Свойства запроса .....</b>	<b>323</b>
	Запрос.....	323
	Источники данных.....	324
	Поля .....	324
	Поля сортировки .....	325
	Критерии выборки.....	325
<b>12.5</b>	<b>Свойства Меню .....</b>	<b>325</b>
<b>12.6</b>	<b>Свойства пунктов меню .....</b>	<b>326</b>
<b>13</b>	<b>ПРИЛОЖЕНИЕ. СРЕДСТВА РАЗРАБОТКИ MORPHX .....</b>	<b>329</b>
<b>13.1</b>	<b>Перекрестные ссылки .....</b>	<b>329</b>
<b>13.2</b>	<b>Объекты приложения .....</b>	<b>330</b>
	Формы объектов приложения.....	330
	Администрирование объектов.....	331
	Использование данных .....	331
	Мониторинг объектов .....	331
	Блокированные объекты .....	331
	Инструменты обновления .....	332
	Реиндексация.....	332
<b>13.3</b>	<b>Системный монитор.....</b>	<b>332</b>
	Отслеживание обращений к базе данных .....	332
	Отслеживание обращений к серверу приложений .....	332
<b>13.4</b>	<b>Профайлер кода.....</b>	<b>333</b>
<b>13.5</b>	<b>Иерархия объектов .....</b>	<b>334</b>
<b>13.6</b>	<b>Визуальное моделирование с MorphXplorer.....</b>	<b>335</b>
<b>13.7</b>	<b>Анализатор кода .....</b>	<b>337</b>
<b>13.8</b>	<b>Описания таблиц .....</b>	<b>337</b>
<b>13.9</b>	<b>Количество записей в таблицах .....</b>	<b>338</b>
<b>13.10</b>	<b>Тексты справки.....</b>	<b>338</b>
<b>13.11</b>	<b>Переход к новой версии.....</b>	<b>338</b>
	Переименованные прикладные объекты .....	339
	Создание проекта обновления приложения .....	339
	Сравнение слоев .....	339
<b>13.12</b>	<b>Мастера.....</b>	<b>340</b>
	Мастер отчетов .....	340

	Мастер мастеров .....	340
	Мастер меточных файлов.....	340
	Мастер создания классов .....	341
	Мастер оболочек для COM-объектов .....	341
<b>13.13</b>	<b>Метка .....</b>	<b>341</b>
	Поиск меток .....	342
	Журнал изменений меток.....	343
	Мастер меточных файлов.....	343
	Меточные интервалы .....	343
<b>14</b>	<b>ПРИЛОЖЕНИЕ. МАСТЕР ОТЧЕТОВ.....</b>	<b>345</b>



## Предисловие

Для успешного программирования первичен вопрос понимания нужд пользователя и перевод их в функциональное, техническое решение. Кроме того очень существенно, чтобы программист ясно представлял, как максимально приспособить систему к требованиям пользователя и затем мог легко модернизировать её.

Эта книга представляет введение в разработку приложения Ахартa. Эта книга не только упражнение в функциональности Ахартa, так как основана на более чем восьми лет разработки в Ахартa. Это так же практическая книга, содержащая множество примеров кода, как продукта разработки, так и разработки решений клиентов.

Издание этой книги было давнишнее мое желание. Я осознал через многие годы, работая с Ахартa, что практическая и в то же время учебная книга в этой области не доступна. Эта книга, MORPHX IT, полностью показывает мой профессиональный интерес к ERP системам и к Ахартa в частности.

Моё путешествие в авторский процесс было восхитительным и дало большой исследовательский опыт, получая почту от сотен людей со всего мира. Люди писали мне с комментариями и предложениями после публикации моей главы для свободной загрузки. Это показало стойкий интерес и необходимость этой книги.

Надеюсь, что эта книга вдохновит новичков для дальнейшего углубленного изучения, которые изучают ERP системы, а так же и для опытных профессионалов в среде разработки. Информацию, которую я публикую в этой книге, многие программисты утомительно собирают индивидуально, и моя цель сделать процесс разработки в Ахартa проще и дать вдохновение к продолжению разработки.

Стин Андерсен (Steen Andreasen)





## Введение

Эта книга – введение в среду разработки в Ахapta или просто MorphX.

Книга MORPHX IT написана как практическая книга. Практическая означает, что вы можете использовать книгу при разработке в Ахapta. Это делает книгу ценной в ежедневной работе, так как книга содержит много примеров. Я сделал именно так, полагая, что это наиболее быстрый и кратчайший путь к освоению языка разработки – использование системы сразу же.

У вас должно быть установлено приложение Ахapta и вы должны иметь базовые знания, как выглядит пользовательский интерфейс Ахapta. Эта информация может быть найдена в manuals в стандартной поставке.

Основное в этой книге – это то, что она написана с точки зрения разработчика. Вы можете использовать эту книгу, не имея знаний в Ахapta. Однако, будет легче понять содержимое, если у вас есть опыт использования приложения.

Вы получите большой результат, выполняя примеры при чтении книги.

Используемые примеры в этой книге включены в zip file

MORPHXIT\_1ED\_EXAMPLES.ZIP, который поставляется с книгой.

Использовалась конфигурация Ахapta 3.0 Service Pack 4 при написании книги.

Если использовать другой Service Pack Ахapta 3.0, у вас будут небольшие отличия.

Эта книга предназначена для прочтения людьми не имеющими знаний в программировании в Ахapta. Совсем не обязательно иметь опыт программирования. Эта книга так же предназначается и для технических специалистов или консультантов, которые хотят освоить среду разработки.

Если вы новичок, я рекомендую прочитать книгу с самого начала, так как главы содержат множество полезных деталей. При чтении, вы может быть обнаружите термины, которые не понимаете. Не все термины, используемые в главе, объясняются сразу же. Я это сделал для того, чтобы упростить содержание. Объяснение всех терминов сделало бы книгу слишком теоретической. Часто будут ссылки на другие секции в той же самой главе или на другие главы, где вы можете получить больше информации по конкретному термину. Как опытный пользователь Ахapta, вы будете с пониманием читать отдельные главы книги и схватывать содержимое.

## Почему эта книга важна

Во время моей работы с Ахарта, у меня был недостаток в документации по среде разработки в приложении Ахарта. Вы можете рассматривать эту книгу, как записную книжку или шпаргалку в начале программирования в Ахарта.

MORPHXIT - это первая книга по программированию и должна рассматриваться, как альтернатива курсам по изучению программирования в Ахарта.

В этой книге вы быстро получите необходимые знания для разработки в приложении Ахарта. Книга научит вас, как оптимизировать ваше приложение таким образом, что бы ваш код был легок в написании, а интерфейс более дружелюбным для простых пользователей.

## Структура книги

При написании этой книги, я должен был выбрать самые интересные части, как для вступительной книги. Веб разработка (web framework) не входит в эту книгу. По нескольким причинам я не затрагивал web framework. Основным было - это введение новичка в программирование в приложении Ахарта. Обычно человек опытный в программировании в Ахарта, начинает разработку в web framework. Так же для многих клиентов нет необходимости использования web части.

При разъяснении используемого инструмента разработки не все поля, кнопки, или возможности инструмента должным образом описаны. Тема часто объясняется с позиции человека, стоящего рядом с вами. Это особенно относится к

### **Приложение Среда разработки MorphX.**

При чтении книги вы увидите много хороших примеров программирования и рекомендаций. На самом деле я не вижу различий в этих двух терминах. Вместе они составляют мой собственный набор правил программирования или управления проектом разработки в Ахарта.

Главы в этой книге следуют в порядке главных узлов среды разработки Ахарта, начиная с вводных глав по среде разработки и языку разработки, и далее как использовать различные инструменты среды разработки.

В конце книги вы найдете главы приложения, которые содержат больше информации по некоторым темам этой книги.

# 1 Введение в MorphX

Среда разработки в Ахapta называется MorphX Development Suite или просто MorphX. MorphX – это интегрированная среда разработки, которая состоит из Репозитария Прикладных Объектов (АОТ), программного языка X++ и нескольких инструментов для работы с объектами приложения.

Все модули приложения Ахapta созданы в среде MorphX. При наличии соответственной лицензии Ахapta, выданной Microsoft, вы можете править любой из объектов в стандартной поставке. MorphX включает такие же инструменты, используемые Microsoft для разработки других программ. Использование технологии MorphX позволит вам расширить существующую функциональность для нужд вашей организации. В итоге Ахapta значительно проще подстраивается под бизнес процессы клиентов, чем другие ERP системы. Вы ускорите процесс разработки, потому что можете использовать и изменять существующий функционал - это быстрее, чем писать свой код с нуля. Так же поддерживается работа с такими СУБД, как Microsoft SQL Server и Oracle, и вы можете не заботиться о типе установленной СУБД. С точки зрения перспектив разработки ядро Ахapta можно приспособить под специфические выпуски других СУБД.


На разработку интерфейса у вас не уйдет много времени. В MorphX пользовательский интерфейс, состоящий из форм, и отчетов, по умолчанию почти полностью сгенерирован системой. Это значит, что вы не потратите много времени на размещение полей в дизайне. Всё это будет разъяснено в главах **Формы и Отчеты**.

## 1.1 Репозиторий Прикладных Объектов (АОТ)

Дерево прикладных объектов (АОТ) – это специальное меню для разработчика в Ахapta. Все объекты приложения хранятся в АОТ и представлены разработчику в виде дерева. Ахapta использует технологию слоев для хранения объектов. Эти слои помогают дифференцировать объекты в Ахapta. Стандартная поставка Ахapta состоит не только из объектов Microsoft, но и из объектов поставщиков решений, партнеров и собственных разработок клиентов.

При раскрытии узла в АОТ, потомки на следующем уровне кэшируются. При первом открытии узла, это занимает несколько секунд, Ахapta кэширует только используемые узлы. Когда в следующий раз Вы открываете кэшированный узел, то обнаружите, что он откроется быстрее.

Доступ к АОТ вы получаете, нажав комбинацию Ctrl+D или кликнув по иконке в верхнем меню. Заметьте, что при использовании демо-версии Ахapta, у вас нет доступа к АОТ. Для доступа к АОТ у вас должна быть соответствующая лицензия.

Для создания нового объекта кликните правой кнопкой мыши по узлу АОТ и в контекстном меню выберите создать. Красная вертикальная черта на узле объекта показывает, что узел изменен и не сохранен. Красная метка устанавливается при модификации узла. Все измененные узлы или недавно созданные можно сохранить, нажав на иконку  в верхнем меню окна АОТ.

Объекты, поля форм, отчетов и методы можно дублировать внутри АОТ. Методы и поля можно копировать. Функции дублирования и копирования доступны через контекстное меню. При дублировании узла, система создает копию выбранного узла в АОТ с префиксом *copyOf*. Это необходимо при тестировании модификации, а так же при резервировании предыдущего статуса объекта для отката назад, если функциональность не работает должным образом.

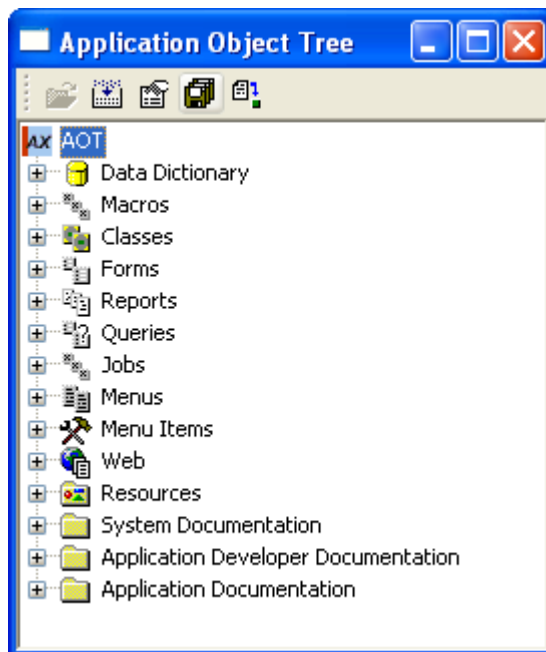


Рисунок 1, Дерево АОТ

АОТ также поддерживает технологию перетаскивания. При добавлении поля к форме или отчету, часто гораздо быстрее использовать перетаскивание полей, прямо из источника данных, чем создавать их с нуля. MorphX создаст поле с соответствующими свойствами.

При выборе узла в АОТ, у вас всегда есть доступ к контекстному меню через правую кнопку мыши текущего узла. Это меню *SysContextMenu* в АОТ, которое расположено в узле *Menu*. При обзоре меню в дереве АОТ все пункты меню доступны в списке контекстного меню. В зависимости от выбранного узла в АОТ у вас становятся доступными дополнительные пункты меню для выполнения специфических действий таких, как создание нового объекта, открытия нового окна или доступ к подменю *Add-Ins*. Открыть новое окно можно с корнем АОТ или текущим узлом. Вы можете открыть столько новых окон, сколько необходимо. Это необходимо при перетаскивании новых объектов в форму или отчет, или позиционировании курсора в АОТ.

## Слои

Технология слоев в Ахарт используется для разделения объектов базовой поставки от сделанных в них изменений и обновлений. Существует восемь

стандартных слоев. Каждый из стандартных слоев имеет дополнительный слой обновлений, итого получается 16 слоев. Нижние четыре слоя используются в стандартной поставке и не доступны ни партнерам, ни клиентам. Партнеры и клиенты имеют доступ к верхним слоям, каждый к двум слоям с дополнительным слоем обновления. Перед запуском Ахарта, вам необходимо определить текущий слой в конфигурационной утилите (Ахарта Configuration). Все слои за исключением верхних (USP, USR) требуют код доступа. При установке специальной лицензии, вы получаете доступ к верхним двум слоям и поэтому не можете модифицировать содержимое нижних шести слоев. Это предотвращает необратимые изменения основного кода Microsoft или бизнес партнеров. Но это не запрещает вам изменять этот код. При модификации стандартного объекта, Ахарта копирует часть или целый объект из одного из этих слоев и переносит в текущий слой. Получается, что объект может иметь изменения более чем в одном слое. Источник копии, верхнего уровня, ниже текущего слоя, в котором существует измененный объект. Ваши изменения сохраняются в текущем слое. Изменения, сделанные на верхнем уровне всегда можно отменить на нижележащем уровне, либо удалив слой. Если вы хотите начать изменения с начала, просто удалите объект в текущем слое, и вы вернетесь в исходное состояние. Текущий слой показывается в статусной строке в окне Ахарта. Для обзора слоев смотри **Рисунок 2: Обзор слоев**.

Слой	Доступ	Описание
SYS	Чтение	Это самый нижний слой. Здесь хранятся все объекты, созданные Microsoft.
SYP	Чтение	Слой обновления для слоя SYS. Используется для Service Packs.
GLS	Чтение	Решения поставщиков хранятся в этом слое. Используется для глобальных сертифицированных модулей, созданных сертифицированными субподрядчиками Microsoft. Это модули CRM и HRM.
GLP	Чтение	Слой обновления для слоя GLS. Используется для Service Packs.
DIS	Чтение	Слой используется для локализаций в пределах страны. Это слой, сгруппированный для стран с похожими требованиями. Если используется одно приложение с различными слоями DIS, слои необходимо объединить вручную.
DIP	Чтение	Слой обновлений для слоя DIS. Используется для Service Packs.
LOS	Чтение	Этот слой используется для локальных решений. Эти модули не сертифицируются глобально, как GLS слой. Например, это используется в Дании для модуля Payroll.
LOP	Чтение	Слой обновления для слоя LOS. Используется для Service Packs.
BUS	Все	Самый низший слой для доступа партнеров. Партнеры могут использовать этот слой для своих модулей. Слой требует лицензионный код.

BUP	Все	Слой обновления для слоя BUS. Партнеры могут использовать этот слой для своих обновлений.
VAR	Все	Партнеры используют этот слой для своих модификаций, со своей спецификой. Слой требует лицензионный код.
VAP	Все	Слой обновления для слоя VAR. Может использоваться для обновлений.
CUS	Все	Этот слой означает использование модификаций, сделанных клиентами.
CUP	Все	Слой обновлений для слоя CUS. Может использоваться для обновления.
USR	Все	Это самый высший уровень. Этот слой используется, если пользователь создает свои собственные модификации такие, как создание отчета. Часто этот слой используется для тестовых целей.
USP	Все	Слой обновления для слоя USR. Используется для собственных обновлений.

Рисунок 2: Обзор слоев

Пользователь заходит в Ахарта в слое, выбранном в конфигурационной утилите. Вы не можете изменить слой без перезапуска клиента приложения. Все сделанные модификации сохраняются в текущем слое. Все изменения объектов сохраняются в текущем слое.

Изменение таких объектов, как формы и отчеты приведет к записи всего объекта в текущий слой, то есть в слое будут существовать как методы и поля, измененные вами, так и с нижележащих слоев. Если изменяются класс или таблица, то только модифицированные методы или добавленные поля записываются в текущий слой.

Каждый слой хранится в физическом файле с названием AX<layer>.AOD. Название для слоя sys - AXSYS.AOD. Все слои проиндексированы в файле AXAPD.AOI. При удалении его, индексный файл строится заново автоматически при запуске системы. При удалении слоя или добавлении, индексный файл также пересоздается. Если возникает ситуация, когда вы не можете определить объект в АОТ, или системы падает при добавлении объекта, то вам следует удалить индексный файл и перестроить его заново при старте системы. Для перестроения индексного файла все пользователи должны выйти, следует остановить Application Object Server (AOS). Далее необходимо запустить одного клиента в двухуровневой конфигурации и перестроить индексы.

---

**Примечание:** при модификации объекта, изменения сохраняются на верхнем слое и могут быть обновлены. Если форма изменена в VAR слое и сделаны модификации для объекта в USR слое, то будет автоматическое обновление с изменениями VAR слоя. При импорте объекта на нижележащий слой, тот же самый объект не модифицируется в верхнем слое.

---

Если вы установили опцию просмотра всех слоев в **Сервис | Параметры**, то все слои объекта, на которых есть изменения, будут показаны в круглых скобках после имени объекта в АОТ. Это - возможность быстрого просмотра, и

определения слоев модификаций данного объекта. Если в контекстном меню выбранного объекта, имеющем изменения более чем в одном слое, выбрать *Слои*, тогда произойдет разделения объекта по слоям, и он будет показан на различных слоях в АОТ. Вы можете затем просматривать модификации в каждом слое. Однако у вас есть возможность изменять только текущий слой. Кликните *Слои* снова для возврата в предыдущее положение до разделения.

## Свойства

Лист свойств доступен через контекстное меню объекта АОТ, выбором *Свойства* или нажав Alt+Enter. Вы обнаружите, что лучше держать окно свойств открытым так, как лист свойств обновляется всегда при выборе нового объекта в АОТ. По умолчанию, лист свойств прикрепляется к правому краю. Если вы решите закрепить лист свойств в другом месте, то через контекстное меню выберете *Запрет Стыковки*. Свойства представлены на двух закладках. Первая закладка – это все свойства вторая закладка – группы свойств. Если у вас большое разрешение экрана, то вы можете просматривать все свойства объекта без прокрутки. Для просмотра свойств отсортированных по алфавиту, перейдите **Сервис | Параметры** закладка **Разработка**. Для просмотра свойств в АОТ, смотрите главу **Приложение Свойства**.

---

**Примечание:** Лист свойств может использоваться для подсчета объектов. Для этого все узлы должны быть заэкшированы в АОТ. Попробуйте пометить все объекты с префиксом Sales и нажать Alt+Enter. Этим вы заэкшируете выделенные узлы. Если вы теперь пометите только некоторые из заэкшированных узлов, количество отмеченных узлов будет показано в круглых скобках наверху листа свойств. Это будет работать, если все выбранные объекты заэкшированы.

---

Каждое свойство объекта имеет значение по умолчанию. Значения по умолчанию в общем случае – это способность MorphX автоматического ускорения процесса разработки. Это означает что такие объекты, как формы и отчеты имеют предустановленные по умолчанию свойства автоматического расположения полей. При изменении значения свойства по умолчанию, свойство выделяется жирным шрифтом для простоты обнаружения изменений в листе свойств. Если вам необходимо изменить одно свойство на нескольких объектах, то просто выделите эти узлы в АОТ. Так же допускается множественно менять любые типы объектов, но только общие свойства выделенных объектов будут показаны в листе свойств.

При выборе значения для свойства, существует три различных типа иконок просмотра, используемых в листе свойств. Иконка стрелочка, направленная вниз используется для определения значения позиционирования или прикрепления поля. Квадратная иконка используется для переключения между определенным значением и возможностью ввода фиксированного значения.



Рисунок 3: Просмотр иконок, стрелочка вниз и квадратик

Просмотр иконки многоточие используется для открытия новой формы (например, метки) или ввода заголовка или выбор шрифта.



Рисунок 4: Просмотр иконки многоточие

Два типа свойств имеют цвета. Если имя в АОТ для объекта помечается красным цветом – это показатель, что поле обязательно для заполнения. Свойство у названия или заголовка установлено в желтый цвет, если заголовок не определен. Это вовсе не означает, что заголовок должен быть обязательно определен, так как заголовок для форм или отчетов определяется по таблице или расширенному типу данных. Если заголовок выбирается из меточного файла, то желтый меняется на белый цвет.

## Add-ins

Подменю Add-Ins в контекстном меню имеет связь с текущим узлом. Стандартная установка инструментов, таких как Перекрёстные Ссылки и Проверка Best Practices может быть вызвана отсюда. Подменю Add-ins всего лишь пункт меню, расположенный в контекстном меню. Стандартные пункты меню в меню Add-Ins создает MorphX. Вы можете создать ваши собственные пункты меню, используя MorphX, и добавить их в это подменю. Большинство пунктов меню из меню Add-Ins можно вызывать из верхнего меню **Сервис | Средства разработки**.

## Редактор

Редактор в MorphX используется в разработке среды Ахарт на языке X++. Для открытия редактора выберите метод в АОТ и два раза щелкните. Выберите узел методов или верхний узел объекта такого, как класс или форма для открытия списка методов для узла.

Путь АОТ к текущему узлу показан в заголовке окна редактора. Вверху окна редактора располагаются иконки для таких простых задач, как сохранение, компилирование и установка точки останова. Левое окно показывает выбранные методы. Для переключения между методами, щелкните на методе в левом окне. Значок \* ставится после имени метода в левом окне и показывает, что метод изменен и не сохранен. Перед закрытием редактора появится диалог для подтверждения сохранения изменений. Однако если вы установили авто сохранение в **Сервис | Параметры** на закладке **Разработка**, то ваши изменения с интервалами будут сохраняться автоматически.



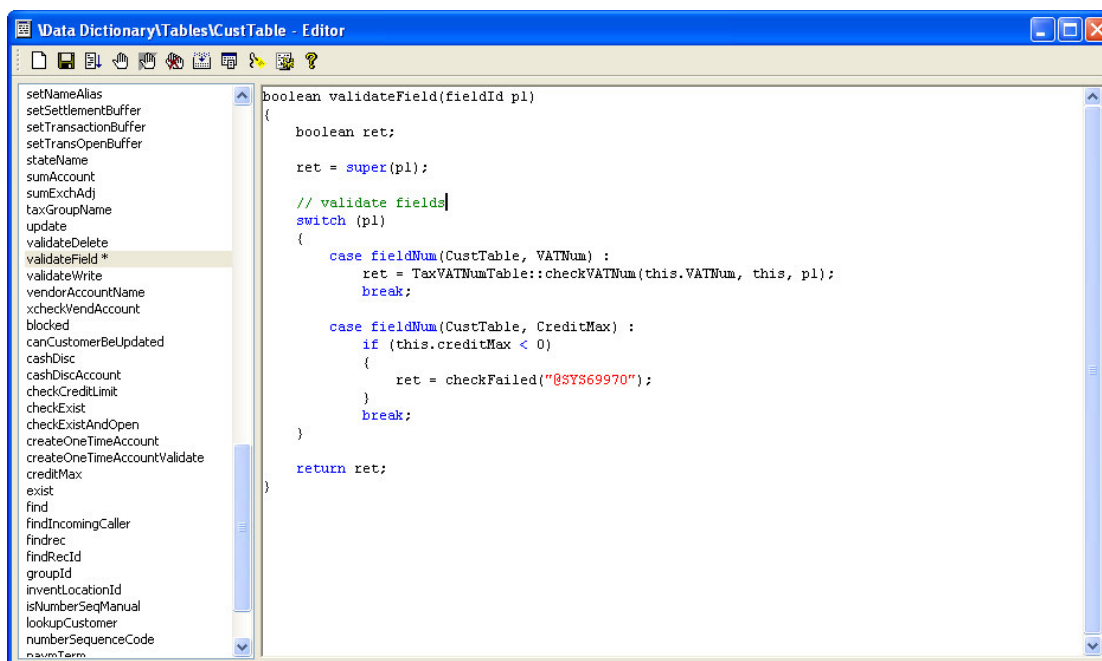


Рисунок 5: Редактор показывает методы таблицы CustTable

### Окно кода

В правом окне показывается код на X++ выбранного метода. Зарезервированные слова помечаются, синим цветом. Комментарии помечаются зеленым, а строка текста подкрашивается красным. Во время написания кода, при вводе не корректного предложения, ошибка подчеркивается красной волнистой линией. Также и метод с ошибкой при компиляции будет подчеркнут в левом окне. Для просмотра наиболее важных горячих клавиш в окне кода смотри **Рисунок 6: Горячие клавиши редактора**. Полный список всех горячих клавиш можно найти в Руководстве Разработчика, расположенного в **Справка | Microsoft Axapta и Руководство Разработчика**.

При нажатии правой кнопки мыши в окне кода, вы обнаружите меню, из которого вы можете просматривать список объектов АОТ, просматривать определение методов или вызвать scripts редактора. Список объектов АОТ, таких как таблицы, классы или расширенные типы данных показываются по имени так, что у вас не уйдет много времени на поиск объекта в АОТ. Вы просто выбираете имя из списка. Альтернатива использованию списка - открытие второго окна АОТ и перетаскивание объекта из второго окна в окно кода. Действительно вы можете перетащить любой узел из АОТ в окно кода. Имя объекта вставляется в окне кода. Однако некоторые узлы добавят строчку кода при перетаскивании. Попробуйте перетащить запрос из АОТ в окно кода. Вы получите готовый код, написанный для выполнения вашего запроса. Только необходимо объявить переменные.

Пункт *Просмотр определения* используется для перехода к коду метода или просмотра метки в меточном файле (label system). Если ваш метод компилируется с ошибкой, то вы не сможете использовать этот пункт меню.

Scripts редактора – это коллекция scripts, сделанных на X++. Они используются для выполнения различных задач таких, как добавление комментариев или формирование кода специальным выражением. Вы можете добавить ваши собственные scripts или модифицировать существующие. Класс *EditorScripts* имеет метод для каждого script.

Функция	Горячие клавиши	Описание
New	ctrl+n	Создание нового метода.
Save	ctrl+s	Сохранение всех методов в левом окне.
Toggle breakpoint	F9	Установка точки останова.
Enable/disable breakpoint	ctrl+F9	Используется для перехода через точку останова без удаления.
Remove all breakpoints	ctrl+shift+F9	Сброс всех точек останова пользователя.
List breakpoints	Shift+F9	Список всех точек останова.
Compile	F7	Компиляция всех методов в левом окне.
Lookup Properties/Methods	ctrl+space	Показывается желтая подсказка. В зависимости от кода может показываться: базовый тип расширенного типа, профиль параметра метода или текст заголовка для метки.
Lookup Label/Text	ctrl+alt+space	Если метка выделена, корреспондирующая метка и текст метки показывается из меточного файла.
Lookup Definitions	ctrl+shift+space	Откроется выбранный метод в новом окне редактора. Не требуется отмечать имя метода. Если метод не переопределен, нового окна не откроется.
Script	Alt+m	Откроется меню script.
List Tables	F2	Список всех таблиц.
List Classes	F12	Список всех классов.
List Types	F4	Список всех расширенных типов.
List Enums	F11	Список всех перечислений.
List Reserved Words	shift+F2	Список всех зарезервированных слов.
List Built-in Functions	shift+F4	Список всех функций.

**Рисунок 6: Горячие клавиши редактора**

## Отладчик

Отладчик запускается при установке точки останова в коде программы и запуске программы. Если отладка производится в трех уровневой конфигурации, вы должны быть уверены, что активирована отладка в AOS, иначе вы сможете отлаживать код только на клиенте. Во время отладки клиент Ахарта блокируется. Если вам нужен доступ к клиенту во время отладки, то вы можете запустить другого клиента Ахарта. Однако отладчик будет связан с клиентом из того места, откуда он активирован. Точки останова используются в разрезе пользователей, поэтому вы можете не беспокоиться о других пользователях использующих отладчик. Для получения списка текущих точек останова нажмите Shift+F9 в любом месте кода в AOT.

---

**Примечание:** Если у вас появляется ошибка в Infolog и вам нужно найти ошибку с помощью отладчика, то установите точку останова перед появлением сообщения в Infolog. Для этого перейдите к методу класса `Info.add()` и установите точку останова в коде на первой строчке кода после объявления переменных.

---

В верхнем окне будет показан отлаживаемый код. Код представляется так же, как в редакторе. Точка останова может быть установлена в редакторе кода и в отладчике. Горячие клавиши для их установки представлены списком на **Рисунке 6: Горячие клавиши редактора**. Точки останова выделяются красной чертой в редакторе. В отладчике точки останова показываются красным кружком с левой стороны. Желтая стрелка слева показывает текущий курсор. Инструменты навигации помещены в верхней части окна отладчика. У вас имеется 4 различных окна, которые используются для отладочных целей. У этих окон может меняться расположение и размеры.

---

**Примечание:** Не забывайте закрыть отладчик после его использования. Во время отладки вы блокируете таблицы с отладочным кодом. Не очень хорошо, если пользователь попытается занести данные в заблокированные таблицы и получит сообщение об ошибке базы данных.

---

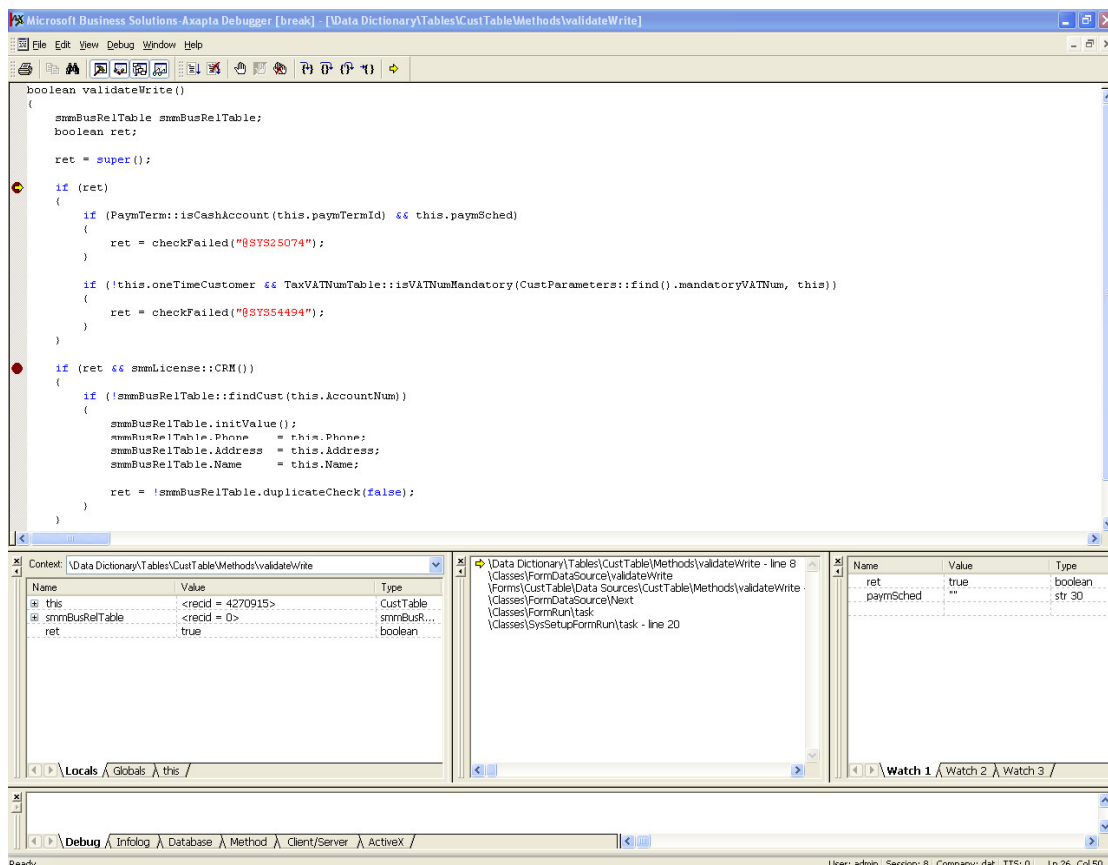


Рисунок 7: Отладчик

### Окно вывода сообщений

Окно вывода по умолчанию находится внизу отладчика. Здесь вы можете просматривать сообщения Infolog и команды печати. Это окно можно использовать для отображения текста, используемого для целей отладки.

### Окно переменных

Это окно самое левое под окном отладчика. В нем выводится список переменных текущего стека вызовов. Все типы переменных таких, как поля таблиц, классов представлены в списке. Переменные, измененные между двумя точками, будут в самом верху. Поле просмотра в верхней части окна переменных используется для просмотра метода, откуда идет вызов. При выборе метода, окно переменных и окно отладчика обновляются вместе с выбранным методом.

### Окно стека вызова

Окно стека вызова обычно располагается в центре под окном отладчика. Окно дает обзор вызываемых методов. Вы можете кликнуть по любому методу в стеке вызова для перехода к просмотру кода метода. При переходе так же обновляется и

окно отладчика, и окно переменных. Отметим, что вы не можете просматривать не редактируемые системные методы. Желтая стрелочка в левой части показывает текущий метод отладки, а при переходе к другому методу зеленый треугольник отмечает текущий выбранный метод.

### Окно просмотра

Это самое правое окно в ряду под окном отладчика. Это окно используется для просмотра значений переменных, которые вы выбираете вручную. Окно имеет три похожие закладки, которые помогают организовать информацию, при трассировке большого количества переменных. Для добавления переменной в это окно, выделите её в отладчике и перетащите. Используйте клавишу DEL для удаления переменной. Так же как и в окне переменных, вы можете менять значение переменной, добавленной в окно просмотра. Пока переменная находится в стеке, показывается её значение. Если переменная не в стеке, показывается ошибка, как значение переменной. Если переменная изменяется между двумя точками останова, выводится последнее значение. Переменные, установленные в окне просмотра сохраняются после компиляции. Это ускоряет отладку так, как вам нет необходимости вносить переменные заново при отладке одного и того же кода несколько раз.

### Окно сообщения компилятора

При компиляции кода X++ из редактора кода или узла в АОТ, активируется окно компилятора. По умолчанию окно компилятора – это нижнее окно. Такое расположение настроено в соответствии со стандартом, так как все окна открываются в Axapta Integrated Development Environment (IDE). Вы можете кликнуть правой кнопкой мыши в окне компилятора и выбрать *Запрет Стыковки*, если вы хотите, что бы окно компилятора было не закрепленным, перемещаемым окном. Это хорошо разгружает рабочее место, особенно при просмотре листа свойств.

Вы можете использовать вместо окна компилятора окно сообщений для просмотра результата компиляции. В версии 3.0 было введено окно компилятора. Окно сообщений использовалось в предыдущих сериях. Перейдите к меню настроек окна компилятора по кнопке *Настройки*. Поле *Носитель* определяет, использовать ли окно компилятора или окно просмотра. Вы можете кликнуть по строчке в этом окне для просмотра информации. Если вы выбрали окно просмотра для вывода информации, вы можете перевыбрать назад окно компилятора в верхнем меню в **Сервис | Параметры** кнопка *Компилятор*. Уровень проверки компиляции можно установить в окне установок. Поле *Уровень диагностики* определяет содержимое сообщений. Если установить Уровень 4, то будет включена проверка хорошего тона программирования (best practice). Перекрестные ссылки обновятся. Помните, что перекрестные ссылки замедляют компиляцию. За информацией о перекрестных ссылках смотри **Приложение**

**MorphX Tools.** Можно вести журнал компиляции. Однако вы можете просто экспортировать результат вашей компиляции из окна компилятора, а затем импортировать и использовать эти результаты (ошибки и предупреждения) в окне компилятора.

Окно компилятора состоит из 4 закладок. Первая закладка показывает обзор компиляции. В зависимости от настроек будут выводиться предупреждения, ошибки, ошибки best practice, будут рассчитаны задачи. Рекомендуется компилировать целиком приложение перед отправкой модификаций. Во время компиляции вы можете увидеть множество ошибок и предупреждений. Не беспокойтесь, это нормально. Полная компиляция состоит из трех циклов и все объекты не распознаются до финального цикла.

Ошибки и предупреждения компиляции можно увидеть в списке на второй закладке. Список содержит также ошибки и предупреждения, найденные в методах листа свойств. Ошибки помечаются красным символом, а предупреждения желтым. Окно компилятора – это стандартная форма Ахapta так, что доступны стандартные возможности по сортировке и фильтрации. Если два раза кликнуть по ошибке или предупреждению в списке, то откроется метод или свойство, содержащее ошибку. Затем вы можете исправить ошибку или предупреждение, сохраните изменения и закройте окно. После того, как ошибки и предупреждения исправлены, они удаляются из списка.

Если проверка best practice активна, то отклонения по best practice будут представлены списком на третьей закладке. Нажмите кнопку *Настройки* и выберите *Best Practices* в окне компилятора для выбора проверки кода по best practice. Отметим, что проверка best practice рассматривается как рекомендация. Используйте здравый смысл при проверке вывода сообщений. Пропущенные заголовки и использование базовых типов вместо расширенных типов теперь легко определяется. Не следует бездумно следовать проверке best practice, если вы не предполагаете результат ваших изменений.

Последняя закладка используется для отслеживания ваших задач. Если вы расположите текст TODO в заголовке методов, то название метода появиться на закладке задач при компиляции. Это приятная возможность так, как помогает запомнить место в коде, которое следует проверить пред отправкой модификации.

---

**Примечание:** В меню Add-ins вы обнаружите пункт меню, названный *Инкрементная Компиляция*. Этот пункт меню доступен на классах, и компилирует все наследуемые классы. Это быстрый путь для компиляции наследников перед тестированием ваших модификаций.

---


## Импорт и Экспорт

У вас имеется две возможности для перемещения ваших модификаций от одной системы к другой. Или копировать весь слой в файл или экспортировать проектом. Каким случаем воспользоваться зависит конкретно от вашего случая.

Копирование целого слоя в файл используется чаще при обновлении приложения у заказчика, так как это единственная возможность, если заказчик не имеет необходимой лицензии на разработку. Для экспорта узла выберите в контекстном меню пункт *Экспорт*. Вы можете экспортировать объекты узла такие, как таблицы, расширенные типы данных, формы или классы, однако методы не могут экспортироваться отдельно от объектов. Вы можете экспортировать несколько объектов за раз, пометив объекты на экспорт. Появится диалог выбора пути экспорта. Здесь вам следует ввести имя экспортируемого файла. Текущий слой проставляется в экспорте по умолчанию. У вас есть возможность экспортировать другой слой, выбрав слой в диалоге. Рекомендуется делать следующее: выбирать слой, даже если вы в слое который экспортируете, таким образом, вы уверены в правильности экспортируемого слоя. Названия полей или как ещё их называют метки, используемые в вашем коде, так же можно экспортировать. Отметим, что импорт меток потребует определенных знаний о меточной системе, поэтому эта опция не рекомендована, если файл будет импортирован пользователем, не имеющим достаточных знаний. Однако возможность экспортирования меток вместе с кодом имеется. Это может быть причиной получения файла без меток.

Экспортируемые объекты можно блокировать на время экспорта. Блокированные объекты помечаются иконкой замочной скважины. Это - хорошая возможность, но не ожидайте многого от этого так, как блокированные объекты могут быть изменены другим разработчиком и каждый может через контекстное меню заблокировать или разблокировать объект. Часто бывает трудно определить, кто блокирует объект и почему.

Все узлы объектов таблиц, расширенных типов и классов имеют уникальный индекс (ID) в АОТ. Индекс – это последовательный номер объекта в текущем слое. Этот индекс так же может включаться в экспортный файл. Эта опция затем может быть использована для импорта и последующей синхронизации индексов.

Для импорта файла кликните на иконке Импорт  в панели инструментов АОТ. Только файлы с расширением ХРО будут доступны для импорта. Файл импортируется в текущий слой. Для проверки текущего слоя, взгляните вниз на строку состояния в окне Ахарт. Для просмотра импортируемых объектов поставьте галочку в «Отобразить Подробности» в диалоге. Дерево похожее на АОТ будет сгенерировано импортируемыми объектами. Существующие объекты в АОТ отмечаются жирным шрифтом. По умолчанию все объекты из файла импортируются. Вы можете изменить эту опцию, убрав галочку в дереве. Для объектов уже существующих в АОТ, вы можете сравнить объект в файле с тем же объектом в АОТ. Это хорошая возможность так, как вы можете выбирать: обновить объект или нет. Для описания инструмента сравнения, смотри главу **Сравнение объектов**. При выборе импорта меток, показывается три дополнительных закладки в окне. Здесь вы определяете язык импортированных меток. Метки в импортном файле представлены в списке. Вы получите сообщение, если метки уже существуют в приложении. Для каждой метки вы можете выбирать: импортировать метку или создать новую. По умолчанию файл

меток, используемый для создания новых меток, показывается в верхнем правом углу диалога. Меточный файл по умолчанию устанавливается меточной системой. За дальнейшей информацией о меточной системе смотри главу **Меточная Система**.

Перед выбором опции удаления таблицы или класса из списка импортируемых объектов, вы должны понимать концепцию целиком. Если экспортируются таблица или класс, а они существуют в другом слое, то только модифицированные методы и свойства таблиц и классов хранятся в экспортированном слое, а не целый слой объекта. При выборе удаления таблицы или класса из АОТ, только модифицированные методы и свойства таблицы или класса удаляются из АОТ из текущего слоя, то есть на других слоях все остается, как есть. Это означает то, что класс, который уже существует в АОТ и имеет модифицированные методы в файле, приобретет новые методы после импорта. Это распространяется на слои, к которым у вас есть доступ. Вы не можете, например, удалить слои SYS или GLS.

Если вы хотите экспортировать файл со значениями идентификаторов, то поставьте галочку в соответствующем поле. Это означает, что при последующем импорте вы поддерживаете текущую идентификацию объектов АОТ. Если Вам необходимо проводить восстановление данных, при импорте объектов с идентификаторами, то сначала необходимо сделать импорт новых объектов без идентификаторов. Вы получите сообщение, если изменяется идентификатор таблицы, содержащей данные, и вы сможете отменить восстановление данных, если не уверены.

Отметьте опцию *Переписать блокированные прикладные объекты*, если вы не уверены будут ли объекты блокироваться.

## Сравнение объектов

Сравнение объектов можно сделать как при импорте файла ХРО, так и через контекстное меню на объекте, который существует в более чем одном слое и выбрать *Сравнить* из Add-Ins меню. При выборе сравнения в двух слоях, система покажет окно сравнения. При нажатии кнопки Сравнение для выбранного одного слоя вы можете заметить, что те же самые слои присутствуют в списке дважды, с пометкой на одном как old layer. Слой Old— это слой, хранимый в приложении в папке old. За информацией по структуре файлов Ахарта смотри инструкцию в стандартной поставке. Сравнение со слоем old необходимо, если вы произвели обновление приложения новым пакетом обновления, и хотите проверить, какие модификации сделаны.

Окно сравнения разделено на две панели. Слева у вас есть дерево просмотра объекта. Справа показывается результат сравнения для выбранного узла в левой панели. Результат сравнения имеет красный и синий цвета. Цвета используются



для показа отличий между двумя слоями. Только узлы, где есть отличия, показываются в дереве. Различия в каждом слое выделены соответствующим цветом. В дереве синий или красный показывают, что узел существует в представленном цветом слое. Двух цветовая иконка показывает, что узел изменен в обоих слоях. На **Рисунке 8: Сравнение двух слоев объекта**, сравниваются два слоя формы CustTable. В окне сравнения, строки кода, которые существуют только в одном слое, подсвечиваются цветом слоя. Стрелочки слева или справа показываются в конце подсвеченной строки кода или блока кода. Кликнув по стрелочке, вы можете добавить или удалить код формы текущего слоя. Этот инструмент не только сравнивает код, но так же и значения свойств объекта. Подобно изменениям в коде, изменения в свойствах между слоями можно изменять при использовании стрелочек для обновления текущего слоя. Конечно, существуют ограничения и с инструментом сравнения: если много изменений сделано между двумя слоями, может быть лучше переписать объект при обновлении, чем разрешать коллизии в строках кода.

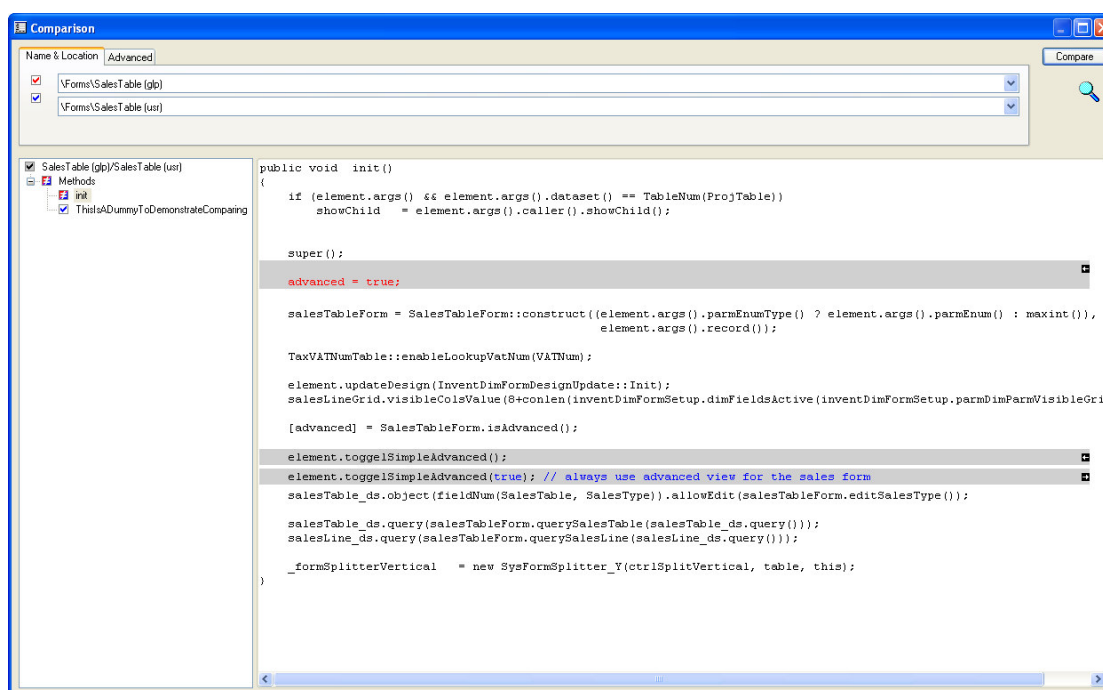


Рисунок 8: Сравнение двух слоев объекта

## Обновление кода

Инструмент Обновление Кода доступен так же из меню Add-Ins. Инструмент сравнения объекта может использоваться для сравнения объектов любого типа, инструмент обновления кода специализируется на сравнении методов. Этот инструмент весьма удобен для сравнения старого слоя с новым слоем, при установке service pack или обновлении версии.

**Рисунок 9: Обновление кода** показывает класс, который был обновлен в нескольких слоях. Вы можете видеть список методов с левой стороны. Измененные методы в сравнении со старым слоем, так и методы на старом слое будут представлены наверху. Например, метод, существующий на SYS слое старой версии, и измененный в новом SYS слое. Когда вы кликните на методе, измененном в более чем одном слое, Ахapta обновит правое окно с закладкой метода модифицированного слоя. Первая закладка, по умолчанию, показывает верхний уровень. Из Рабочей области вы можете править код метода. Предпочтительно открывать и править код в редакторе по кнопке *Правка*. В подменю *Предложение*, кнопка слияние будет активна, если выбран соответствующий метод. Вы можете использовать кнопку *Слияние* для слияния кода из всех слоев в рабочую область. Это не затронет ваше обновление, но поможет иметь весь код в одном месте. Слои модифицированного метода будут представлены в подменю *Предложение*. Инструмент обновления кода слоев служит для принятия или отклонения обновления. Кликните по названию слоя в Рабочей области закладки. Кнопка *Сравнить* используется для сравнения метода в двух слоях. Сравнимый код в слоях показывается на новой закладке разным цветом. Для выделенного метода, закладки сравнения кода автоматически создаются для сравниваемых слоев с подкрашенными отличиями.

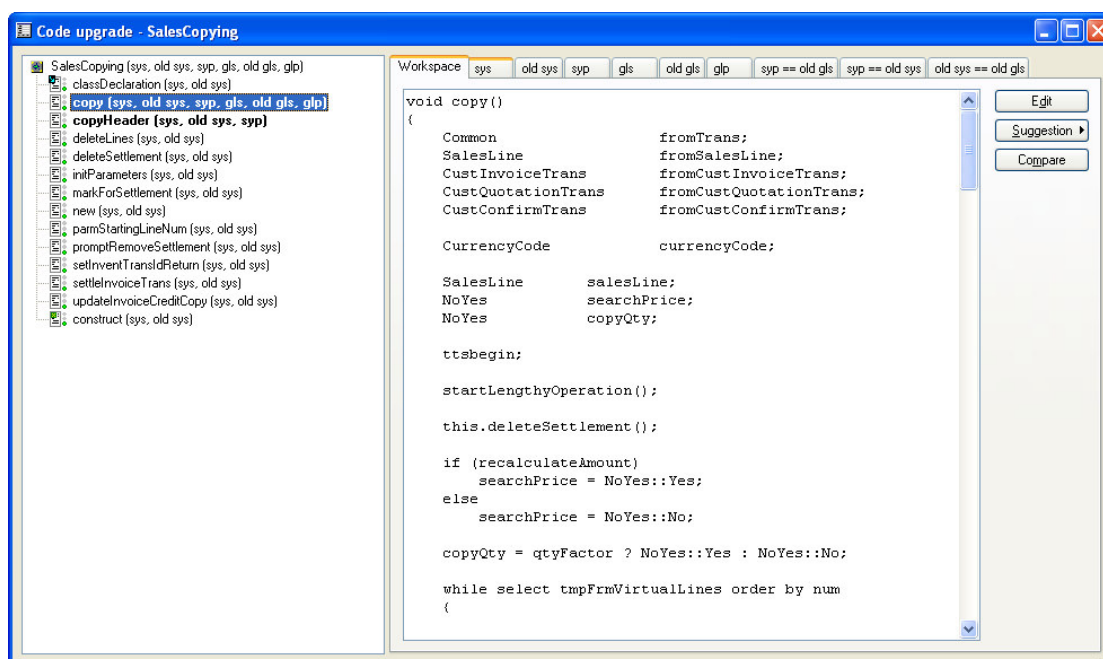


Рисунок 9: Обновление кода




## Поиск

Поиск объектов по АОТ можно выполнять выбором узла или комбинацией клавиш Ctrl+F. Появится диалог поиска по методам текущего активного узла дерева. Найденные методы выводятся списком в табличку с указанием пути в АОТ. Двойным кликом вы можете открыть метод. Вы можете изменить поиск,

добавив поиск по свойствам. Для этого надо изменить критерии поиска, выбрав *Все Узлы*, и закладка Свойства станет видна. Закладка Свойства включает все свойства, используемые в АОТ, отсортированные по имени. Если вы хотите найти формы, использующие специфическое свойство, поставьте галочку напротив свойства и введите значение свойства, которое вам необходимо найти. Например, найти все формы, имеющие свойство AlwaysOnTop, отметьте свойство и введите Yes в критерий поиска для свойства. Формы, у которых установлено данное свойство, будут представлены списком в табличке, внизу формы. Отметим, что вы не можете перейти к листу свойств, просто кликнув мышкой. Вместо этого через контекстное меню правой кнопки выберите *Свойства* из меню Утилиты.

Вы можете осуществлять поиск и по методам, нажав Ctrl+F. Функция поиска по методам подобна функции поиска в Microsoft Word. Однако если вы собираетесь найти и заменить код, например, изменить переменные, то вам лучше сначала воспользоваться инструментом перекрестных ссылок для получения информации об использовании объекта. За дополнительной информацией по перекрестным ссылкам, смотрите **Приложение Среда разработки MorphX**.

## Infolog

Информационное окно используется для вывода сообщений пользователю. Вы используете Infolog для информирования пользователя о совершаемых действиях, например, совершении ошибки или выполнении определенного задания. Infolog автоматически открывается в отдельном окне при вызове из программы. Информация из Infolog удаляется при закрытии окна. Если вам нужна выводимая информация, то вы можете распечатать содержимое из окна, используя **Файл \ Печать**. Информация в Infolog может выводиться как из кода, так из ядра системы. Информация из ядра обычно затрагивает проблему целостности системы такую, например, как информация о заполнении полей по умолчанию. Из X++ вы можете выводить в Infolog контрольные значения переменных. Существует три типа сообщений в Infolog. Типы определяются по соответствующим иконкам: info , warning  и error . Текст Info обычно выводит информацию о результате выполняемых пользователем действий в системе. Warnings и errors обычно предупреждают о неправильных действиях пользователя и предупреждают, что процесс не возможно выполнить. Вы можете прикрепить информационную страницу или действие к Infolog при двойном нажатии на строчке в Infolog, как показано ниже.

```
static void Intro_Infolog(Args _args)
{
    int i;
;
    info("This is an info.");
    warning("This is a warning.");
    error("This is an error.");

    setprefix("prefix text");
}
```

```

for (i=1; i<=3; i++)
{
    setprefix("1. for loop");
    info("loop 1");
}

for (i=1; i<=3; i++)
{
    setprefix("2. for loop");
    info("loop 2");
}

info("Check customer parameters.", "",
    SysInfoAction_Formrun::newFormname(formStr(CustParameters),
        identifierStr(Customer_defaultCust), ""));
info("Check the sales form help page.", "ApplDoc://Forms/SalesTable");

throw error("This error stop execution.", "");
}

```

Это пример использования Infolog из X++. Первые три строчки показывают пример использования info, warning и error. Использование типа сообщений warning() и error() обычно ставится в операторе throw, как показано в последней строчке так, как throw останавливает любые действия, например, обновление записи.

Если вы выводите много сообщений в Infolog, то следует использовать функцию setprefix() для организации более читабельной информации. Рассмотренный пример показывает, как вы можете структурировать информацию, например, сделать отдельную группу в Infolog. Префиксы группы, выводящие информацию в информационное окно, каждого цикла представлены различными значками. Добиться подобного можно и несколько другим путем. В различных уровнях вложенности setprefix() не всегда корректно отображается. Вот пример такого использования.

```

static void Info_LevelText(Args _args)
{
    ;
    info('n\t1\ttext1');
    info('n\t1\ttext2');
    info('n\t2\ttext3');
}

```

Вывод в Infolog в предыдущем примере имеет три параметра. Последние два параметра – опции, использующие связь с полем формы или связь со страницей справки. Последние две строчки info() показывают, как использовать параметры вывода сообщений. Первая строчка показывает, как привязать сообщение к форме параметров клиентов. Система откроет эту форму, при двойном клике на сообщении в Infolog. Последнее сообщение info() выводит страницу справки о

заказах. Отметьте, что параметры используются только один раз, вы не можете привязать и форму, и справочную систему. Использование связей в Infolog имеет более наглядный интерфейс для пользователя, при возникновении ошибок. Вы можете снабдить пользователя информацией, где именно произошла проблема или сообщить дополнительную информацию, как разрешить проблему. К сожалению, при изменениях объектов эти связи не обновляются, так если вы изменили поле, на которое ссылались из info, то вам необходимо вручную изменять и ссылку.

Infolog имеет ограничение по размеру выводимых сообщений в 10,000 строк. Сообщения Infolog не следует использовать для детализированного отчета по представлению разности. Вы можете изменить максимальный размер Infolog, но помните, что Infolog не предназначался для работы с большими объемами данных. Построение Infolog с несколькими тысячами записей займет время и производственные ресурсы, поэтому вам лучше использовать отчет.

Один из фундаментальных классов в Ахapta – это класс Info. Класс Info работает с Infolog. За дальнейшей информацией по фундаментальным классам смотри главу **Классы**.

## Мусорная корзина

Мусорная корзина АОТ располагается в верхнем меню **Файл | Открыть | Мусорная корзина АОТ**. Мусорная корзина может быть использована для восстановления таких объектов, как таблицы, расширенные типы данных или формы. Части объектов, как поля таблиц или методы формы, не могут быть восстановлены. Вы можете только восстановить удаленные объекты для текущей сессии так, как мусорная корзина очищается при закрытии клиента Ахapta. Удаленные объекты показываются в списке, отсортированном в порядке удаления объектов так, что последний удаленный объект показывается первым. Если объект с тем же самым именем удален дважды, объект появляется в списке два раза. Запомните: если таблица уже удалена, то при восстановлении в ней не будет данных.

## Настройки пользователя

Форма располагается в меню **Сервис | Параметры** и используется для настроек пользователя, под которым производится вход в систему. Она включает и общие настройки и настройки, которые используются в разработке. Для загрузки настроек по умолчанию используйте кнопку *По умолчанию*. За дальнейшей информацией по кнопке *Компилятор*, смотри секцию **Окно сообщений компилятора**.

### Разное

Закладка *Разное* используется для определения таких базовых настроек, как имя и пароль и где вы устанавливаете сетевое имя пользователя. Так, при работе в сети, ваш login автоматически определится в Aхapta.

### Строка состояния

Закладка строка состояния определяет информацию, которая отображается в строке, в нижней правой части окна Ахapta. Поле *Прикладной слой* наиболее важное среди полей на этой закладке: при включении этого поля, отображается текущий прикладной слой. *Код компании* – также важное поле и выбирается по умолчанию. Когда это поле отмечено, Ахapta выводит предупреждение о смене текущей компании.

### Шрифты

Вы можете установить шрифт по умолчанию и его размер на закладке *Шрифты*. Обычно стандартные настройки достаточны. Однако вы можете использовать специальную настройку шрифтов для ваших отчетов. Это глобальные настройки. Если вы хотите настроить специфические шрифты для конкретных форм или отчетов, то вам следует изменить соответствующие свойства каждого объекта.

### Разработка

Не удивляетесь, но существует и закладка *Разработка* – специфический интерес разработчиков. Она оптимизирует настройку среды разработки. В группе *Разное* вы можете выбрать проект для загрузки при старте системы. Это очень необходимо, когда вы разрабатываете проект длительное время, и вы не хотите тратить время на поиск и открытие его вручную при старте вашего клиента. За дальнейшей информацией по проекту смотри секцию **Проекты**. Поле *Слой прикладных объектов* используется для установки показа изменений слоев в АОТ. Опция *Показать все слои* дает информацию обо всех измененных слоях объекта и поможет вам понять, какие объекты, на каких слоях хранятся.

Если вы хотите использовать окно сообщений для вывода информации трассировки или компиляции, то установите лимит о выводе сообщений. При отметке поля *Сообщения для разработчиков*, система генерирует большое число сообщений, большая часть которых весьма сомнительного значения такие, например, как информация о выборке без индекса.

Группа полей *Редактор* используется для настройки вставки или добавления текста. Если поле *IntelliSense* отмечено, тогда в методах текущего объекта редактор выводит список свойств или методов объекта, при установке точки после имени объекта. Это бывает очень необходимо при разработке и обычно это поле оставляется отмеченным. Это сохраняет ваше время на написание кода.

Например, при установке точки после названия класса, методы класса будут представлены в списке автоматически. IntelliSense также подсвечивает желтым цветом информацию о базовом типе при наведении и нажатии **Ctrl+Пробел** на переменную в методе редактора.

Поле *Отладка* по умолчанию устанавливается, когда есть точка останова. Это означает, что при установке точки останова в методе, отладчик автоматически загружается при прохождении через метод.

Группа полей *Автоматически* используется для автоматического сохранения модификаций объекта. Если установить галочку в поле *Автосохранение*, объекты будут сохраняться через определенный промежуток времени. Если установлен флажок *Автообновление*, то объекты, созданные и измененные другими разработчиками автоматически обновятся. Это особенно необходимо при разработке приложения несколькими разработчиками, когда один из программистов создает новую таблицу или класс. Новый объект затем становится автоматически доступным через определенный промежуток времени. Если не использовать *Автообновление*, то вам необходимо перезапускать клиента Ахпта для просмотра изменений другими программистами в АОТ. Размер сборщика мусора так же необходимо определять, так как это число показывает максимальное число изменений, которое необходимо держать в памяти. Увеличивая это число, вы затрачиваете больше объема памяти и загружаете лишний раз CPU, как следствие система будет собирать мусор медленнее.

Вы можете установить опции трассировки запросов к базе данных, методов, клиент/сервер и вызовов ActiveX. Окно сообщений будет использоваться для настроенных опций трассировки. Отметьте, что при вызове этого метода будет генерироваться огромное число строчек в окне сообщений.

Если вы желаете, что бы лист свойств был отсортирован по алфавиту, установите галочку в поле *Сортировка в алфавитном порядке*. Более подробную информацию по свойствам смотри секцию **Свойства**.

## SQL

Опция трассировки базы данных устанавливается на закладке *SQL*. Отметьте поле *Мониторинг запросов SQL* для производства трассировки. Вы можете определить, куда выводить результат трассировки в окно сообщений, Infolog, базу данных или записывать в файл. Вам следует использовать опцию трассировки для оптимизации вашего кода. Вы можете распечатать информацию о трассировке, используя окно сообщений или Infolog. Вы получаете быстрый доступ к коду. Двойным кликом на строчке кода откройте форму мониторинга запроса, показывающую информацию о трассированной строчке. Эта форма позволит вам сразу править код.

### Подтверждение

Настройка подтверждения очень важна для пользователя. Различные поля относятся к различным группам полей. Более подробная информация о группе полей находится в главе **Словарь Данных**. По умолчанию установлены все галочки в группе полей подтверждения удаления. При тестировании приложения, важно, чтобы настройки пользователя были те же самые.

### Кэширование таблиц

Это список таблиц, где все записи определенных таблиц кэшируются. Для установки кэширования таблицы смотри секцию **Словарь Данных**. Таблицы, на которые часто ссылаются можно установить с упреждающей загрузкой. Поскольку загрузка данных из больших таблиц занимает некоторое время, для некоторых таблиц вы можете ее запретить. Этот список установлен по умолчанию и всегда должен проверяться для каждой инсталляции Aha!pt. Если запретить упреждающую загрузку, то в первый раз время на кэширование всей таблицы не потребуется, но общая производительность снизится.

### Использование данных

Кнопка *Использование данных* показывает хранение настроек для таких объектов, как формы, отчеты и классы. При изменении пользователем вывода формы, изменения формы запроса или ввод значений в диалог отчета, настройки пользователя сохраняются. Эти данные хранятся в системной таблице SysLastValue. Сохраняется только одна запись для объекта и пользователя, так как только последнее измененное значение сохраняется. Если вы хотите посмотреть использование данных для всех пользователей, тогда вы можете вызвать форму использования данных **Сервис | Средства разработки | Объекты приложения | Использование данных**.

Форма использования данных показывает содержимое системной таблицы SysLastValue для текущего пользователя, разделенная на закладки для каждого типа объекта. На закладке разное имеется кнопка, удаляющая все содержимое SysLastValue для текущего пользователя. Это весьма удобно при тестировании вашей модификации, так как вы можете начать тестирование с теми же настройками, что и у пользователя формы или отчета. Тестирование модификации с сохраненными пользовательскими настройками для объектов – это источник ошибок, так как объект может действовать отлично от ожидаемого, например, при добавлении критерия в запрос. Закладка *Все данные* представляет список объектов со всех закладок формы использования данных. Отметим, что эта закладка так же выводит список и для классов.


### Best Practice

Эта кнопка вызывает форму, которая позволяет разработчику устанавливать опцию настройки хорошего тона программирования. По умолчанию все



параметры включены, но вы можете выключить некоторые специфические настройки. Изменяя параметры best practice можно оставить всего одну проверку на пропущенные метки. Более подробную информацию по best practice, смотри секцию **Окно сообщения компилятора**.

## 1.2 Проект

Окно проекта в АОТ открывается по нажатию иконки Проект  в верхнем меню. Отметим, что он всего лишь ссылается на Проекты в АОТ.

Проект используется для группировки объектов в АОТ. Объекты в проекте действуют так же, как и в АОТ, это означает, что у вас есть доступ к тому же самому листу свойств, при нажатии правой кнопки на объекте. Объект может существовать в любом количестве проектов, объекты «проживают» в АОТ. Вам следует помнить, что объекты в проекте – это всего лишь ссылки на объекты в АОТ. Если объект проекта удаляется или переименовывается в АОТ, то ссылка на него в проекте все еще существует, но иконка объекта меняется. Проекты не обладают возможностью обновления, например, как поля формы и служат только для группировки ваших объектов. В проекте у вас имеется опция просмотра модифицированных объектов для реализации дальнейших действий. Организацией объектов по проектам в приложении, вы облегчите процесс обновления.

---


**Примечание:** Если вы импортировали пример из этой книги, то вы увидите лист свойств с префиксом MORPHXIT.

---

### Изменения в проекте

Окно проекта показывает хранимые объекты в списке. Для открытия проекта, кликните два раза по узлу проекта. Выбранный проект откроется в новом окне. При добавлении объекта в проект, вы можете воспользоваться перетаскиванием из узла АОТ или через контекстное меню, выбрав *Создать* и выбрав тип узла АОТ. Лист доступных узлов включает и специальный тип узла *Group*, который используется для группировки объектов по типу, как это реализовано в АОТ. При создании группы, вам необходимо определить тип объектов, которые будут содержаться в группе в свойстве **ProjectGroupType**. Вы можете установить тип группы, например, Tables или Forms. Установкой типа группы меняется вид иконки группы в соответствии с иконкой аналогичной группы в АОТ. Когда тип установлен, только объекты определенного типа могут быть добавлены в эту группу. Вы, наверное, уже замечали, что узел групп имеет свойство **GroupMask**. Это свойство используется для фильтрации или выбора объектов этой группы. Это обычно делается при инициализации установки проекта, так как использование group mask сообщает системе переместить содержимое группы с теми объектами в АОТ, которые помечены специфическим критерием. Например,

если свойство **ProjectGroupType** установлено в Forms, а в свойстве **GroupMask** поставить значение Asset, то группа будет включать все формы, в названии которых будет Asset.

Панель инструментов в верхнем меню окна проекта очень похожа на меню в АОТ. Отличие в иконке фильтра  не доступной в меню АОТ. Иконка фильтр используется для создания проекта, основанного на выбранных настройках фильтра. Объекты проекта могут быть сгруппированы в АОТ, выбором в поле АОТ в группе полей Группировка. Кликнув по кнопке *Выбор* в диалоге фильтра, вы можете выбрать какой объект АОТ включить. Опции фильтра – просмотр системной таблицы *UtilElements*, которая содержит информацию о каждом объекте в АОТ. Например, если вы хотите создать проект, который будет содержать все измененные объекты в определенном слое, тогда установите критерий для поля *UtilElements.utilLevel*.

### Типы проектов

Существует две группы проектов в АОТ: private и shared. Проекты private видны только для пользователя, создавшего проект. Shared проекты видны всем пользователям. Вам следует использовать только private проекты для тестовых целей или подобных задач. Использование shared проекта означает, что более чем один разработчик может работать с проектом в одно и то же время. Для создания проекта, кликните правой кнопкой мыши на узле private или shared и выберете *Создать*.

Помимо private и shared, существует третий тип проектов. Первый тип проекта просто называют *project* – это тип проекта по умолчанию. Этот проект создается при нажатии Ctrl+N на узле проектов. Другие два типа проектов, Web проект и Справка. Вы можете создать ваш собственный тип проектов, используя наследование от системного класса *ProjectNode*. Иконка в списке проекта показывает тип проекта. Дополнительные опции могут быть добавлены к типу проекта, просто добавив пункт меню к меню Context.

## 1.3 Резюме

Теперь вы должны знать, как получить доступ к инструментам разработчика и как осуществлять навигацию в АОТ. У вас есть базовые знания о структуре приложения Ахарт с использованием технологии слоев и как эти слои используются в АОТ.

Следующая глава поможет вам сделать шаг дальше, и расскажет о конструкции языка в MorphX, называемого X++.

## 2 Введение в X++

Язык программирования в среде разработки MorphX называется X++. Это – объектно-ориентированный язык созданный для написания логики программных процессов в Ахарт.

Может возникнуть вопрос, почему был создан еще один язык программирования специально для Ахарт. Это, на самом деле, ключ к гибкости Ахарт, так как X++ оптимизирован для создания и изменения бизнес объектов. Структура языка простая, со встроенным синтаксисом SQL, поэтому нет нужды настраивать и управлять соединениями с источниками данных. Можно просто создавать свои SQL запросы точно так же, как Вы бы это делали, используя стандартный SQL. X++ также тесно интегрирован с набором инструментов MorphX, такими, как генератор форм и генератор отчетов. При добавлении какой-либо логики в формы или отчеты существует несколько стандартных методов, которые можно использовать для включения своих изменений.

Язык X++ был создан с помощью C++. Исходный код ядра на C++ не доступен разработчикам. В Вашем распоряжении имеется только набор системных объектов с определенным известным профилем параметров. Весь исходный код X++ открыт. Вы не можете спрятать написанный Вами на X++ код, и код на X++, который используется в стандартном приложении, также доступен. Это является большим преимуществом, так как Вы быстро научитесь проверять наличие похожего функционала в стандартном приложении перед тем, как создавать собственные модификации «с нуля». Часто Вы будете находить код, который упростит Вашу работу при написании собственных модификаций. Язык X++ имеет синтаксис схожий с языком Java, к которому добавлена возможность написания запросов обработки данных с использованием команд SQL. Хотя X++ и является объектно-ориентированным языком, у разработчика нет возможности наследовать все типы объектов как, к примеру, в C#. Классы в X++ могут быть наследованы, как в любом другом объектно-ориентированном языке. Помимо этого, Вы можете наследовать Расширенные Типы Данных, а также набор базовых классов, которые дают возможность делать общие изменения пользовательского интерфейса, такие, как изменение поведения всех форм. Описание базовых классов будет приведено ниже, в разделе **Классы [Classes]**.

Для того чтобы выполнить скрипт, написанный на X++, Вы можете воспользоваться узлом *Jobs* Репозитория Прикладных Объектов [AOT]. Для получения более детальной информации об узле *Jobs*, обратитесь к разделу **Задания [Jobs]**.

## 2.1 Переменные

В X++, переменные объявляются в верхней части редактора перед строками кода. Объявления переменных часто отделяются от основного кода строкой, содержащей только точку с запятой «;». Это связано со спецификой разбора кода компилятором. Если не добавить «точку с запятой» перед кодом, Вы можете получить сообщение об ошибке. «Точка с запятой» не обязательна во всех случаях, но на сегодняшний день использование «точки с запятой» в коде метода сразу после объявления переменных является стандартом Ахapta.

Считается правилом хорошего тона [best practice] использовать при объявлении переменных в X++ расширенные типы данных [EDT] вместо использования базовых типов языка. Причиной такого поведения является тот факт, что расширенные типы данных содержат информацию такую как длина строки, её выравнивание, др. Так как переменные могут быть использованы во многих местах в системе, изменение расширенного типа данных, а не типов переменных при объявлении, ускоряет разработку. Более детально расширенные типы данных рассмотрены в разделе **Словарь Данных [Data Dictionary]**.

Переменные рекомендуется называть с учетом контекста их использования, присваивая им мнемонические названия. Например, переменной для подсчета клиентов лучше присвоить имя noOfCustomers, а не состоящее из одной буквы i. При использовании зарезервированных слов в качестве названия переменной компиляция завершится с ошибками. Зарезервированные слова окрашены в редакторе X++ в синий цвет. Полный перечень зарезервированных слов можно найти в репозитории прикладных объектов в узле System Documentation.

Синтаксис объявления переменной приведен ниже:

```
CustAccount MyCustAccount;
```

Здесь объявляется переменная на основании расширенного типа данных CustAccount. Расширенный тип данных CustAccount имеет базовый тип string. Сначала вводится название расширенного типа, а затем имя переменной. Заметьте также, что объявление завершается «точкой с запятой». «Точка с запятой» должна завершать любое выражение в X++. Вы можете объявить больше одной переменной за раз в той же строке, разделяя переменные запятыми. Все базовые типы описаны в **Таблица 1: Базовые типы X++**.

Синтаксис объявления более одной переменной за раз:

```
CustAccount MyCustAccount1, MyCustAccount2, MyCustAccount3;
```

Этот код полностью работоспособен, но его нужно использовать с осторожностью. Компилятор определит тип первой переменной без проблем. Но если одна из последующих переменных, MyCustAccount2 или MyCustAccount3, будет иметь имя, совпадающее с именем какого-то класса или таблицы, то такой код будет откомпилирован с ошибками.

**Примечание:** Язык X++ не обладает возможностью объявления констант. При необходимости создания константы используются Макросы. Раздел **Макросы [Macros]** содержит детальное их описание.

Базовый Тип	Описание
Str	Содержит цифробуквенные символы. Для работы со строками используются функции, которые начинаются на str.
Int	Целочисленные значения. Функции minInt() и maxInt() могут быть использованы для получения минимального и максимального значения целого числа.
Real	Числа с плавающей запятой. Моделируют действительные числа.
Date	Дата рассчитывается, начиная с года 1901. В численном представлении, дата является количеством дней с начала 1901 года. Максимальное значение даты – конец 2155 года. Класс Global содержит несколько методов с префиксом date, которые используются для операций с датами, к примеру, определение даты прошлой или следующей недели, месяца, года.
Timeofday	Тип для определения времени, рассчитывается как количество секунд, начиная с полночи. На самом деле, этот тип является больше системным, чем базовым, но тип timeofday может быть выбран как один из базовых типов при создании расширенного типа данных в репозитории прикладных объектов.
Enum	Тип Перечисления используется для представления фиксированного набора значений. Для объявления переменных типа enum, как имя типа необходимо указывать один из созданных в АОТ перечислимый тип. Один из наиболее часто используемых перечислимых типов – это тип Boolean, который принимает значения истина [true] и ложь [false]. Перечисления могут содержать максимум 255 значений.
Container	Контейнеры могут быть объявлены с помощью зарезервированного слова container и следующим за ним идентификатором переменной. Container очень похож на массив, но может содержать данные различных типов. Контейнеры используются, к примеру, при работе с запросами. X++ содержит функции для работы с контейнерами, которые можно найти по префиксу con.
Anytype	Тип Anytype часто используется при передаче параметров, так как anytype может содержать данные любого базового типа. Anytype при инициализации принимает значение одного из базовых типов. Примером использования типа anytype служит статический метод <i>Global::queryValue()</i> .

**Таблица 10: Базовые типы in X++**

При объявлении переменных можно одновременно, в той же строке, инициализировать их начальными значениями. Это делается конструкцией вида = <Начальное значение>, которая дописывается после имени переменной.

Некоторые базовые типы допускают автоматическое или неявное преобразование типов, к примеру, запись целого числа в переменную типа `real`. Компилятор отслеживает присвоения с неявным преобразованием, и сообщит, если при преобразовании будут утеряны какие-либо данные. Компилятор также определит попытку неверной инициализации переменной, например, присвоение целочисленного значения строковой переменной.

```
static void Intro_BaseTypes(Args _args)
{
    Description      myString      = "A X++ string";
    Counter          myInteger     = 100;
    Qty              myReal        = 12.25;
    TransDate        myDate        = str2Date('12-31-2005', 213);
    TimeHour24       myTime        = str2Time('14:05');
    NoYesId          myEnum        = NoYes::Yes;
    PackedQueryRun   myContainer   = ['12', 'test', 'tada'];
    AnyType          myAnyType     = systemdateget();
;

    print myString;
    print myInteger;
    print myReal;
    print myDate;
    print myTime;
    print myEnum;
    print conPeek(myContainer, 1);
    print myAnyType;

    pause;
}
```

В приведенном выше примере объявляются переменные различных типов. При этом каждая переменная инициализируется при объявлении, после чего значения переменных выводятся на экран. Определить тип объявленной в коде переменной можно нажатием `ctrl+пробел` на имени переменной. Базовый тип переменной при этом будет показан как всплывающая подсказка.

При инициализации переменных с типом «Дата» и «Строка», было использовано две функции работы со строками. Использование функций в X++ описано далее в этой главе. Обратите также внимание на инициализацию переменной перечислимого типа (`enum`). Сначала указывается название перечислимого типа, за которым следует двойное двоеточие и конкретное значение перечислимого типа.

Переменная типа «контейнер» инициализируется значением длиной 3. Контейнер может содержать данные различных базовых типов, даже еще один контейнер. Доступ к содержимому контейнера организован с использованием специальных функций. В вышеуказанном примере на экран выводится значение, которое хранится в контейнере первым. Переменная типа `anytype` в нашем примере будет

иметь тип «Дата», так как при инициализации ей присваивается значение текущей даты.

Обратите внимание, что текст в коде берется в кавычки, причем при указании строкового значения в X++ можно использовать как двойные, так и одинарные кавычки. Компилятору безразлично, какая именно нотация используется. Правила хорошего тона разработки [Best practice] уточняют, что двойные кавычки следует использовать при написании текста, который будет отображаться на экране пользователя, а одинарные кавычки в том случае, когда строковое значение используется только в коде.

В X++ также можно объявлять массивы базовых типов. Для того, чтобы объявить переменную базового типа как динамический массив, после имени переменной необходимо вставить []. В квадратных скобках при объявлении массива можно указывать до двух параметров. Первый используется для того, чтобы установить массиву фиксированный размер, а второй - при использовании больших массивов, для указания количества значений, которые будут храниться в оперативной памяти (остальные будут записываться на жесткий диск). Необходимо заметить, что X++ умеет работать только с одномерными массивами.

```
static void Intro_Array(Args _args)
{
    CustAccount    myCustAccount[];
    ;

    MyCustAccount[1] = "4000";
    MyCustAccount[3] = "4001";
}
```

В примере создается массив на основании расширенного типа данных CustAccount. Первый и третий элементы массива инициализируются значениями. При использовании массивов инициализировать их конкретными значениями не обязательно.

Вы, должно быть, заметили также системный класс Array. Если вам нужен массив сложных типов данных, как, например, массив объектов, вам нужно будет использовать системный класс Array. Класс Array – один из фундаментальных классов и описан в главе **Классы**.

Переменная типа «контейнер» и переменные, объявленные как массивы – примеры сложных типов данных в X++, что иногда может запутать, так как тип контейнер присутствует в списке базовых типов данных, и контейнер в MorphX рассматривается как базовый тип. Таблицы и классы – это два других примера сложных типов данных X++.

---

**Примечание:** В X++ нет необходимости волноваться о выделении памяти. Когда объект уже больше не используется, сборщик мусора автоматически удалит объект и освободит память.

Сборщик мусора может быть настроен для специфических нужд для пользователя в меню **Сервис | Параметры**.

## 2.2 Операторы

Язык X++ поддерживает как унарные (одноместные), так и бинарные арифметические операторы. Есть единственный тернарный оператор в X++, который представляет собой короткую форму записи условия выбора if-else. Тернарный оператор, то есть оператор с тремя операндами, описывается в разделе «Условные операторы и операторы цикла».

Последовательность выполнения операторов зависит от приоритета каждого из них. Если в выражении участвует более одного оператора, их приоритеты используются для определения порядка выполнения операций выражения.

```
x + y * z
```

Операция умножения имеет приоритет больший оператора сложения, поэтому  $y * z$  будет выполнено до  $x + y$ . Для явного указания компилятору последовательности выполнения операций в выражении используются скобки.

```
(x + y) * z
```

Так как выражение  $x + y$  взято в скобки, то оно будет обработано первым. Несмотря на знание порядка выполнения операндов, рекомендуется использовать скобки для явного указания последовательности выполнения, так как это делает код намного более читабельным.

### Операторы присваивания

Операторы присваивания нужны для изменения значений переменных. Это может быть как простая инициализация переменной, так и возвращение вычисленного значения из метода.

Оператор	Выражение	Описание
=	$x = y$	Устанавливает значение $x$ равным значению $y$ .
+	$x + y$	Возвращает сумму значений $x$ и $y$ .
-	$x - y$	Возвращает разницу $x$ и $y$ . Этот оператор также может использоваться как унарный в качестве префикса целого или действительного числа.
*	$x * y$	Возвращает произведение $x$ и $y$ .
/	$x / y$	Возвращает частное $x$ и $y$ . Если $y$ имеет значение 0, то компилятор сгенерирует сообщение об ошибке «Деление на ноль».



DIV	x DIV y	Возвращает целочисленное частное x и y. Результат будет округлен в меньшую сторону.
MOD	x MOD y	Возвращает остаток от деления x на y, как целое число.
++	++x	Увеличивает значение x на 1. Это короткая форма записи выражения $x = x + 1$ . Это пример унарного оператора. Оператор может использоваться как префиксный, так и как постфиксный. Результат выполнения операции от размещения оператора не изменится.
--	--y;	Уменьшает значение x на 1. Это короткая форма записи выражения $x = x - 1$ . Это пример унарного оператора. Оператор может использоваться как префиксный, так и как постфиксный. Результат выполнения операции от размещения оператора не изменится.
+=	x += y	Увеличивает значение x на значение y. Это короткая форма записи выражения $x = x + y$ .
-=	x -= y	Уменьшает значение x на значение y. Это короткая форма записи выражения $x = x - y$ .

Таблица 11: Операторы присваивания

## Операторы отношения

Оператор отношения связывает два выражения и возвращает значение true или false. Операторы отношения используются в условных операторах, таких как оператор if-else для определения хода выполнения программы. При выборке данных из таблицы, операторы отношения используются для ограничения количества строк выборки – фильтрации записей.

Оператор	Выражение	Описание
>	x > y	Больше, чем: Возвращает True, если значение x превышает значение y.
>=	x >= y	Больше или равно: Возвращает True, если значение x больше или равно значению y.
<	x < y	Меньше, чем: Возвращает True, если значение x меньше значения y.
<=	x <= y	Меньше или равно: Возвращает True, если значение x меньше или равно значению y.
==	x == y	Равно: Возвращает True, если значение x равно значению y.
!=	x != y	Не равно: Возвращает True, если значение x отлично от значения y.
like	x like y	Подобный: Возвращает True, если значение x равно значению y. Оператор <i>like</i> использует символы обобщения * и ? для вычисления значения выражения. Часто, оператор <i>like</i> используется в запросах для выбора записей по значению, только частично совпадающему с

		искомым.
&&	x && y	Логическое И: Это один из логических операторов. Принимает значение true, если выражение x и y оба имеют значение True. Часто этот оператор используется совместно с другими операторами отношения, где && используется для проверки двух условий.
	x    y	Логическое ИЛИ: Принимает значение True, если выражение x или y, или оба выражения имеют значение True. Это также логический оператор, используемый для проверки условий.
!	!x	Логическое отрицание: Принимает значение True, если выражение x имеет значение False. Это единственный логический унарный оператор

Таблица 12: Операторы отношений

**Примечание:** При использовании выражения вида x && y в случае, если выражение x имеет значение false, то выражение y не будет рассчитываться, так как && возвращает true только в случае, если оба операнда принимают значение true. Эта информация может пригодиться, так как с помощью этого можно оптимизировать код, располагая сложные длительные вычисления в конце условного оператора, к примеру, выражения доступа к базе данных.

## Бинарные операторы

Бинарные операторы, как видно из названия, используются для вычисления выражений с использованием арифметических и условных операторов на уровне битов. Эти бинарные операторы могут быть использованы только с целочисленными значениями.

Бинарные операторы наиболее часто используются как альтернатива набору переменных, которые содержат значения true либо false, в случае, если нужно задавать доступ к элементам формы в несколько уровней. Но нужно учесть, что код с использованием бинарных операторов намного сложнее читать. Возможно это основная причина малой распространенности бинарных операторов в базовом коде приложения. Скорее всего, единственным местом, в котором у Вас будут использоваться бинарные операторы, будет код взаимодействия с внешними приложениями, такими как прямое обращение к функциям Windows API или необходимость низкоуровневого взаимодействия с базой данных.

Для понимания результата при использовании бинарных операторов необходимо перевести десятичные числа в двоичный код.

13 & 10	// 1101 бинарное И 1010
---------	-------------------------

Выражение `13 & 10` вернет результат 8. Это объясняется тем, что числа 13 и 10 сравниваются бит за битом и считаются только те биты, которые установлены для обоих чисел. В коде комментария приведено двоичное представление обоих чисел. Только в четвертом бите единичные значения совпадают – поэтому результатом бинарного оператора И будет число 8 (1000).

Оператор	Выражение	Описание
<code>&lt;&lt;</code>	<code>x &lt;&lt; y</code>	Сдвиг влево: Двоичное значение <code>x</code> будет сдвинуто на <code>y</code> позиций влево. Это увеличит значение <code>x</code> в $2^y$ раз.
<code>&gt;&gt;</code>	<code>x &gt;&gt; y</code>	Сдвиг вправо: Двоичное значение <code>x</code> будет сдвинуто на <code>y</code> позиций вправо. Это уменьшит значение <code>x</code> в $2^y$ раз.
<code>&amp;</code>	<code>x &amp; y</code>	Бинарное И операндов <code>x</code> и <code>y</code> : Результат будет состоять из тех битов, которые установлены как для <code>x</code> , так и для <code>y</code> .
<code> </code>	<code>x   y</code>	Бинарное ИЛИ операндов <code>x</code> и <code>y</code> : Результат будет состоять из битов, установленных для <code>x</code> или <code>y</code> .
<code>^</code>	<code>x ^ y</code>	Бинарное исключающее ИЛИ операндов <code>x</code> и <code>y</code> : Результат будет состоять из тех битов, которые не установлены как для <code>x</code> , так и для <code>y</code> .
<code>~</code>	<code>~x</code>	Бинарное отрицание <code>x</code> : Все биты будут инвертированы.

Таблица 13: Бинарные операторы

## 2.3 Условные операторы и операторы цикла

Код на C++ всегда выполняется последовательно. Часто появляется необходимость выполнить какую-то часть кода несколько раз, или же выполнить только часть кода. Условные операторы и операторы цикла используются совместно с операторами отношения для управления ходом выполнения программы.

### Циклы

Циклы используются для многократного повторения части кода. Для определения количества раз выполнения в циклах используются операторы отношения.

```
static void Intro_While(Args _args)
{
    Counter    counter = 1;
    ;
}
```

```
while (counter <= 10)
{
    info(strfmt("while loop %1", counter));
    counter++;
}
```

Это пример цикла с использованием оператора *while*. Целочисленная переменная используется для контроля количества итераций. Изначально ее значение устанавливается в 1. Тело цикла будет выполняться до тех пор, пока значение этой переменной не превысит 10. На каждой итерации в Infolog выводиться строка, и значение счетчика увеличивается на 1.

Функция `strfmt()` используется для форматированного вывода и преобразования любого из базовых типов в строку. Так как метод `info()` принимает в качестве первого аргумента только переменные типа `string` (строка), значение счетчика предварительно преобразуется в строку. Функция `Strfmt()` использует символ `%<номер аргумента>` для вставки преобразованных в строку аргументов.

Также стоит обратить внимание на выделение начала и конца тела цикла *while* фигурными скобками `{ }`. Они используются для выделения начала и конца части кода. В нашем примере итерационно будет выполняться только та часть кода, которая заключена в фигурные скобки. Использование фигурных скобок не является обязательным, но в противном случае выполнялась бы только первая строка после оператора *while*. Если бы мы забыли использовать фигурные скобки, то в нашем случае цикл бы никогда не прервался. Считается правилом хорошего тона использовать выделение кода фигурными скобками, даже если будет выделяться всего лишь одна строка, так как благодаря этому повышается читабельность кода. Также, если бы понадобилось добавить большее количество строк в тело цикла, то операторы, его ограничивающие, уже были бы добавлены.

В X++ циклы с оператором *while* часто используются для выборки данных из базы данных. Примеры такого использования циклов *while* приведены в разделе **Запросы к базе данных**.

```
static void Intro_DoWhile(Args _args)
{
    Counter    counter = 1;
;

    do
    {
        info(strFmt("do-while loop %1", counter));
        counter++;
    }
    while (counter <= 10);
}
```

Оператор цикла *do-while* похож на оператор *while*. Основным отличием является то, что цикл *do-while* выполнится по крайней мере один раз, так как условие проверки выхода из цикла описано после тела цикла. Цикл с использованием оператора *while*, где условие выхода анализируется в начале итерации, не выполнится ни разу, если условия изначально не выполняются. В качестве примера приведен точно тот же код, что и в предыдущем примере. Обратите внимание, что условие выхода из цикла должно завершаться точкой с запятой.

```
static void Intro_For(Args _args)
{
    Container names = ["Lay", "Kai", "Zbigniew", "Rolf", "Memed"];
    Counter counter;
;

    for (counter=1; counter <= conlen(names); counter++)
    {
        info(strFmt("%1: Name: %2", counter, conpeek(names, counter)));
    }
}
```

Оператор *for* работает по тому же принципу, что и оператор *while*, но имеет более лаконичную запись. Оператор *for* состоит из трех компонент заключенных в скобки: *инициализация счетчика*; *выражение условия выполнения цикла*; *обновление или инкремент*. После каждой итерации цикла, переменная счетчика увеличивается на значение, описанное в блоке обновления. Фигурные скобки используются для определения тела цикла. Выражение условия цикла анализируется перед началом каждой итерации и как только оно вернет значение *false*, выполнение тела цикла завершится и управление перейдет к оператору непосредственно за закрывающей фигурной скобкой. Цикл с оператором *for*, использованный в примере, проходит по элементам контейнера *names*. Значение счетчика *counter* устанавливается в 1 и используется для получения значения текущего элемента контейнера. Этот блок повторяется, пока значение счетчика не превысит количество элементов контейнера.

## Условные операторы

Причиной использования условных операторов является частая необходимость выполнения одной или более команд в зависимости от истинности или ложности какого-то условия.

```
static void Intro_IfElse(Args _args)
{
    NoYesId printToInfolog = true;
;
    if (printToInfolog)
    {
        info("Print to Infolog");
    }
    else
```

```
{
    print "Print to window";
    pause;
}
```

Оператор *If* – наиболее распространенный условный оператор. В наиболее простой форме, он состоит из одного оператора *if* и одного логического условия, заключенного в круглые скобки. В случае, если условие в скобках принимает значение *true*, код после оператора *if* выполняется, будь то одна строка кода или целый набор строк, заключенных в фигурные скобки *{ }*. Расширением оператора *if* является добавление после него оператора *else*. В этом случае, код, размещенный за оператором *else*, будет выполняться в том случае, если условие при *if* принимает значение *false*. Пример, приведенный выше, иллюстрирует написание условного оператора *if-else*. Более сложная логика может быть получена вложением операторов *if* один в другой. В приведенном ниже примере, на блок кода, выполняющийся по *else*, наложено дополнительное условие с использованием оператора *if*.

```
static void Intro_IfElse(Args _args)
{
    NoYesId    printToInfolog = true;
    NoYesId    printToWindow = true;
;
    if (printToInfolog)
    {
        info("Print to Infolog");
    }
    else if (printToWindow)
    {
        print "Print to window";
        pause;
    }
}
```

В примере тот же результат можно было получить простым использованием двух отдельных операторов *if*. Размещение второго оператора *if* после оператора *else* позволяет не анализировать второе условие, если истинно условие первого оператора *if*, что дает незначительный выигрыш в производительности кода.

Вложенность операторов *if* неограниченна, но следует учитывать, что многоуровневые условные операторы затрудняют чтение и отладку кода. Очень часто, необходимость условий с большим количеством операторов *if* возникает в программе при проверке значения всего одной переменной, от конкретного значения которой и зависит ход выполнения. Если количество допустимых значений этой переменной мало, то и использование оператора *if* является целесообразным. Но когда переменная может принимать 3 и более значений, перечисление всех их с использованием оператора *if* уже не является разумным

решением. К счастью, X++ поддерживает оператор, специально разработанный для таких нужд, оператор *switch*:

```
static void Intro_Switch(Args _args)
{
    SalesStatus    salesStatus = SalesStatus::Invoiced;
    ;

    switch (salesStatus)
    {
        case SalesStatus::Delivered:
            info("Salesorder is delivered");
            break;
        case SalesStatus::Invoiced:
            info("Salesorder is invoiced");
            break;
        case SalesStatus::Canceled:
            info("Salesorder is cancelled");
            break;
        default:
            info("Do nothing");
    }
}
```

Если бы вместо оператора *switch* в вышеприведенном примере использовались операторы *if*, то это потребовало бы нескольких уровней вложенности и повлекло за собой сложность при чтении написанного кода. При использовании оператора *switch* все значения перечисляются на одном уровне, что облегчает прочтение кода, а также, в перспективе, позволяет без особых трудностей добавить обработку дополнительных значений проверяемого условия.

Справа от оператора *switch* в круглых скобках указывается анализируемое выражение. Следом за ним указывается один или более оператор *case*. Каждый оператор *case* указывает на одно или более значений, разделенных запятыми, после которых ставится двоеточие. X++ вычисляет значение анализируемого выражения и последовательно, сверху вниз, сравнивает его с каждым из значений, приведенных справа от операторов *case*. В случае совпадения вычисленного значения с тем, которое указано на одном из операторов *case*, выполняется код, следующий за этим оператором. Если же значение не совпало ни с одним из перечисленных значений, то выполняется код после оператора *default*. Оператор *default* не является обязательным при использовании оператора *switch*. Его следует добавлять только в случае необходимости, к примеру, чтобы сообщить пользователю о том, что ни одно из приведенных значений не удовлетворяет приведенному условию. Оператор *break* добавлен в конце каждого блока кода *case*. При переходе на строку с оператором *break*, управление передается оператору, который следует за закрывающей фигурной скобкой оператора *switch*. Если явно не указать оператор *break* после окончания кода в блоке *case*, после окончания выполнения этого кода будет также выполнен весь код, следующий за тем блоком *case*, значение которого совпадает с условием. Это, к слову, одна из

наиболее распространенных ошибок, связанных с использованием оператора *switch*. Запуск проверки на Правила Хорошего Тона (Best Practice) выведет операторы *case*, которые не завершаются оператором *break*.

```
static void Intro_TernaryOperator(Args _args)
{
    Boolean    printCustomerName = false;
    ;

    print printCustomerName ? "Customer name" : "Customer account";
    pause;
}
```

Если нужно написать условный оператор с простым видом условия, можно использовать более короткую альтернативу блока *if-else*, тернарный оператор, то есть оператор трех операндов. Первый операнд представляет собой логическое условие, такое, как и в блоке *if*, за которым следует вопросительный знак. Вторым и третьим операндами – это выражения, одно из которых будет выполнено в зависимости от того, пример первый операнд значение *true* или *false*. В случае, если принятое значение *true*, выполняется второй операнд, иначе управление передается третьему операнду. Эти операнды указываются после вопросительного знака и разделяются двоеточием. Следует заметить, что значения, получаемые в результате исполнения кода второго и третьего операндов должны иметь один и тот же тип данных.

Так как тернарный оператор позволяет записать проверку условия в одну строку, то он в значительной мере повышает легкость прочтения кода программы. Наиболее удобно использовать тернарные операторы с операторами присвоения в случаях, когда присваиваемой переменной может быть назначено только одно из двух значений. К примеру, они часто используются при определении доступности и видимости полей формы в зависимости от определенных условий. Тернарные операторы также могут иметь вложенные уровни.

## Исключительные ситуации

Циклы и условные операторы разработаны для управления ходом выполнения программы. Непредвиденные условия, такие как предупреждения и ошибки необходимо показывать пользователю по мере их возникновения. В случае возникновения исключительной ситуации необходимо прервать выполнение кода, выполняемого по умолчанию, и передать управление тому блоку кода, который отвечает за выполнение действий на основании информации о той исключительной ситуации, которая привела к ошибке выполнения обычного потока программы.



```
static void Intro_TryCatch(Args _args)
{
    Counter counter;

    try
    {
        while (counter < 10)
        {
            counter++;

            if (counter MOD 7 == 0)
                throw error("Counter MOD 7 is zero");

            if (counter MOD 3 == 0)
                throw warning("Counter MOD 3 is zero");
        }
    }
    catch (Exception::Error)
    {
        print ( strfmt("An error appeared at loop %1", counter));
    }
    catch (Exception::Warning)
    {
        print ( strfmt("A warning appeared at loop %1", counter));
        retry;
    }

    pause;
}
```

Оператор *Try-catch* используется в X++ для управления исключительными ситуациями. Оператор *try-catch* состоит из двух или более отдельных блоков кода: блок *try*, в котором производится попытка выполнения какого-то действия, и один или более блоков *catch*, в котором перехватываются исключительные ситуации, которые возникли в блоке *try*. Исключительные ситуации могут возникнуть в программе по вызову ядра или же специального оператора *throw*. Блок оператора *catch* отвечает за обработку возникшей исключительной ситуации. В коде может быть несколько блоков *catch*, следующих один за другим и перехватывающих различные типы исключительных ситуаций. Какие-либо действия будут производиться только для тех видов исключительных ситуаций, которые явно прописаны в коде. В примере выше использование оператора *try-catch* будет производить только обработку исключительных ситуаций *error* и *warning*. Также можно добавить блок с оператором *catch* без указания какого-либо типа исключительной ситуации, что повлечет перехват и обработку всех видов исключений, кроме тех, которые явно описаны в других блоках оператора *catch*.

Несмотря на то, что всегда можно написать код, аналогичный обработке исключений, используя обычные условные операторы, можно заметить повсеместное использование блоков *try-catch*. Наибольшим преимуществом

использования обработчиков исключительных ситуаций является то, что исключения можно перехватить до того, как они будут помещены в Infolog, и предпринять соответствующее действие. Особенную ценность они представляют при операциях с базой данных, таких, как обновление записей. При обновлении записей могут происходить исключительные ситуации, например, блокировка таблицы. Обычно это временное явление, и если попробовать обновить запись заново, обновление может пройти успешно. Если бы не было обработчика исключительных ситуаций, неудачная попытка обновления передалась бы сразу системному обработчику ошибок, который отобразил бы сообщение об ошибке в infolog и прекратил бы выполнение программы, что было бы в данной ситуации излишним. При возникновении исключения оператор *retry* возвращает исполнение кода в начало блока *try*, и пользователь даже не заметит, что возникла исключительная ситуация. При создании кода обработки для блока *catch*, необходимо включать в него логику, которая будет определять условия возникновения исключения, и проверять, закончились ли они, иначе программа может войти в бесконечный цикл, так как будет беспрерывно повторять тот код, который вызвал исключительную ситуацию.

## Вспомогательные операторы

Язык X++ содержит команды, которые могут быть использованы для изменения порядка выполнения кода. Часто такие команды используются совместно с условными операторами, если продолжать выполнение кода не нужно.

```
static void Intro_Break(Args _args)
{
    Counter    counter;
;

    for (counter=1; counter <= 10; counter++)
    {
        print counter;

        if (counter == 5)
        {
            break;
        }
    }

    pause;
}
```

В предыдущих примерах уже использовался оператор *break* для завершения каждого из блоков *case* в операторе *switch*. Оператор *break* можно использовать и в любом другом участке кода. При использовании в теле цикла условие выхода из цикла будет игнорировано и будет произведен выход из цикла. В примере, производится выход из цикла после пяти прогонов. При выполнении какого-то тяжелого куска кода, как например поиск в базе данных, можно ускорить свой код

добавлением оператора *break* после получения желаемого результата вместо того, чтобы ожидать окончания всех операций.

Использование оператора *return* в приведенном цикле даст точно тот же результат, что и оператор *break*, но оператор *break* является более предпочтительным. Оператор *return* используется для возвращения значения из функции. Использование методов описано в разделе **Классы [Classes]**.

```
static void Intro_Continue(Args _args)
{
    Counter    counter;
;

    for (counter=1; counter <= 10; counter++)
    {
        if (counter MOD 3 == 0)
        {
            continue;
        }

        print counter;
    }

    pause;
}
```

Отличие между оператором *break* и оператором *continue* состоит в том, что оператор *break* производит выход из тела цикла, а оператор *continue* просто перейдет к следующей итерации, пропустив оставшийся код тела цикла. Тот же результат можно получить обычным использованием оператора *if*. Использовать оператор *continue* может быть удобно во избежание добавления условного оператора *if* и отступа для всего тела оператора.

## 2.4 Запросы к базе данных

Для выборки данных из базы данных в языке X++ используется оператор *select*. Запросы к базе данных в X++ могут быть написаны в любом месте кода, как и любые другие операторы. Для того чтобы сделать выборку из определенной таблицы, сначала необходимо объявить переменные этой таблицы. Особая форма оператора *while*, оператор *while select*, используется для выбора набора записей базы данных, удовлетворяющих определенным условиям. Условия выборки определяются с помощью операторов и переменных. Базовый тип строка – *str* – не может быть использован в условиях выборки, если не указана его фиксированная длина. Это можно считать еще одной причиной для использования расширенных типов данных.

```
static void Intro_Select(Args _args)
{
```

```

CustTable custTable;
CustTrans custTrans;
TransDate fromStartYear = systemdateget();
;

while select custTable
    join custTrans
        where custTrans.accountNum == custTable.accountNum
            && custTrans.transDate >= fromStartYear
    {
        info(strfmt("%1, %2, %3", custTable.accountNum, custTrans.transDate, custTrans.voucher));
    }
}

```

Выше приведен пример использования оператора *while select* в коде X++. Таблицы CustTable [Справочник клиентов] и CustTrans [Проводки клиента] соединяются по номеру счета клиента, и отсекаются записи CustTrans, у которых дата проводки меньше даты переменной fromStartYear. Для хранения значения даты создана переменная, в которую записывается результат функции systemdateget(), которая возвращает текущую системную дату. Хотя эту функцию и можно непосредственно использовать в условии выборки, разумнее действовать через переменную во избежание пересчета функции systemdateget() на каждой итерации цикла выборки.

Запросы к базе данных в X++ при использовании оператора *select* не полностью совпадают со стандартом SQL. Также, в X++ не поддерживаются все ключевые слова стандартного SQL. При наличии базовых знаний написания запросов на SQL, будет довольно легко привыкнуть к специфике написания запросов X++. Перечень ключевых слов, которые поддерживаются в языке X++, приведен (по возрастанию) в **таблице 5: Обзор ключевых слов Select**.

Ключевое слово	Описание
asc	<p>Устанавливает возрастающий порядок сортировки. Все поля сортировки в запросах к базе данных по умолчанию имеют сортировку по возрастанию. Используется совместно с <i>order by</i> и <i>group by</i>.</p> <p><u>Пример</u> select custTable order by accountNum asc;</p>
avg	<p>Агрегатное ключевое слово используется для возвращения среднего значения поля из выбранных записей. Запросы с использованием агрегатных функций возвращают результат по большому количеству строк, и возвращают одну строку. Примером может служить метод класса <i>KMKnowledgeCollectorStatisticsExecute.runQuery()</i>.</p> <p><u>Пример</u> select avg(amountMST) from custTrans;</p>
count	<p>Агрегатное ключевое слово используется для возвращения количества строк, которые вернула выборка.</p>

	<p>Примером может служить метод класса <i>KMKnowledgeCollectorStatisticsExecute.runQuery()</i>.</p> <p><u>Пример</u>  select count(recId) from custTrans;</p>
delete_from	<p>Удаляет набор строк за одно обращение к базе данных. Очень ускоряет процесс при удалении большого количества строк по сравнению с обычным <i>while select</i>, который при вызове метода <i>delete</i> генерирует запрос к базе данных для каждой удаляемой строки.</p> <p>Примером может служить метод класса <i>InventCostCleanUp.updateDelSettlement()</i>.</p> <p><u>Пример</u>  delete_from myTable  where myTable.amountMST &lt;='1000';</p>
desc	<p>Устанавливает ниспадающий порядок сортировки. Используется совместно с <i>order by</i> и <i>group by</i>. Запросы с сортировкой по убыванию могут приводить к значительному снижению производительности, так как индексы всегда сортируются в порядке возрастания.</p> <p>Примером может служить метод таблицы <i>CustTable.lastPayment()</i>.</p> <p><u>Пример</u>  select custTable order by name desc;</p>
exists join	<p>Ключевое слово <i>Exists join</i> используется для выборки строк таблицы, для которых есть хотя бы одна запись в связанной подчиненной таблице, удовлетворяющая условиям выборки. Записи из подчиненной таблицы при использовании <i>exists join</i> выбираться не будут.</p> <p>Примером может служить метод класса <i>InventAdj_Cancel.cancelInventSettlements()</i>.</p> <p><u>Пример</u>  while select custTable  exists join custTrans  where custTable.accountNum == custTrans.accountNum</p>
firstfast	<p>Инструкция, которая позволяет сократить время выборки первой строки выборки. Может быть использована в ситуациях, когда возвращается всего одна строка, например для отображения в диалоге.</p> <p>Примером может служить метод класса <i>ProjPeriodCreatePeriod.dialog()</i>.</p> <p><u>Пример</u>  select firstfast custTable order by accountNum;</p>
firstonly	<p>Инструкция указывает на то, что будет выбрана только первая строка, удовлетворяющая условиям выборки. <i>Firstonly</i> следует использовать во всех запросах, в которых не используется <i>while</i> для множественной выборки, даже если запрос может вернуть всего одну строку. Статические методы таблиц <i>find()</i> и <i>exists()</i> всегда используют инструкцию <i>firstonly</i>.</p>

forceliterals	<p>При использовании ключевого слова <i>forceliterals</i> ядро всегда будет выбирать строки без использования подготовленного запроса. Используется для того, чтобы убедиться, что будет использован оптимальный индекс для условий, указанных в запросе. Для оптимизации соединений [<i>join</i>] больших таблиц ядро по умолчанию использует инструкцию <i>forceliterals</i> при выборке. Для определения необходимости использования этой инструкции используется свойство таблицы <i>TableGroup</i>. Инструкция <i>forceliterals</i> используется в том случае, если в соединении участвуют как минимум две таблицы, в свойстве <i>TableGroup</i> которых установлено не <i>Parameter</i> и не <i>Group</i>.</p> <p><u>Пример</u>  <pre>select forceliterals custTrans order by voucher, transDate where custTrans.accountNum &gt;= "4000";</pre></p>
forcenestedloop	<p>Используется в соединениях (<i>join</i>) для явного указания ядру выбирать запись из главной таблицы до выборки строк из подчиненных таблиц. Часто используется в комбинации с инструкцией <i>forceselectorder</i>. Примером может служить метод класса <i>ReqCalc.actionCalcDimTrans()</i>.</p> <p><u>Пример</u>  <pre>select forcenestedloop forceselectorder custTable join custTrans where custTable.accountNum == custTrans.accountNum &amp;&amp; custTable.name &gt;= "4000";</pre></p>
forceplaceholders	<p>Дает ядру инструкцию выбирать записи с использованием подготовленного запроса. Используется для запрета использования инструкции <i>forceliterals</i>, которая по умолчанию используется при соединении больших таблиц. Ядро использует эту инструкцию при выполнении запросов, которые не работают с большими таблицами и их соединениями. Примером может служить метод класса <i>ReqCalc.actionCalcDimTrans()</i>.</p> <p><u>Пример</u>  <pre>select forceplaceholders custTable join custTrans where custTable.accountNum == custTrans.accountNum &amp;&amp; custTable.name &gt;= "4000";</pre></p>
forceselectorder	<p><i>Forceselectorder</i> явно указывает ядру обращаться к соединенным (<i>join</i>) таблицам в том порядке, в котором они были соединены в запросе. Часто используется в комбинации с инструкцией <i>forcenestedloop</i>.</p>
forupdate	<p>Если выбираемые запросом записи необходимо будет изменять, ключевое слово <i>forupdate</i> должно быть добавлено в запрос.</p> <p><u>Пример</u>  <pre>while select forupdate custTable</pre></p>
from	<p>По умолчанию запрос выбирает все поля таблицы при выборке. Ключевое слово <i>From</i> используется для указания конкретного набора полей выборки. Это оптимизирует запрос, особенно в случае таблиц с большим количеством полей. Использовать ключевое слово следует</p>

	<p>только для оптимизации, так как перечисление требуемых полей очень засоряет код.</p> <p><u>Пример</u>  select accountNum, name from custTable;</p>
group by	<p>Производит группировку по указанным в <i>group by</i> полям, одновременно сортируя в том же порядке. Если не используются агрегатные функции, будут выбраны только те поля таблицы, которые перечислены в <i>group by</i>.  Примером может служить метод класса <i>InventStatisticsUS.calcTotals()</i>.</p> <p><u>Пример</u>  while select custTable group by custGroup;</p>
index	<p>Указывает базе данных о необходимости использования указанного индекса. Устанавливает порядок сортировки выбираемых строк, используя поля из указанного индекса. Ключевое слово следует использовать только в случае необходимости определенной сортировки, так как база данных автоматически подбирает оптимальный для запроса индекс.</p> <p><u>Пример</u>  while select custTable index accountIdx</p>
index hint	<p>Предлагает базе данных использовать указанный индекс при обработке запроса. База данных может проигнорировать эту команду, если есть более подходящий с точки производительности индекс.</p> <p><u>Пример</u>  while select custTable index hint accountIdx</p>
insert_recordset	<p>Используется для вставки большого набора строк в таблицу. <i>Insert_recordset</i> полезен при копировании строк из одной таблицы в другую, так как он выполняет это за один запрос к базе данных. Количество полей в таблице, из которой производится копирование, должно совпадать с количеством полей той таблицы, в которую производится вставка строк, а также должны совпадать базовые типы этих полей.  Примером может служить метод класса <i>ReleaseUpdateDB_V25toV30.updateASPEmail()</i>.</p> <p><u>Пример</u>  insert_recordset myTable (myNum, mySum)  select myNum, sum(myValue) from anotherTable  group by myNum  where myNum &lt;= 100;</p>
join	<p>Соединяет таблицы, используя тип связи <i>inner join</i>. Выбираются записи обеих таблиц при совпадении значений в полях, по которым производилось соединение.  Примером является табличный метод <i>SalesTable.LastConfirm()</i>.</p> <p><u>Пример</u>  while select custTable</p>

	<pre>join custTrans   where custTable.accountNum == custTrans.accountNum</pre>
maxof	<p>Агрегатная функция, которая возвращает максимальное значение указанного поля из строк, удовлетворяющих условиям запроса. Примером может служить метод класса <i>KMKnowledgeCollectorStatisticsExecute.runQuery()</i>.</p> <p><u>Пример</u>  <pre>select maxOf(amountMST) from custTrans;</pre></p>
minof	<p>Агрегатная функция, которая возвращает минимальное значение указанного поля из строк, удовлетворяющих условиям запроса. Примером может служить метод класса <i>KMKnowledgeCollectorStatisticsExecute.runQuery()</i>.</p> <p><u>Пример</u>  <pre>select minOf(amountMST) from custTrans;</pre></p>
next	<p><i>Next</i> используется для перехода к следующей записи выборки. Предпочтительным является использование оператора <i>while select</i>.</p> <p><u>Пример</u>  <pre>select custTable;  next custTable;</pre></p>
nofetch	<p>Указывает, что нет записей, представленных в выборке. Обычно используется, когда результат выборки передается другому прикладному объекту, например запросу, который будет выполнять актуальную выборку. Примером может служить метод класса <i>PurchCalcTax_PurchOrder.initCursor()</i>.</p> <p><u>Пример</u>  <pre>select nofetch custTable;</pre></p>
notexists join	<p>Противоположность <i>exists join</i>. Будут выбраны записи из главной таблицы, для которых нет соответствия в подчиненной таблице с учетом условий выборки и соединения. Примером может служить метод класса <i>InventConsistencyCheck_Trans.run()</i>.</p> <p><u>Пример</u>  <pre>while select custTable   notexists join custTrans     where custTable.accountNum == custTrans.accountNum</pre></p>
order by	<p>Устанавливает порядок сортировки выбранных записей по полям списка. В случае отсутствия индекса по полям списка сортировки эта операция может выполняться продолжительное время. Примером может служить табличный метод <i>CustTrans.transactionDate()</i>.</p> <p><u>Пример</u></p>



	select custTrans order by accountNum, transdate desc
outer join	<p>При использовании <i>Outer join</i> будут выбраны записи из обеих таблиц вне зависимости от совпадений в полях связи. При этом из главной таблицы извлекаются все записи, а из подчиненной – только те записи, которые имеют связь в главной таблице. Примером может служить метод класса <i>SysHelpStatistics.doTeams()</i>.</p> <p><u>Пример</u> while select custTable outer join custTrans</p>
reverse	<p>Записи будут выбраны в обратном порядке. Ключевые слова <i>asc</i> и <i>desc</i> используются для инвертирования порядка сортировки по одному полю, тогда как <i>reverse</i> относится ко всей таблице полностью. Примером может служить метод класса <i>InventUpd_Reservation.updateReserveLess()</i>.</p> <p><u>Пример</u> while select reverse custTable</p>
setting	Это ключевое слово используется совместно с <i>update_recordset</i> .
sum	<p>Агрегатная функция, которая возвращает сумму по указанному полю из всех выбранных строк. Примером может служить метод класса <i>KMKnowledgeCollectorStatisticsExecute.runQuery()</i>.</p> <p><u>Пример</u> select sum(amountMST) from custTrans;</p>
update_recordset	<p>С помощью команды <i>update_recordset</i> можно за одно обращение к базе данных обновить несколько записей. Полезно для быстрой инициализации набора полей. Поля, которые будут обновляться, указываются после ключевого слова <i>setting</i>. Примером может служить метод класса <i>ProdUpdHistoricalCost.postScrap()</i>.</p> <p><u>Пример</u> update_recordset myTable setting field1 = myTable.field1 * 1.10;</p>

Таблица 14: Обзор ключевых слов запроса

В предыдущей версии Ахарты обычно использовалось явное указание индекса для использования в запросах. Для этого использовались ключевые слова *index* и *index hint*. Современные средства оптимизации базы данных очень хорошо справляются с задачей определения наилучшего индекса для запроса, поэтому нет смысла явно указывать специфический индекс в запросе. Более того, указание неправильного индекса может резко снизить скорость выполнения запроса. Использовать индексы следует только в том случае, если нужно указать определенный порядок сортировки полей выборки.

Ключевые слова *forceliterals*, *forcenestedloops*, *forceplaceholders*, *forceselectorder* используются для оптимизации времени исполнения запросов. Эти команды меняют способ выборки данных по умолчанию, называемый планом выполнения. При использовании этих команд следует соблюдать осторожность. Возможно, удастся получить лучшую производительность на тестовых данных, но на реальных данных производительность может оказаться намного хуже. Влияние этих команд на время исполнения зависит от структуры данных. Не следует использовать эти команды без полного понимания их работы.

При написании запросов на выборку данных для обновления или удаления необходимо использовать систему отслеживания целостности транзакций [*Transaction Tracking System*], которую обычно называют просто *TTS*. Система *TTS* позволяет указать логические границы транзакции. Логическая транзакция может содержать обновления нескольких таблиц. Обычно целостность базы данных сильно зависит от связей между обновляемыми записями. Система отслеживания транзакций *TTS* гарантирует, что при отказе хотя бы одного обновления в теле логической транзакции остальные обновления в этой транзакции будут отменены до состояния, которое было перед началом транзакции, для того, чтобы сохранить целостность базы данных. Система не позволит выбрать запись на обновление без использования *TTS*, сгенерировав соответствующее сообщение об ошибке. Более детальную информацию о транзакциях можно найти в главе **Словарь Данных** [*Data Dictionary*].

## 2.5 Функции

Язык X++ располагает набором системных функций доступных для выполнения различного рода задач. Эти функции написаны на C++ и являются частью ядра системы. Имеются функции для преобразования данных, например функция для преобразования целого числа в строку. Также есть набор полезных функций для работы с датами и строками. Обзор основных системных функций можно найти в узле *System Documentation* Репозитория прикладных объектов [АОТ].

Для того чтобы использовать системную функцию в коде X++, достаточно написать имя функции и передать параметры после открывающей круглой скобки. Опытные разработчики X++ возможно заметили, что доступ к системным функциям организован аналогично методам класса *Global*. С точки зрения X++, между ними нет разницы. Одним из способов определения происхождения функции является Просмотр определения функции или метода по правому клику мыши - нельзя просмотреть системную функцию.

## 2.6 Резюме

В этой главе был описан язык программирования X++, используемый в среде разработки MorphX. После прочтения этой главы Вы должны были получить базовые знания о синтаксисе X++. Прочитав эту главу, Вы должны были освоить способы определения переменных, использование условных операторов и формирования циклов, а также методы выборки данных из базы данных. Изучение языка X++ является первым шагом в создании собственных модификаций для Ахарта. Следующая глава расскажет о каждом из узлов Репозитория прикладных объектов [АОТ], а также о том, как добавлять дополнительную логику в работу хранимых в АОТ объектов, используя для этого X++.



## 3 Словарь данных [Data Dictionary]

Модель данных Ахарты (типы данных, таблицы) создаётся и настраивается в узле *Data Dictionary* репозитория прикладных объектов. Этот узел является отправной точкой при модификации модели данных стандартной инсталляции Ахарты. Данные Ахарты хранятся в реляционной базе данных. Вкратце, это значит, что данные хранятся в отдельных таблицах для предотвращения дублирования данных в разных таблицах, а между таблицами настроены связи для доступа к информации из связанных таблиц. Базовые знания реляционной модели баз данных и нормализации данных являются необходимостью при изменении модели данных Ахарты. Описание реляционных баз данных, принципов их построения и поддержки не рассматриваются в этой книге. На сегодняшний день имеется большое количество литературы об этом, а также огромное количество информации можно найти в сети Интернет.

Для хранения данных в Ахарты предусмотрена работа с СУБД Microsoft SQL Server или Oracle посредством настроенного на рабочей станции источника данных. Таблицы, Представления, Поля и Индексы синхронизируются с базой данных при их создании, изменении или удалении из Ахарты. Это описание данных – единственная информация о словаре данных, которая хранится в базе данных, сами данные физически хранятся в базе данных Microsoft SQL Server или Oracle. Информация касательно связей между таблицами и действий при удалении записей [deleteActions] может быть найдена в MorphX. Визуальный MorphXplorer может быть использован для построения диаграмм связей между объектами (ER-диаграмм).

### 3.1 Таблицы

Существует две основных категории таблиц:

- *Прикладные таблицы.* Эти таблицы используются для определения и построения прикладных модулей системы. При первоначальной установке Ахарты вы определяете те таблицы, которые будете использовать как прикладные. Это определяется активацией определенных конфигурационных ключей. Прикладные таблицы будут синхронизированы с базой данных при загрузке системы.
- *Системные таблицы.* Эти таблицы содержат информацию критичную для работоспособности инфраструктуры Ахарты и создаются ядром приложения. Системные таблицы автоматически синхронизируются с базой данных.

При проблемах с синхронизацией может оказаться полезным инструмент «Администрирование SQL», который находится в главном меню в **Администрирование | Периодические операции | SQL Администрирование.**

Запуск задания Проверки/Синхронизации решит большинство наиболее распространенных ошибок синхронизации.

## Компания

В Ахарты имеется возможность использования одного и того же приложения для работы с информацией из разных компаний. Хотя данные и хранятся в одной и той же базе данных на физическом уровне, Ахарты самостоятельно определит, какие данные необходимо использовать для каждой компании, и при этом приложение будет выглядеть абсолютно неизменным в интерфейсе и функциональности.

Это может оказаться полезным в нескольких случаях:

- У Вас могут быть настроены тестовые данные для обучения нового персонала работе с системой Ахарты. Для этого можно использовать компанию 'test', благодаря чему вы будете спокойны, что никакие из критичных «живых» данных не будут повреждены.
- У вас может присутствовать компания 'planning' (планирование) или 'budget' (бюджет), в которых вы сможете готовить модели на будущие периоды.
- У вас могут быть более одной реальной компании, каждая из которых будет работать в своей сфере бизнеса, но использовать один и тот же функционал системы. У вас может быть различная информация в каждой из компаний: Например, разные счета Главной Книги, разные клиенты и поставщики, правила управления складом и запасами в наличии и даже организационная структура компании. Вся эта информация будет храниться под отдельным кодом компании.

Ахарты управляет разграничением данных, благодаря использованию системного поля *dataAreaId*, которое определяет, какой компании принадлежат данные. При добавлении записи, *dataAreaId* автоматически инициализируется кодом текущей компании. При запросе на показ каких-либо данных пользователем, будь то отчет, форма или выборка данных с использованием оператора *select*, Ахарты всегда вернет данные именно из текущей компании, так как фильтр по полю *dataAreaId* будет автоматически добавлен ядром приложения.

```
static void DataDic_ChangeCompany(Args _args)
{
    DataArea  dataArea;
    CustTable custTable;
;

    while select dataArea
    {
        if (!dataArea.isVirtual)
        {
```

```
info(strfmt("Company: %1", dataArea.id));

changeCompany(dataArea.id)
{
    custTable = null;

    while select custTable
    {
        info(strfmt("%1, %2", custTable.dataAreaId, custTable.accountNum));
    }
}
}
```


Пример, приведенный выше, иллюстрирует, как следует поступить при необходимости выборки данных из более, чем одной компании. В примере происходит выборка и вывод всех клиентов, которые принадлежат компаниям, не являющимся виртуальными. `changeCompany()` используется для перехода к указанной компании. Компания изменяется только на время выполнения блока кода после `changeCompany()`, а после его завершения компания возвращается к начальному значению. Обратите внимание на присваивание табличной переменной `CustTable` значения `null`. Табличная переменная должна быть «сброшена», иначе данные будут выбраны из текущей компании. `ChangeCompany()` следует использовать с осторожностью, иначе может получиться так, что вы измените данные в другой компании. Виртуальные компании описаны далее в разделе **Табличные Коллекции**.

## Прикладные таблицы

При необходимости создания новой таблицы и формы в системе Ахapta, вводимые данные должны быть сохранены в базе данных в соответствующей таблице. Новые поля можно добавить как к существующим, так и к новым таблицам. При добавлении новых полей к уже существующей таблице, новые поля будут добавлены в текущем слое, несмотря на то, что таблица была создана на другом слое. Это правило касается всех объектов на таблице, кроме действий при удалении [delete actions]. При обновлении приложения новым релизом это является большим плюсом, так как вам нужно будет всего лишь вручную загрузить узлы АОТ, измененные на более чем одном слое. Узел Таблицы является под-узлом узла *Data Dictionary* в репозитории АОТ.

## Обозреватель таблицы

Обозреватель таблицы можно вызвать по правому щелчку мышкой на таблице, выбрав пункт «Обозреватель таблицы» из подменю Add-Ins. Обозреватель таблицы – это стандартная форма, которая называется `SysTableBrowser` в АОТ. Обозреватель можно вызвать для любой таблицы в АОТ, как прикладной, так и системной. Также обозреватель можно вызвать по правому клику на источнике

данных формы. По умолчанию, отображаются все поля, у которых свойство **Visible** установлено в **'true'**. Для того, чтобы просмотреть поля, которые отображаются в Авто-Отчете, необходимо переключить настройку «Отображать поля» в значение Авто-отчет. Можно наложить фильтр на строки, выводимые в обозревателе, изменив запрос в нижней части формы, но намного проще накладывать фильтры использованием значка  из верхнего меню.

Обозреватель таблицы является достаточно удобным инструментом, если нужно получить общую характеристику данных в таблице, так как для этого не нужно знать, где в меню находится соответствующая форма. Обозреватель может использоваться для создания или изменения данных, но только в режиме тестирования. И очень важным является использование этого способа только для тестирования. Данные, изменяемые в обозревателе таблиц, *будут* проверены бизнес-логикой, определенной на таблицах, но формы, созданные в Ахартa для отображения данных таблицы, могут содержать дополнительную логику, и поэтому для ввода реальных данных следует использовать именно формы. Используя обозреватель таблицы в реальном приложении, вы можете повредить данные, поэтому этот инструмент не является пользовательским.

### Создание таблиц

Очень важным при создании новых таблиц является общая цель, для которой создаются эти таблицы. После установки таблиц на рабочее приложение и ввода данных в эти таблицы будет намного сложнее изменить структуру ваших таблиц и индексы, так как при этом также придется конвертировать и уже существующие данные.

Также при создании следует учитывать производительность. При создании новой таблицы для операций, нужно убедиться, что создаваемый индекс для выборки данных хорошо продуман и подходит как для простых запросов, так и для соединений с другими таблицами (join). При необходимости добавления полей в таблицу, возможно, будет принято решение о создании новой таблицы и соединении ее с первой. Это может оказаться более правильным решением; просто нужно при этом учитывать, что каждый раз при доступе к полям этой таблицы нужно будет соединять ее с изначальной таблицей. Это может вызывать раздражение, и может работать более медленно при выборке данных из соединяемых таблиц. Но это является допустимым решением, если ваше видение структуры таблиц таково. Смысл в том, что при создании таблиц, перед их внедрением, необходимо иметь общую картину того, как таблицы будут использоваться.

---

#### Пример 1: Создание таблицы

##### Элементы проекта MORPHXIT\_DataDictionary

- Таблица, MyTable



В этом первом примере будет создана таблица для хранения информации о клиентах. Изначально, в таблицу будет добавлено всего несколько полей. По ходу данного раздела к ней будет добавлена дополнительная функциональность.

1. Создайте новую таблицу в узле *Tables* в дереве прикладных объектов, кликнув на нем правой кнопкой и выбрав *Создать Table*. Новая таблица с именем "Table1" будет создана в АОТ. Откройте окно свойств новой таблицы и измените имя на "MyTable".
  2. Разверните созданный узел, чтобы увидеть его под-узлы. Откройте еще одно окно с АОТ и перейдите к узлу *Data Dictionary/Extended Data Types [Расширенные типы данных]*. Найдите расширенный тип AccountNum и перетащите его мышкой в узел *Fields* созданной таблицы MyTable.
  3. Найдите расширенные типы CustName, CustGroupId и CurrencyCode и все их перетащите в узел *Fields* таблицы MyTable.
  4. Сохраните таблицу нажатием ctrl+s на узле MyTable. Таблица при этом синхронизируется с базой данных, и, вы создали вашу первую таблицу!
- 

При перетаскивании расширенных типов данных из АОТ в узел *Fields* таблицы, в ней были созданы одноименные поля. Это является самым простым способом создания полей. Можно выделить несколько расширенных типов данных одновременно и перетащить их все в узел полей таблицы. После этого уже можно будет задать индивидуальные свойства для каждого из созданных полей.

Теперь, используя обозреватель таблиц, можно добавить записи в созданную таблицу MyTable. При создании новой записи обратите внимание на то, что поля custGroupId и currencyCode имеют кнопку выбора из списка и внести в эти поля можно только значения из связанных с полями справочников. Причиной такого поведения этих полей является связь, которая настроена для них на расширенных типах к таблицам CustGroup и Currency соответственно. Без единой строчки кода вы создали таблицу, которая уже связана с другими таблицами!

Свойство таблицы **TableGroup** используется для объединения таблиц в группы. Об этом свойстве часто забывают при создании новых таблиц. Значение этого свойства выбирается в зависимости от содержимого таблицы, например, справочники или таблицы для хранения операций; Такие как справочник клиентов и таблица проводок клиентов, соответственно. Исследование существующих таблиц и анализ их содержимого поможет вам понять, какое значение выбирать для свойства TableGroup в каждом конкретном случае. Значением свойства TableGroup по умолчанию является "Miscellaneous", что означает, что таблица содержит разнородные данные. Так как выверка и экспорт данных могут фильтроваться по данному свойству, то его следует указывать всегда. Если неправильно задать это свойство для таблиц, таких как таблицы, содержащие проводки, запросы, содержащие соединения не смогут выполняться в

оптимальном по производительности режиме, так как ядро использует значение свойства `TableGroup` для определения необходимости использования литералов в запросе (`forceLiterals`). Более подробно о запросах рассказано в разделе **Введение в X++**.

Помимо этого, есть еще несколько свойств, значение которых необходимо указывать для создаваемых таблиц, в особенности свойства, связанные с безопасностью и производительностью. Наиболее часто применяемые значения свойств будут рассмотрены далее в этом разделе.

Полный обзор свойств, доступных в словаре данных Data Dictionary приведен в разделе **Свойства в Приложении**.

### Табличные переменные

Таблицы в коде X++ объявляются таким же образом, как и любая другая переменная. Для инициализации табличной переменной конкретным значением используются запросы с использованием ключевого слова *select*. Также можно присвоить одну таблицу другой, если при этом таблицы имеют одинаковый тип.

```
static void DataDic_OneCursor(Args _args)
{
    CustTable custTable, newCustTable;
;

    select firstonly custTable;

    newCustTable = custTable;
}
```

В приведенном выше примере с таблицей `CustTable` будет существовать только одна ссылка на строку таблицы при присвоении одной табличной переменной другой. Если далее одну из этих табличных переменных использовать для выбора другой строки, обе табличные переменные станут указателями на выбранную строку.

```
static void DataDic_TwoCursors(Args _args)
{
    CustTable custTable, newCustTable;
;

    select firstonly custTable;

    newCustTable.data(custTable);
}
```

Для того чтобы иметь две копии строки данных из таблицы, необходимо использовать метод `data()`, как показано в примере выше. В этом случае табличные переменные `custTable` и `newCustTable` обе будут содержать запись из таблицы.

### Временные таблицы

Временная таблица не содержит данных и не может быть синхронизирована с базой данных. Временная таблица может использоваться в операциях соединения (joins) и выборки данных (select), как любая другая таблица. Свойство таблицы **Temporary** определяет, является ли таблица временной. Любая таблица в словаре данных в АОТ может быть сделана временной указанием свойства *Temporary* “Да”. Предостережение: Установка свойства *Temporary* на таблице, в которой присутствуют данные, приведет к удалению всех данных из этой таблицы! Для того, чтобы можно было легко найти все временные таблицы в АОТ, все они созданы с префиксом «Tmp\*» в имени таблицы. Временные таблицы часто используются для сортировки данных. Вам может понадобиться отобразить специфическим образом отсортированные данные в форме или отчете, когда стандартным способом, используя запросы, сделать этого не удастся. Вместо этого, вы сможете создать временную таблицу с требуемыми полями и сортировкой, и уже ее использовать в приложении. Во временную таблицу данные следует вставлять в том порядке сортировки, который необходим на создаваемой форме или отчете. Временная таблица будет содержать данные на протяжении всего времени ее существования в текущей области видимости. Содержимое временной таблицы хранится в файловой системе во временном файле.

```
static void DataDic_TmpTable(Args _args)
{
    CustTable          custTable;
    CustTrans          custTrans;
    TmpCustLedger      tmpCustLedger;
;

    while select custTable
    {
        tmpCustLedger.accountNum = custTable.accountNum;
        tmpCustLedger.name       = custTable.name;
        tmpCustLedger.insert();
    }

    while select tmpCustLedger
    {
        info(strFmt("%1, %2", tmpCustLedger.accountNum, tmpCustLedger.name));
    }
}
```

В приведенном выше упрощенном примере коды и названия всех клиентов вставляются во временную таблицу TmpCustLedger. После этого, запускается цикл по содержимому TmpCustLedger с выводом кода и названия каждого клиента в Infolog. Данные, вставляемые во временные таблицы, обычно будут выбираться из одной или нескольких связанных таблиц.

При использовании временных таблиц появляются дополнительные накладные расходы, связанные с тем, что данные должны быть выбраны запросом и вставлены во временную таблицу. И только после этого должен формироваться запрос уже по временной таблице. Если вы создали форму, использующую сложный запрос, из-за которого форма медленно реагирует на пользовательские команды при навигации, использование временной таблицы может улучшить ситуацию. Конечно, первоначально придется заполнить данными временную таблицу, что замедлит открытие формы, но уже после загрузки данных и открытия формы, она будет работать значительно быстрее, обеспечивая тем самым более дружелюбный интерфейс.

Использование временных таблиц в формах более детально описано в разделе **Формы**.

```
static void DataDic_SetTmp(Args _args)
{
    CustGroup custGroup;
;

    custGroup.setTmp();

    custGroup.custGroup = "10";
    custGroup.name      = "Test customer 10";
    custGroup.insert();

    while select custGroup
    {
        info(strFmt("%1, %2", custGroup.custGroup, custGroup.name));
    }
}
```

Экземпляр таблицы может быть сделан временным из кода X++. Данные из таблицы удалены не будут, но табличная переменная будет пустой. В примере выше, таблица CustGroup устанавливается как временная. После этого в таблицу CustGroup вставляется одна запись с использованием объявленной табличной переменной. При просмотре записей временной таблицы в цикле while select будет напечатана только одна вставленная запись. Следует соблюдать осторожность при использовании стандартных таблиц в качестве временных. При использовании на рабочей базе данных, если случится, что кто-то прокомментирует строку, в которой устанавливается временность таблицы, то результат может быть фатальным. Примеры такого использования временных таблиц можно найти в стандартном приложении. Дополнительное время, которое вы потратите на создание временной таблицы в АОТ, может с лихвой окупиться в будущем.

Если появляется необходимость работы с временной таблицей, которая будет содержать большое количество записей, то лучше поместить обработку таблицы на сервере. Первая запись, вставляемая во временную таблицу, определяет место, где будет храниться временная таблица. Преимуществом может оказаться

создание класса для обработки временной таблицы, так как в свойствах класса можно указать, что он будет выполняться на сервере.

Более детально о том, где устанавливается место для выполнения кода, написано в разделе **Классы**.

## Системные таблицы

В отличие от прикладных таблиц, системные таблицы не подлежат изменению. То есть нельзя изменить модель данных системных таблиц. Эти таблицы используются ядром для управления различными системными задачами, такими как поддержка базы данных в синхронизации с приложением, управления лицензиями и информацией о пользователях. Перечень системных таблиц можно найти в AOT в узле *System Documentation/Tables*.

Системные таблицы могут использоваться как любые другие в коде X++. При использовании системных таблиц не стоит забывать, что данные, которые хранятся в них, критически важны для работоспособности системы. Системные таблицы с префиксом `sql*` в названии содержат информацию необходимую для синхронизации базы данных с приложением Ахарт. Изменение данных в этой таблице может привести к нестабильной работе системы.

При обновлении системной информации, такой как ввод или удаление пользователей или изменение лицензионных ключей, системные таблицы обновляются прикладными объектами. Однако большинство задач на системных таблицах выполняет ядро приложения. Стоит обратить внимание на одну из системных таблиц `SysLastValue`, в которой хранятся пользовательские настройки данные. Эта таблица повсеместно используется в приложении, так как в ней хранятся последние пользовательские настройки объектов системы. В коде явных объявлений таблицы `SysLastValue` может и не быть, так как таблица обернута в `runbase` средой или вызывается из класса `xSysLastValue`.

Более детально о пользовательских настройках можно прочесть в разделе **Введение в MorphX**.

## Таблица Common

Системная таблица `Common` является базовой для всех таблиц системы. Эта таблица не содержит данных, ее можно сравнить с временной прикладной таблицей. Таблица `Common` используется приблизительно так же, как базовый тип `anytype`. Ей можно присвоить табличную переменную любой таблицы. Это очень полезно при передаче записи таблицы в качестве параметра, если неизвестно заранее, какого типа будет передаваемая запись.

```
static void DataDic_Common(Args _args)
{
```

```
Common      common;  
CustTable   custTable;  
;  
  
common = custTable;  
  
if (common.tableId == tablenum(custTable))  
{  
    while select common  
    {  
        info(common.(fieldnum(custTable, name)));  
    }  
}  
}
```

В приведенном примере, таблица Common инициализируется CustTable. Добавлена дополнительная проверка для того, чтобы убедиться, что Common действительно теперь содержит записи CustTable. Далее производится цикл по всем записям CustTable с выводом названия клиента в InfoLog. Обратите внимание на то, что системная функция fieldnum() используется для получения кода поля name – название клиента. Если вам не известно название поля, которое будет выбираться, то можно использовать код поля в скобках вместо этого, даже в режиме выполнения.

### Системные таблицы Util\*

Системные таблицы с префиксом Util\* в названии, на самом деле, не являются настоящими таблицами. Вы не найдете ни одной из этих таблиц в базе данных. Они хранятся в файлах слоев, но отображаются в АОТ как системные таблицы. Таблицы Util\* содержат информацию обо всех объектах, содержащихся в АОТ, включая старые слои, даже если вы обновили ваше приложение, а поэтому в них содержатся огромные объемы данных. Поэтому, не следует использовать эти таблицы в сложных запросах, так как простое соединение даже двух таблиц типа Util\* уже вызовет большое замедление выборки. Также, при использовании таблиц Util\* не забывайте использовать наиболее эффективный в конкретном случае индекс. Доступные индексы можно просмотреть в древовидной структуре приложения.

Таблицы с префиксом Util\* используются в большом количестве инструментов, предоставляемых средой MorphX. При разработке собственных инструментов в MorphX, эти таблицы могут значительно облегчить разработку. Более детально об инструментах, которые используют таблицы с префиксом Util\*, смотрите в разделе **Инструменты разработки MorphX в Приложении**.

### Поля

Для создания полей в таблице могут использоваться как Расширенные Типы Данных, так и Перечислимые Типы. Для этого достаточно перетащить выбранный тип на узел Fields таблицы. Если ни один из существующих типов не

удовлетворяет вашим требованиям, то перед созданием полей необходимо создать требуемые типы данных. При выборе уже существующего расширенного или перечислимого типа, следует убедиться, что у него в свойствах определен конфигурационный ключ, так как ваши поля не будут видны пользователю, если у него отключен требуемый конфигурационный ключ.

Все поля таблицы следует создавать на основании расширенного или перечислимого типа. Вы, конечно, имеете возможность ручного добавления полей в таблицу, но, взяв за привычку добавление полей с помощью drag&drop, вы всегда сможете быть спокойны, что все необходимые свойства на поле будут заполнены. Преимуществом использования расширенных и перечислимых типов данных является то, что при смене значений свойств, таких как длина поля или выравнивание, это изменение отразится на всех полях, которые используют этот расширенный или перечислимый тип. Если конечный пользователь сделает правый клик на одном из полей и выберет «*Переименовать*» из формы «Паспорт Записи», то именно расширенный тип данных для кода клиента будет использован для поиска таблиц, в которых необходимо изменить значение соответствующих полей. Это – один из очень мощных инструментов Ахапта. На рынке программных продуктов всего несколько приложений могут похвастаться схожей по гибкости настройки функциональностью.

Согласно правилам хорошего тона разработки в Аксапте, все поля должны начинаться с буквы в нижнем регистре. Так как все расширенные и перечислимые типы начинаются с заглавной буквы, поля необходимо вручную переименовать после перетаскивания из узла типов в узел полей. Это действие занимает много времени, хотя особого смысла в реальной жизни не имеет. Большого выигрыша при просмотре в редакторе кода такое наименование не даст, так как некоторые поля созданы с использованием нижнего регистра, а некоторые – с использованием верхнего.

При просмотре полей в АОТ сложно не заметить иконки, которые расположены слева от имени поля. Эти иконки информируют о базовом типе поля. Точно такие же иконки используются расширенными и перечислимыми типами. Это очень удобная функциональность, особенно при выборе подходящего поля в редакторе X++.

### Свойства полей

В предыдущем примере с таблицей MyTable мы создали саму таблицу и поля в ней. После этого должны быть установлены определенные свойства на полях этой таблицы. Поле AccountNum определено как ключевое для таблицы MyTable. Для ключевых полей свойство **Mandatory [Обязательное]** всегда должно быть установлено в “Да”, так как это предотвратит сохранение записи с пустым значением в ключевом поле. Обязательные поля очень легко отличить из обозревателя таблиц или из любой формы, так как они подчеркнуты красной волнистой линией. Обычной практикой в Аксапте считается запрет редактирования ключевых полей после создания. Функция Переименования, упомянутая ранее, должна быть использована для изменения ключевого поля.

Свойство **AllowEdit** (**Доступ на редактирование**) следует установить в значение “Нет”, а свойство **AllowEditOnCreate** (**Доступ на редактирование при создании**) оставить со значением по умолчанию “Да”, оставив возможность ввода значения ключевого поля при создании записи.

Метки полей, которые видны конечному пользователю, наследуются от расширенного или перечислимого типа, на основании которого было создано поле. Вводить значение в свойстве Метка [Label] следует только в том случае, если нужно изменить значение, которое установлено для типа данных, в противном случае это свойство следует оставить пустым. Если вам часто приходится изменять метку для поля, следует подумать о создании нового, более подходящего расширенного типа данных.

Если в созданной таблице присутствуют поля, которые не несут пользы для конечного пользователя, такие как контейнерные поля, описанные ниже, следует их свойство **Visible** [**Видимое поле**] установить в значение “Нет”. Это спрячет выбранное поле от конечного пользователя. Стоит также отметить, что при этом это поле не будет отображаться и в обозревателе таблицы.

### Поля типа Контейнер

Поля базового типа контейнер часто используются для хранения таких данных, как точечные рисунки. Следует с большой осторожностью рассматривать добавление таких полей в уже существующие таблицы, так как этот тип поля обычно имеет намного больший размер по сравнению с остальными полями. Таблица CompanyInfo является примером использования такого типа поля; При добавлении большого рисунка в качестве логотипа компании производительность приложения может снизиться. Таблица CompanyInfo широко используется в системе, и так как обычно из нее выбираются все поля, то и логотип компании постоянно будет запрашиваться при выборе записи из CompanyInfo. В этом случае следует подумать об использовании небольших рисунков в качестве логотипа или создании связанной таблицы для хранения логотипа. Таблица CompanyImage является таблицей общего назначения, основной целью которой является хранение рисунков компании. Альтернативой этого решения является добавление ваших бинарных файлов как ресурсов в узле Resources в АОТ. Более полная информация о ресурсах приведена в разделе **Ресурсы**.

### Системные поля

У всех таблиц системы есть набор системных полей, которые автоматически добавляются к таблице при ее создании. Системные поля не могут быть просмотрены через АОТ. Перечень системных полей можно просмотреть, изучив список полей таблицы Common. Поля с префиксом modified\* и created\* не содержат никаких данных, если в свойствах таблицы не проставлены соответствующие им свойства. По умолчанию все эти поля отключены, так как это имеет незначительное, но все же, влияние на производительность.

Системное поле RecId – это поле, которое назначается каждой записи и является уникальным в системе. Это целочисленный счетчик, которые делят между собой



все таблицы. Системная таблица SystemSequences содержит информацию о значении счетчика. Каждый раз при вставке записи в какую-либо таблицу счетчик увеличивается на единицу, а поле recId инициализируется этим значением. Наличие recId является одной из основных причин использования COM для вставки записей в базу данных Ахарта из внешней системы. При непосредственной вставке записей в базу данных, значение recId пришлось бы контролировать самостоятельно.

Системное поле dataAreaId описано в начале раздела **Таблицы**.

## Группы полей

Поля, которые видимы пользователям приложения, всегда должны быть включены хотя бы в одну группу полей. Конечно, в MorphX можно использовать и поля, которые не входят ни в одну группу. Но, только включая поля в группы, вы получите полную мощь работы MorphX. Группы полей широко используются при создании форм и отчетов. Вместо того чтобы по одному перетаскивать поля в дизайн формы или отчета, следует добавлять туда целые группы полей. Если через какое-то время состав группы полей будет изменен, к примеру, будет добавлено еще одно поле, то оно будет автоматически добавлено везде, где использовалась эта группа полей.

Пользователи приложения могут просмотреть полный перечень полей таблицы, кликнув «Показать все поля» из формы Паспорт записи. Поля будут сгруппированы по группам полей. Поля, которые не входят ни в одну из групп, будут отображены в отдельной группе полей по умолчанию. Если же все поля будут разбиты на группы, то пользователь сможет получить более логичную и общую картину перечня полей таблицы.

---

### Пример 2: Добавление групп полей

#### Элементы проекта MORPHXIT DataDictionary


##### ➤ Таблица, MyTable

В предыдущем примере была создана таблица MyTable и в нее были добавлены поля. Сейчас мы добавим в таблицу MyTable группы полей.

1. Перейдите к узлу *AutoReport* в узле *Field Groups* на таблице MyTable. Выделите все 4 поля и перетащите их в узел *AutoReport*.
2. Перейдите к узлу *AutoLookup*. Перетащите поля *accountNum* и *custName* в узел *AutoLookup*.
3. Создайте новую группу полей “Identification”, правой кнопкой мыши кликнув на узле *Field Groups* и выбрав *Создать Group*. Название группы полей устанавливается посредством листа свойств. Добавьте поле *accountNum* к созданной группе полей.

4. Создайте новую группу полей “Overview” и добавьте в нее все 4 поля.
  5. Наконец добавьте новую группу полей “Details” и добавьте в нее поля custName, custGroupId и currencyCode.
  6. Сохраните таблицу.
- 

Возникает вопрос – зачем понадобилось так много групп полей для такого небольшого их количества: AutoReport и AutoLookup являются зарезервированными группами полей. Оставшиеся 3 группы используются для отображения в отчетах и на формах.

При нажатии кнопки печати  на форме, поля из группы AutoReport выводятся на печать. Если в этой группе нет ни одного поля, то отчет будет пустым. Эту возможность называют печатью авто отчета, и она может использоваться пользователями приложения для печати простых отчетов непосредственно из форм. Все основные поля следует добавлять в группу полей AutoReport. Группа полей AutoLookup может использоваться для указания полей, которые нужно показывать при выборе из выпадающего списка значений [lookup]. Если группа AutoLookup останется пустой, то в качестве полей для выпадающего списка будут использоваться поля из свойств таблицы TitleField1 и TitleField2 совместно с первым полем из каждого индекса таблицы. Формы выпадающих списков более детально описаны в разделе **Формы**.

Рекомендацией является создание также двух групп полей Identification [Идентификация] и Overview [Обзор]. Группа полей Identification должна содержать все ключевые поля таблицы. Если в таблице присутствует уникальный индекс, то в эту группу полей следует добавить все поля этого индекса. Группу Overview следует рассматривать как итоговый вид таблицы. Эту группу полей следует использовать на закладке Overview, которая присутствует на большинстве форм. Все основные поля, или хотя бы все обязательные поля следует поместить в эту группу полей.

Помимо группы Overview, все поля следует добавить в еще как минимум одну группу для того, чтобы получить логичное разбиение полей на группы. Группы полей Identification и Details являются логичным разбиением полей таблицы MyTable. Перед использованием групп полей на формах или отчетах, необходимо назначить им соответствующие метки. Метку для группы полей можно задать через окно свойств этой группы.

Группы полей могут содержать не только поля таблицы. Display и edit методы также могут быть включены в группу. Но такое решение имеет недостатки, так как методы, добавляемые в группу полей, идентифицируются последовательной нумерацией методов. При добавлении или удалении метода таблицы, методы будут перенумерованы, и необходимо будет выполнить проверку того, что

правильные `display` и `edit` методы помещены в группу полей. Использование методов описано в разделе **Методы**.

## Индексы

Любая таблица должна содержать, по крайней мере, один индекс. Если таблица содержит уникальный индекс, то можно будет запросить конкретную запись таблицы из базы данных. Наличие такого индекса, с другой стороны, не всегда возможно и не всегда необходимо. Операционные [Transaction] таблицы часто могут содержать записи с одинаковыми значениями, делая невозможным наличие уникального индекса. Так как операционные таблицы обычно часто используются и содержат большое количество записей, наличие в них уникального индекса сказалось бы на производительности. Суть в том, что чем лучше будет ваш индекс, тем производительнее будет система. Основные [Main] и групповые [Group] таблицы, такие как Справочник Клиентов и Групп Клиентов должны содержать уникальные индексы, тогда как большие операционные таблицы с большим количеством записей следует создавать без уникального индекса. Также следует брать во внимание, что создание большого количества индексов на таблице скажется на производительности, так что следует тщательно обдумывать каждый создаваемый индекс. Каждый раз при создании записи в таблице, база данных будет производить обновление индексов этой таблицы.

---

### Пример 3: Создание индексов

#### Элементы проекта MORPHXIT DataDictionary

##### ➤ Таблица, MyTable

Следующим шагом является создание индексов таблицы MyTable. В этом примере будет создано два индекса этой таблицы.

1. Перейдите к узлу *Indexes* в MyTable. По правому клику мыши выберите *Создать Index*, в результате чего будет создан индекс. Через окно свойств задайте имя индекса “AccountIdx”. Перетащите поле `accountNum` на индекс.
  2. Откройте свойства индекса AccountIdx и установите свойство **AllowDuplicates** в значение “Нет”.
  3. Создайте новый индекс GroupIdx. Добавьте в него поля `custGroupId` и `accountNum` в соответствующем порядке.
  4. Сохраните таблицу.
- 

В таблицу MyTable был добавлен один уникальный индекс установкой свойства AllowDuplicates в значение «Нет». Это предотвратит создание двух записей с одинаковым набором полей, которые включены в уникальный индекс. Если

таблица уже содержит данные, и после этого какой-то индекс делается уникальным, то при наличии повторяющихся записей Вы получите ошибку, информирующую о невозможности создания такого индекса. И даже если в текущей компании данных в таблице может и не быть, создание индекса будет невозможным, так как *ни одна* компания не должна содержать повторяющихся по ключевым полям записей. Возможно также создать более одного уникального табличного индекса, но это не особо приветствуется и рассматривается, как плохой дизайн базы данных. В дополнение, наличие нескольких уникальных индексов на таблице сделает ее использование более сложным при работе. На таблицы с уникальным индексом следует проставить также свойства PrimaryIndex и ClusterIndex. Только уникальный индекс может быть указан в качестве первичного индекса. Первичный индекс используется для сортировки данных при первоначальном запуске формы. При кэшировании таблицы также используется первичный индекс. Уникальный индекс указывается, как кластерный (ClusterIndex) для улучшения производительности. Кластерные индексы содержат как индекс, так и данные, тогда как обычный индекс содержит только отсортированный список ключевых полей. При выборке записи с использованием обычного индекса база данных сначала должна найти нужный индекс, и уже по нему искать запись.

## Связи [Отношения]

При нажатии кнопки Проводки на форме Клиентов будет открыта форма проводок клиента. При изменении текущей записи в форме клиентов проводки клиента также автоматически обновляются проводками нового клиента. Это удобная функциональность является одной из причин использования связей таблиц. Связи организуются между таблицами AOT и дают MorphX картину модели данных системы. Использование связей между таблицами также избавляет от необходимости написания большого количества кода. Вышеприведенный пример с проводками клиента не содержит ни одной строки кода.

---

### Пример 4: Добавление отношений между таблицами

#### Элементы проекта MORPHXIT DataDictionary

##### ➤ Таблица, CustTable

Будет создана связь таблицы CustTable с таблицей MyTable, что позволит связать код клиента со счетом из таблицы MyTable.

1. Найдите таблицу CustTable в AOT и перейдите к ее узлу *Fields*. Добавьте новое поле на основании расширенного типа данных AccountNum. Переименуйте его в “altCustAccountNum”.

2. Перейдите к группе полей Delivery таблицы CustTable и добавьте к ней поле altCustAccountNum.
  3. Перейдите к узлу Relations в таблице CustTable. Сделайте правый клик мыши по узлу *Relations* и выберите «Создать Relation». Переименуйте созданное отношение в “MyTable” через окно свойств отношения.
  4. По правому клику мыши на новом отношении MyTable выберите Создать -> Нормально. Перейдите к окну свойств отношения и выберите поле altCustAccountNum в свойстве **Field** и поле accountNum в свойстве **RelatedField**.
  5. Сохраните таблицу.
- 

В таблицу CustTable было добавлено новое поле и по нему настроено отношение с таблицей MyTable. Новое поле altCustAccountNum добавлено в группу полей Delivery. При открытии формы CustTable и переходе на закладку Настройки в группе полей Доставка появится созданное поле с меткой, унаследованной от расширенного типа данных AccountNum - Счет. Обратите внимание на то, что поле содержит возможность выбора значения из выпадающего списка. Она появилась вследствие того, что была настроена связь между таблицами MyTable и CustTable, кнопка выпадающего списка появилась в результате этой настройки автоматически. Теперь можно выбрать счет из таблицы MyTable, используя выпадающий список. При попытке ввода в это поле счета, который отсутствует в таблице MyTable, произойдет ошибка. Отношение, по умолчанию, проверяет, что вводимые значения присутствуют в справочнике таблицы отношения. При необходимости можно через окно свойств отношения отключить проверку наличия поля в справочнике.

Отношение созданное на таблице CustTable было типа Нормально и использовало одно поле в качестве связи. Любое количество полей может быть добавлено для создания отношения. Так как таблица MyTable содержит первичный ключ из одного поля, то одного поля будет достаточно. Все поля первичного индекса должны быть добавлены в отношении вида Нормально. Также существуют два других вида отношения: поле фиксировано, и поле ссылки фиксировано. Они используются для сужения поиска отношения и часто используются для фильтрации данных на основании перечислимого типа, или даже для определения таблицы для связи. Примером может послужить таблица LedgerJournalTrans, которая содержит ручную вводимые строки журналов. В зависимости от типа счета, при выборе его из выпадающего списка будет выбираться информация из различных таблиц, таких как План счетов, Клиенты или Поставщики. Если же посмотреть на отношения LedgerTable, CustTable и VendTable на таблице LedgerJournalTrans, то становится понятен принцип работы: к каждой таблице создана связь типа нормально по двум полям и одна связь типа поле фиксировано по типу счета.

В примере было показано, как настраиваются связи к разным таблицам. Также связь типа поле фиксировано может использоваться для фильтрации данных одной и той же таблицы, так как для одной таблицы может быть создано более одного отношения. Не стоит забывать, что при использовании связи типа поле фиксировано или связи типа поле ссылки фиксировано соответствующие условия будут добавлены в Ваш запрос.

Использование выпадающих списков на формах описано в главе **Формы**.

На Расширенном типе данных также можно настроить отношение к таблице. Но, если на таблице будет настроено отношение по тому же полю, то отношение на таблице будет доминировать над тем, которое настроено на типе данных. Создание отношений на нескольких объектах может запутать при первом знакомстве с MorphX. Использование Визуальное моделирование с MorphXplorer может помочь, так как на диаграммах отображаются отношения настроенные как на таблице, так и на типах данных.

## Действия при удалении [Delete Actions]

Для обеспечения целостности данных на таблицах настраивают действия при удалении. Если, к примеру, Вы создадите таблицу для хранения дополнительной информации о клиентах, то эта таблица будет связана со справочником клиентов по номеру счета клиента. Если же удалить пользователя из справочника клиентов, то данные в Вашей таблице с дополнительной информацией о нем останутся, хотя эта информация уже больше не нужна. Это нарушит целостность данных и может привести к нежелательным последствиям в будущем: К примеру, если бы Ваша созданная таблица содержала информацию о клиентских проводках, и был добавлен клиент с таким же кодом клиента, как был у удаленного когда-то клиента, то данные из связанной таблицы опять стали бы доступны и ассоциировались бы с новым клиентом, что неверно.

Для предотвращения таких ситуаций, действия при удалении будут или удалять данные из связанных таблиц при удалении клиента, или препятствовать удалению клиента, пока в связанных таблицах присутствуют данные этого клиента. Обычно, проводки по клиенту препятствуют удалению клиента, тогда как другая информация, касающаяся только клиента, такая как его личные данные, будет удалена вместе с клиентом.

---

### Пример 5: Добавление действий при удалении

#### Элементы проекта MORPHXIT DataDictionary project

- Таблица, CustTable

В этом примере будут созданы действия при удалении для таблиц CustTable и MyTable для обеспечения целостности данных.

1. Перейдите к узлу *DeleteActions* таблицы *MyTable*. Кликните по нему правой кнопкой мыши и выберите *Создать DeleteAction*. Далее с помощью окна свойств нового действия при удалении выберите в качестве таблицы *CustTable*. Действие, применяемое при удалении, установить следует в значение *Restricted*.
  2. Перейдите к таблице *CustTable* и добавьте новое действие при удалении с таблицей *MyTable*. Откройте окно свойств нового действия при удалении и выберите в свойстве *Table* таблицу *MyTable*. Установите действие в значение *Cascade*.
  3. Сохраните таблицу.
- 

Откройте форму *CustTable*, создайте нового клиента и выберите для него альтернативный номер счета из таблицы *MyTable*. После этого, попробуйте удалить из обозревателя таблицы *MyTable* номер счета, назначенный новому клиенту в качестве альтернативного. При попытке удаления Вы получите сообщение об ошибке. Действие при удалении “restricted” на таблице *MyTable* предотвращает удаление записей из связанной таблицы. Если же удалить созданного клиента и вновь просмотреть содержимое таблицы *MyTable* в обозревателе, то мы увидим, что альтернативный номер счета удаляемого клиента также был удален. Установка действия при удалении в значение *cascade* удаляет связанные записи из таблицы.

Действия при удалении используют отношения настроенные на таблице для определения необходимости удаления или предотвращения удаления связанной информации. Если не настроено отношение к таблице, которая выбрана в действии при удалении, то это действие не будет использоваться.

```
static void DataDic_DeleteActions(Args _args)
{
    MyTable myTable;
;

    ttsbegin;
    select forupdate myTable
        where myTable.accountNum == "10";

    if (myTable.validateDelete())
        myTable.delete();
    ttscommit;
}
```

При удалении записей из кода X++ для того, чтобы учитывать действия при удалении, необходимо соблюдать определенные правила. В этом примере выбирается запись из таблицы *MyTable*. Перед удалением записи из таблицы выполняется проверка на возможность удаления. Если бы запись просто

удалялась, без учета `validateDelete()`, то действие при удалении не было бы применено, что привело бы к противоречивости данных. Действие при удалении имеет еще одно возможное значение “Cascade + Restricted”. Этот механизм ведет себя, как и обычный `restricted`, при использовании из обозревателя таблицы или формы. Если же запись удаляется из кода X++ без вызова метода `validateDelete()`, то будет применено каскадное удаление записей из связанной таблицы. Применимо к приведенному выше примеру, это было бы сродни удалению записи клиента из формы клиентов с последующим удалением связанной записи в связанной таблице `MyTable`. Запутанно? Способ действия при удалении `Cascade + Restricted` почти не используется, и видимо для этого есть хорошие основания.

## Методы

Методы используются для добавления Вашего кода на X++ в приложение. Код методов также часто называют бизнес логикой системы. При изменении, добавлении или удалении записей таблицы выполняются различные методы по умолчанию в заданном порядке. У Вас есть возможность создания своих собственных методов и вызова их из одного из стандартных методов. Изменение стандартного метода называется его перекрытием. Для перекрытия метода перейдите к узлу *Methods* таблицы, сделайте правый клик мыши и выберите *Перекрыть метод*. Будет показан список стандартных методов, которые можно перекрыть. Если перекрыть, к примеру, метод `validateField()`, то в редакторе будет отображен нижеприведенный код:

```
public boolean validateField(fieldId _fieldIdToCheck)
{
    boolean ret;

    ret = super(_fieldIdToCheck);

    return ret;
}
```

Обратите внимание на вызов `super()`. Этот вызов выполняет код соответствующего метода класса-родителя. На самом деле, при перекрытии одного из методов таблицы, на самом деле вы перекрываете метод системного класса, а именно системного класса `XRecord`.

При добавлении логики в приложение предпочтительным является добавление кода в табличные методы и классы. Отсутствие бизнес логики в пользовательском интерфейсе позволит повторно использовать и обновлять код. Понятно, что полностью избежать кода на формах или отчетах не удастся, но следует стремиться не изменять данные из методов форм и отчетов. Это даст возможность подмены интерфейса пользователя в будущем, к примеру, web-дизайном с сохранением всей бизнес логики приложения.



Использование методов было рассмотрено очень кратко, только для базового понимания принципов работы с методами. В разделе **Классы** методы будут рассмотрены более детально.

### Основные методы

Во время изменения данных, будь то добавление, обновление или удаление записей, выполняются базовые табличные методы. При создании новой записи из обозревателя таблицы или формы, будет выполнен табличный метод `initValue()`. `InitValue()` используется для установки значений по умолчанию в поля.

```
public void initValue()
{
    super();

    this.custGroupId = "10";
}
```

Попробуйте перекрыть метод `initValue()` на таблице `MyTable` и добавьте приведенную строку кода. Откройте таблицу `MyTable` в обозревателе таблиц и нажмите `ctrl+n` для создания новой записи. Поле `custGroupId` будет иметь значение 10. Ключевое слово *this* используется в методе `initValue()` как ссылка на текущий объект. Нельзя изменить *this* на `MyTable`, так как в таком случае компилятор бы ожидал объявления переменной с таким именем в заголовке метода. Использование ключевого слова *this* четко отделяет объект, метод которого выполняется от других переменных, и эта нотация используется повсюду в MorphX.

---

**Примечание:** Чтобы проследить последовательность выполнения методов, можно перекрыть те методы, которые хотите проверить и добавить вывод сообщения в InfoLog в каждом из перекрытых методов. Или же, можно перекрыть метод и установить точку останова на `super()`. При достижении точки останова, выполнявшиеся методы можно будет посмотреть в окне содержимого стека отладчика.

---

Каждый раз при изменении значения одного из полей вызывается табличный метод `modifiedField()`. Метод полезен тем, что в нем можно инициализировать значения других полей таблицы определенными значениями в зависимости от значения изменяемого поля.

```
public void modifiedField(fieldId _fieldId)
{
    switch (_fieldId)
    {
        case fieldnum(MyTable, custGroupId) :
            this.custCurrencyCode = "";
            break;
        default :
            super(_fieldId);
    }
}
```

```
}
```

В примере, приведенном выше, при изменении поля `custGroupId` значение поля `custCurrencyCode` сбрасывается в пустое значение. Метод `ModifiedField()` получает на вход номер (*fieldId*) поля, которое изменяется. Для определения, какое именно поле изменяется, используется блок *switch*. Если же ни одно из полей перечисленных в блоках *case* не активно, то вызов передается на `super()`. В примере блок *switch* содержит только один блок *case*, поэтому вместо этого мог быть использован блок *if-else*. Но использование сразу блока *switch case* позволит в дальнейшем легко расширить количество полей, при изменении которых выполняются дополнительные строки кода.

Этот метод существует, начиная с версии Аксапта 3.0. До этого, проверки при изменении значения полей необходимо было делать непосредственно на формах. Из-за этого большое количество бизнес логики, которую следует писать на таблицах, приходилось располагать на формах. По этой причине до сих пор можно найти много кода с проверкой изменения полей на формах. Идеальным решением было бы выделить весь общий код и перенести его на таблицу. Но, иногда, если логика работы специфична для конкретной формы, можно добавлять ее непосредственно в методах формы.

```
print this.orig().custCurrencyCode;
```

При изменении поля имеется удобная возможность определения значения поля до изменения. Это значение является значением последней сохраненной транзакции. Метод `orig()` используется для получения этого сохраненного значения. `Orig()` возвращает экземпляр текущей таблицы. Конкретное поле из `orig()` получается указанием поля после точки. При подтверждении транзакции курсор, возвращаемый `orig()` будет обновлен новым значением.

Табличный метод `validateField()` чем-то схож с методом `modifiedField()`. Оба метода получают в качестве параметра идентификатор текущего поля. Метод `modifiedField()` используется для инициализации значений других полей и не возвращает никакого значения, тогда как метод `validateField()` используется только для проверки и возвращает `true` или `false`. Если `validateField()` вернет значение `false`, то пользователь не сможет изменить значение этого поля. Очень важно использовать соответствующие методы для проверки и обновления значений полей, так как благодаря этому код намного легче читать.

```
public boolean validateField(fieldId _fieldIdToCheck)
{
    boolean ret;

    ret = super(_fieldIdToCheck);

    if (ret)
    {
        switch (_fieldIdToCheck)
        {
```

```
case fieldnum(MyTable, custName) :  
    if (strlen(this.custName) <= 3)  
        ret = checkFailed("Customer name must be longer than 3 characters.");  
    }  
}  
  
return ret;  
}
```

Вызов `super()` осуществит проверку корректности введенного значения при наличии на таблице связи по активному полю. Обязательное поле также будет проверено. В приведенном выше примере, при выполнении всех проверок в `super()` он вернет значение `true`, после чего будет выполнена дополнительная проверка значения поля `custName`. Если длина введенного значения меньше 3 символов, то в Infolog будет выведено предупреждающее сообщение, сообщающее об этом. Глобальный метод `checkFailed()` (он принадлежит классу `Global`) обычно используется для помещения предупреждающих сообщений в Infolog.

При сохранении записи сначала вызывается табличный метод `validateWrite()`. Метод `validateField()` проверяет значение, вводимое в поле, а `validateWrite()` проверит только обязательные поля. Проверки, проводимые методом `validateWrite()` такие же, как и вызов `super()` в методе `validateField()`. Если поле не является обязательным для заполнения, пользователь может не вводить его значение перед сохранением. Поэтому, если ваше условие не относится к значению конкретного поля таблицы, то его можно поместить в метод `validateWrite()`. Синтаксис метода `validateWrite()` похож на метод `validateField()`. После вызова `super()` следует проверить значение, которое вернул этот вызов перед проведением дальнейших проверок.

Если метод `validateWrite()` возвращает значение `true`, метод `insert()` или `update()` будет выполнен следом. Если сохраняемая запись новая, то вызывается метод `insert()`. Метод `Update()` вызывается в том случае, если системное поле `recId` уже имеет значение, что означает, что запись уже была сохранена ранее. `Insert()` и `update()` редко перекрываются на таблице. К этому моменту уже должны были пройти проверки, и установлены значения всех необходимых полей. Если Вам необходимо перекрыть метод `insert()` или `update()`, возможно вы движетесь не в том направлении. Но, если Вам необходимо убедиться, что поле будет иметь определенное значение при добавлении в таблицу, то это значение можно установить перед вызовом `super()` в методе `insert()`. Также могут существовать дополнительные ситуации, требующие перекрытия этого метода; к примеру, если есть необходимость синхронизации содержимого этой таблицы с другой таблицей. Примерами такой синхронизации могут служить методы таблиц `CustTable` и `EmplTable`. Такое решение не приветствуется, но оно может быть единственным рациональным именно в Вашей ситуации.

При удалении записи из таблицы имеет похожую последовательность вызовов. При удалении записи сначала вызывается метод `validateDelete()`. Если он вернет значение `true`, то вызывается метод `delete()`, который и удаляет запись. Удаления

записей касаются те же правила, что и добавления и обновления данных. Поэтому, следует выполнять необходимые проверки до вызова метода `delete()`. `ValidateDelete()` также дополнительно проверяет, разрешено ли удаление строки с учетом действий при удалении [*DeleteActions*].

Ввод данных из X++ требует немного большего, чем обычный ввод через пользовательский интерфейс форм, так как будут выполняться только табличные методы проверки. Если принудительно не вызвать метод проверки, то никакой проверки при добавлении, обновлении или удалении данных выполнено не будет. Единственной проверкой будет проверка на уникальность индекса.

```
static void DataDic_InsertRecord(Args _args)
{
    MyTable    myTable;
;

    ttsbegin;
    myTable.initValue();
    myTable.accountNum      = "100";
    myTable.custName        = "Alt. customer id 100";
    myTable.custCurrencyCode = "USD";

    if (myTable.validateWrite())
        myTable.insert();
    ttscommit;
}
```

В примере показано, как следует использовать табличные методы для вставки записи в таблицу `MyTable`. Сначала вызывается метод `InitValue()` для установки значения поля `custGroupId`. Далее, запись будет добавлена в таблицу только в случае, если метод `validateWrite()` вернет значение `true`. Так как все обязательные поля заполнены, то запись будет успешно добавлена.

Вместо использования вызова `insert()` можно вызывать метод `write()`. Этот метод обновит существующую запись, а если записи еще не существует, то она будет добавлена.

Вызов методов проверки требует времени, что снижает производительность – поэтому, иногда следует рассматривать возможности вызова других методов для проверки вводимых данных. Также, можно избежать выполнения табличных методов `insert()`, `update()` и `delete()`, и вставить запись непосредственно в таблицу. Это делается с помощью соответствующих методов с добавлением префикса `do*`. Эти методы нельзя перекрыть, и вызвать их можно только из кода X++. Но, если Вы собираетесь использовать эти методы, чтобы пропустить всю логику, заложенную в методы на таблице, то Вам самим необходимо следить за обеспечением целостности данных.

Ключевые слова `delete_from`, `insert_recordset` и `update_recordset` посылают всего один запрос базе данных с клиента при обработке большого количества записей. Перекрытие методов `insert`, `update` и `delete` повлияет на способ выполнения этих

команд повышенной производительности, так как при этом записи будут обрабатываться строка за строкой по одной.

На большинстве таблиц можно найти методы `find()` и `exist()`. Эти методы не являются перекрытыми. Эти статические методы обычно создаются для выборки единственной записи с использованием уникального индекса. Рекомендуется добавлять эти два метода на таблице при ее создании, так как рано или поздно они вам все равно понадобятся. Все поля уникального индекса должны быть переданы в эти методы в качестве параметров. `Find()` используется для получения записи из таблицы. Это может использоваться, к примеру, в методе `initValue()` для установки значения какого-то из полей, скажем, из таблицы параметров модуля. `Exist()` используется для определения существования записи, удовлетворяющей параметрам уникального индекса, в таблице. Проверка такого рода часто осуществляется перед добавлением записи в таблицу.

Примеров реализации этих методов очень много, достаточно просто пройти по таблицам в АОТ. Методы, которые используются для подобных операций, но без использования уникального индекса, не следует называть `find()` или `exist()`. Считайте, что эти методы зарезервированы и используйте суффиксы в названиях методов, как, к примеру, метод `findVoucherDate()`.

### Система Управления Транзакциями

Как видно из названия, Система Управления Транзакциями, вместо которого обычно используется аббревиатура TTS, используется для контроля транзакций. Суть этого контроля заключается в обеспечении завершенности всех операций внутри транзакции перед вставкой данных в базу данных. Этот контроль критичен для целостности реляционной базы данных и особенно для ERP системы. В процессе разnosки накладной необходимо быть уверенным в том, что сбой системы не приведет к порче данных. TTS гарантирует, что все операции с базой данных внутри цикла транзакции будут успешно завершены; в противном случае никаких изменений не будет произведено.

Каждый раз при выполнении операции `insert`, `update` или `delete` следует использовать TTS. Более того, операцию обновления записи невозможно выполнить вне транзакции. Транзакция открывается с помощью ключевого слова *ttsbegin*. Ключевое слово *ttscommit* запишет все операции, выполненные внутри транзакции, в базу данных. Транзакция может быть отменена ключевым словом *ttsabort*, которое откатит состояние базы данных до значения, которое было до начала транзакции. При сбое системы, *ttsabort* автоматически вызывается базой данных. Пример использования транзакций был приведен выше при вставке записи в таблицу `MyTable`.

Все вышесказанное не означает, что каждый раз, когда вам необходимо добавить запись в базу данных, следует открывать новую транзакцию. Если ваш код вызывается другим методом, который уже открыл транзакцию, новую транзакцию открывать уже необязательно. Добавление новой транзакции в уже открытой открывает новый уровень транзакции. Каждый уровень должен быть завершен

собственным ключевым словом `ttscommit`. Следует учесть, что если в коде будет присутствовать несколько уровней вложенности транзакции, и количество выражений `ttsbegin` и `ttscommit` не будет совпадать, то произойдет ошибка транзакций. Обычно такого рода ошибки получаются из-за отсутствия необходимого количества `ttscommit`. Для исправления ошибки транзакций можно выполнить задание `[job]`, которое закрывает транзакцию с помощью `ttscommit`. Конечно, источник ошибки придется все равно устранить, но возможно необходимо сохранить данные из транзакции. Альтернативой этого решения является перезапуск клиента Акспта.

```
static void DataDic_UpdateRecord(Args _args)
{
    MyTable    myTable;
    ;

    ttsbegin;
    select forupdate firstonly myTable;
    myTable.custName = "Customer updated";

    if (myTable.validateWrite())
        myTable.update();
    ttscommit;
}
```

При необходимости обновления записи она должна выбираться на обновление в теле транзакции, на что указывает ключевое слово *forupdate*. Часто встречающейся ошибкой использования команды *forupdate* является выборка данных на обновления до открытия транзакции. Выполнение такого кода привело бы к ошибке. В вышеприведенном примере, из таблицы `MyTable` выбирается первая запись, и значение поля `custName` изменяется на новое, после чего транзакция завершается с помощью `ttscommit`..

## 3.2 Карты соответствия (Maps)

В Ахарт, определенная функциональность модуля Расчеты с Поставщиками и модуля Расчеты с Клиентами очень схожи, к примеру, основные справочники клиентов или поставщиков, таблицы проводок и настроек. Функциональность журналов для ручного ввода документов ГК также существует в нескольких модулях. В этой схожей функциональности бизнес-логика также имеет мало отличий, что позволяет повторно использовать большую часть кода. Но, несмотря на схожесть таблиц и полей в них, названия скорей всего отличаются. Вот здесь и находят применение карты соответствия.

Карты соответствия часто называют табличными картами соответствия для того, чтобы не путать их с объектами базового класса `Map`.

Наиболее распространенная карта соответствия в `MorphX` - это `AddressMap`. Адресная информация, используемая во многих таблицах, и проверка заполнения

информации, такой как индексы, будет мало отличаться для клиентов и поставщиков. Настройка соответствия всех таблиц, которые используют адрес, в одной карте соответствия позволяет повторно использовать код, так как не придется учитывать специфические названия полей в каждой из таблиц. Карты соответствия содержат такие же узлы, как и таблицы, плюс дополнительно узел *Mapping*, в котором настраиваются соответствия полей. Карта соответствия может использоваться в коде X++, как любая другая таблица или объект, такой как форма или отчет. Только при использовании карты соответствия на форме или в отчете, стоит задуматься о создании групп полей в этой карте соответствия. В противном случае, обычно просто создают поля, так как все свойства уже настроены на соответствующих таблицах.

---

#### Пример 6: Создание карты соответствия

##### Элементы проекта те MORPHXIT DataDictionary

###### ➤ Карта соответствия, MyMap

Будет создана карта соответствия, в которой будут созданы соответствия общих полей для таблиц *CustTable* и *MyTable*.

1. Перейдите к узлу *Maps*, сделайте по нему щелчок правой кнопкой мыши и выберите *Создать Map*. Переименуйте созданную карту соответствия в “MyMap” через панель свойств.
2. Перетащите расширенные типы данных *AccountNum*, *CustName*, *CustGroupId* и *CustCurrencyCode* в узел *Fields* карты соответствия *MyMap*.
3. Сохраните изменения в *MyMap*, сделайте по ней правый щелчок и выберите *Восстановить*.
4. Перейдите к узлу *Mappings*, Сделайте правый клик и выберите *Создать Mapping*. Откройте панель свойств созданного соответствия и выберите таблицу *CustTable*. Будет создана строка для каждого из полей карты соответствия *MyMap*. Через панель свойств, для каждого из полей в соответствии укажите связанное с ним поле из выбранной таблицы, указывая его в свойстве **MapFieldTo**. Связанными полями из таблицы *CustTable* являются поля *accountNum*, *name*, *currency* и *custGroup*.
5. Повторите шаг 4 для таблицы *MyTable*.
6. Перейдите к узлу методов *Methods* и создайте нижеприведенный метод:

```
void listRecords(MyMap _myMap)
{
;
while select _myMap
{
```

```
        info(strFmt("%1, %2", _myMap.accountNum, _myMap.custName));  
    }  
}
```

## 7. Сохраните изменения.

---

В примере была применена операция Восстановления карты соответствия после добавления полей. Каждый раз при изменении узла `fields` карты соответствия, необходимо Восстановить ее, иначе изменения не будут учтены до следующего перезапуска клиента Аксапты.

На карту соответствия был добавлен метод, который выводит записи из связанных таблиц. Это простой метод, который печатает значения текущей записи, которой инициализирована карта соответствия. Поэкспериментируйте с ним, создав `job` со следующим кодом:

```
static void DataDic_TestMyMap(Args _args)  
{  
    MyMap    myMap;  
    CustTable custTable;  
    MyTable  myTable;  
;  
    myMap.listRecords(myTable);  
}
```

Обратите внимание, что карта соответствия объявляется точно так же, как и любая таблица. Понять, что это карта соответствия, можно только по имени. Поэтому рекомендуется добавлять суффикс `*Map` к имени карты соответствия, благодаря чему код станет более читабельным. Созданный метод карты соответствия вызывается с параметром `MyTable`. В результате такого вызова будут выведены в `InfoLog` все записи таблицы `MyTable`. Карта соответствия не содержит записей, можно сказать, что это специализированная таблица. Она может быть проинициализированная переменной любой из таблиц, на которые настроены соответствия.

Методы карты соответствия похожи на методы таблицы. Есть возможность установки начальных значений, реакции на изменения полей, проверки перед сохранением или удалением. Вызов стандартного метода `insert()` карты соответствия приведет к вызову одноименного метода на связанной таблице. Это можно использовать для создания метода для всех таблиц. Но в этом случае необходимо убедиться, что метод называется одинаково для всех поставленных в соответствие таблиц. Примером такого использования может служить метод `CustVendTrans.existInvoice()`. Этот метод вызывает соответствующий метод `existInvoice()` сопоставленной таблицы, что позволяет иметь различные проверки для таблиц, сопоставленных карте.



### 3.3 Представления

В реляционной базе данных информация хранится в большом количестве таблиц для предотвращения хранения повторяющихся данных и для увеличения производительности. Все это хорошо сказывается на управлении данными, но выборка данных для отчетов становится более сложной, так как чаще приходится выбирать данные из нескольких таблиц сразу. Для облегчения выборки данных возможно создание представлений. Представление является результатом соединения двух или более таблиц (используется `inner join`). Представления доступны только для чтения и поддерживают возможность использования агрегатных функций. Представления могут использоваться как альтернативный вариант источника данных в отчетах. Но основным предназначением представлений является обеспечение выборки данных для OLAP кубов. Представления синхронизируются с базой данных. Это облегчает чтение данных из таблиц Ахарта при использовании внешних инструментов, так как выборка производится непосредственно из базы данных, без использования COM интерфейса.

OLAP кубы в Ахарта заполняются с использованием сервисов Microsoft SQL Server Analysis Services. Описание использования технологии OLAP выходит за рамки данной книги. Более детальную информацию об использовании OLAP в Ахарта можно найти в стандартных руководствах по Ахарта.

---

#### Пример 7: Создание представления

##### Элементы проекта MORPHXIT DataDictionary

##### ➤ Представление, MyView

В этом примере будет создано представление для суммирования проводок по клиенту и вывода информации о клиенте.

1. Перейдите к узлу *Views*, сделайте по нему щелчок правой кнопкой мыши и выберите *Создать View*. Переименуйте представление в "Myview" через панель свойств.
2. Перейдите к узлу *Metadata/Data Sources*, сделайте правый клик и выберите *Создать Data Source*. Откройте панель свойств созданного источника данных и выберите таблицу CustTable в свойстве **Table**.
3. Раскройте источник данных CustTable и перейдите к узлу *CustTable/Data Sources*. Добавьте еще один источник данных по щелчку правой кнопкой мыши. Выберите в качестве источника данных таблицу CustTrans.
4. Откройте панель свойств источника данных CustTrans и установите значение «Yes» в свойстве **Relations**. Раскройте узел *Relations* источника данных CustTrans и убедитесь, что была автоматически добавлена связь двух таблиц.

5. Откройте узел *Metadata* в новом окне, выбрав соответствующий пункт в выпадающем по правой кнопке мыши меню. Перейдите в открывшемся окне к узлу *CustTable/Fields*. Выберите поля *AccountNum*, *Name* и *CustGroup* и перетащите выделенные поля в узел *Fields* на первом уровне древовидной структуры представления.
  6. Повторите шаг 5, перетащив поле *AmountMST* из источника данных *CustTrans/Fields*. Откройте панель свойств для этого поля и установите свойство **Aggregation** в значение *Sum*.
  7. Сохраните созданное представление.
- 

При создании представления для выборки данных из таблицы, представление определяется точно таким же образом, как и стандартный запрос Ахapta. Поля, которые будут использоваться в представлении, должны быть выбраны из соответствующих таблиц. В приведенном примере была использована агрегатная функция для суммирования проводок по клиенту. При использовании агрегатных функций в представлении записи будут выбраны с использованием *group by*. В примере были выбраны суммы по проводкам всех клиентов, с группировкой по клиентам. Если бы поле *TransDate* из таблицы *CustTrans* было также выбрано, без использования агрегатной функции, то поле *AmountMST* было бы сгруппировано и вычислено по клиенту и дате проводки, а не просто по клиенту. Обратите внимание, что агрегированные поля в представлении автоматически получают в имени префикс с названием использованной агрегатной функции. Результат создания представления *MyView* можно увидеть в обозревателе таблиц. Перечислены все клиенты из таблицы *CustTable* с просуммированными проводками по каждому из клиентов (поле *AmountMST* из *CustTrans*).

Представления могут использоваться как обычные таблицы за исключением некоторых ограничений. Представления не могут быть использованы в Отношениях, Действиях при удалении и Табличных коллекциях. Представление может быть использовано как альтернатива таблиц из кода X++ или как источник данных. При создании отчета можно использовать представления во избежание необходимости выполнения расчетов сумм, нужных в вашем отчете. Но, с другой стороны, это наложило бы некоторые ограничения на пользователя приложения, так как обычно у них была возможность в режиме реального времени указывать настройки для расчетов сумм.

### 3.4 Расширенные типы данных

Расширенные типы данных [Extended Data Types] являются центральной частью MorphX. Их следует всегда использовать вместо базовых типов, будь то добавление поля в таблицу или объявление переменной из кода X++. Основной

причиной этого является более простая поддержка ваших модификаций системы, а также однообразное отображение поля в системе.

Расширенный тип данных наследуется от базового типа данных или от другого расширенного типа. Уровни вложенности наследования типов не ограничены. Отличием базового типа данных от расширенного является наличие у последнего панели свойств, таких как метка, длина, правое и левое выравнивание. На расширенном типе данных могут быть также настроены отношения к таблицам.

Перед созданием расширенного типа данных следует убедиться, что в системе уже нет EDT, который бы удовлетворял Вашим требованиям. При создании таблицы и добавлении в нее поля с возможностью выбора из выпадающего списка кодов номенклатуры из таблицы InventTable, следует использовать EDT ItemId. Используя ItemId, не придется заново указывать метку или отношение к таблице InventTable, так как эти свойства уже определены на расширенном типе.

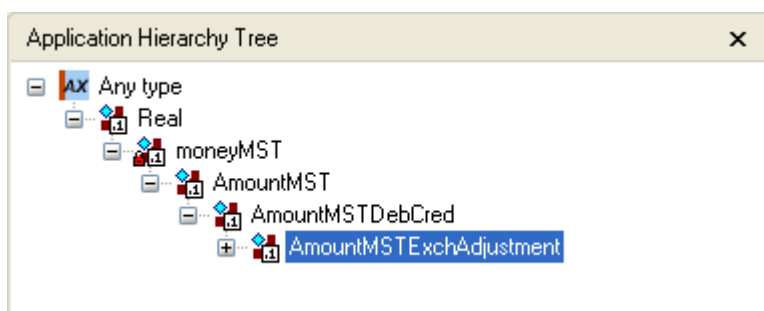


Рисунок 15: Просмотр расширенного типа данных в Иерархии Прикладных Объектов

Выбор расширенного типа для использования не является тривиальной задачей, особенно если EDT несколько раз наследовался. Иерархия объектов облегчает задачу выбора. Просмотреть иерархию объекта можно из всплывающего меню Add-Ins. В иерархии расширенного типа будут отображены только базовый тип данных и предки в цепочке наследования. Для работы этой функциональности необходимо наличие обновленных перекрестных ссылок.

Некоторые расширенные типы наследуются от системных расширенных типов, которые можно найти в узле AOT *System Documentation/Types*. Региональные настройки, такие как суммы, время и др. также созданы как системные типы.

---

#### Пример 8: Создание расширенного типа данных

##### Элементы проекта MORPHXIT DataDictionary

- Расширенный Тип Данных, DataDic\_AltCustAccount

В примере будет создан расширенный тип данных, который будет наследником уже существующего типа. Созданный расширенный тип данных снимет необходимость существования отношения на таблице CustTable.

1. Перейдите к узлу *Extended Data Types*, сделайте по нему правой клик и создайте новый тип на основании String. Назовите созданный тип данных "DataDic\_AltCustAccount" через панель свойств EDT.
  2. Присвойте расширенному типу метку "Alt. customer" и help text [подсказка] "Identification for alternative customer account".
  3. Унаследуйте расширенный тип данных DataDic\_AltCustAccount от типа AccountNum, указав его в свойстве **Extends**.
  4. Раскройте подузлы созданного типа DataDic\_AltCustAccount и перейдите к узлу *Relations*. По правому клику добавьте новое отношение с помощью команды *Создать* и укажите тип *Normal*. Через панель свойств выберите таблицу MyTable и поле accountNum в качестве связующего.
  5. Сохраните расширенный тип данных и дождитесь завершения синхронизации базы данных.
  6. Перейдите к таблице CustTable и найдите поле altCustAccountNum. Откройте панель его свойств и укажите в свойстве **ExtendedDataType** название созданного расширенного типа - DataDict\_AltCustAccount.
  7. Перейдите к узлу Relations таблицы CustTable и найдите отношение к таблице MyTable. Нажмите клавишу Del для удаления отношения к таблице MyTable.
  8. Сохраните таблицу CustTable.
  9. Измените свойство Extended Data Type для поля accountNum таблицы MyTable аналогично тому, как это сделано в шаге 6 и сохраните MyTable.
- 

Мы создали новый расширенный тип данных для обозначения альтернативного кода клиента. Табличное отношение, которое было удалено на таблице CustTable, являлось лишним, так как созданный расширенный тип теперь самостоятельно будет управлять этим отношением. Настройка отношений на расширенных типах данных является более предпочтительной, так как не придется настраивать отношения на всех таблицах, в которых используется данных расширенный тип данных. Расширенный тип данных также был изменен для поля accountNum таблицы MyTable. Кнопка всплывающего списка не будет добавлена на поле accountNum, так как таблица в отношении – это таблица, которая содержит поле accountNum. Этот тип отношения еще называют отношением само на себя. Метка и подсказка были указаны для созданного расширенного типа. Если никакие метки для расширенного типа не указать, то они наследуются от EDT AccountNum. Если Вы находите в АОТ расширенный тип, который Вам подходит по всем свойствам, кроме одной из меток, то следует создать новый расширенный тип данных. Не следует использовать существующий расширенный тип данных,

изменив у него необходимую метку на уровне поля в таблице. При ссылке на это поле из кода X++, будет использоваться информация из расширенного типа данных, а соответственно значение метки поля будет недоступно. Более детально этот процесс описан при знакомстве с `display` и `edit` методами в главе **Формы**.

---

**Примечание:** При сохранении расширенного типа данных, который наследуется от существующего EDT, будет проведена синхронизация всей базы данных. Если необходимо создать большое количество расширенных типов, разумным будет прерывать синхронизацию с помощью комбинации клавиш `ctrl+break` и выполнить ее только после создания последнего необходимого расширенного типа данных.

---

Поля могут быть созданы как на основании расширенных типов, так и на основании перечислимых типов. Тогда возникает вопрос: зачем присутствует возможность создавать расширенный тип данных на основании перечислимого типа? Ответ на этот вопрос прост: Только расширенные типы данных могут быть использованы при добавлении поля в диалог. Если создаваемое в диалоге поле должно быть перечислимого типа `NoYes`, то необходимо указывать тип `NoYesId` в качестве расширенного типа для создаваемого поля. Диалоговые окна создаются с помощью классов и более детально описаны в главе **Классы**.

## Расширенный тип данных из нескольких элементов

Определение расширенного типа данных как массива из нескольких элементов является одним из мощных инструментов работы с EDT. Каждый элемент массива в расширенном типе данных будет храниться в базе данных как отдельное поле. Как в AOT, так и в коде X++ поле, основанное на расширенном типе данных с несколькими элементами, будет выглядеть и использоваться точно так же, как и любое другое поле.

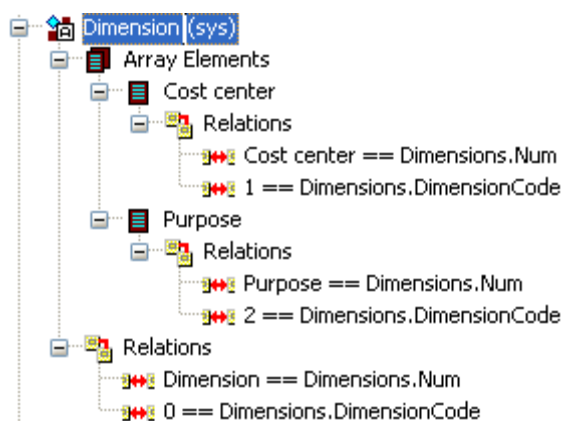


Рисунок 16: Расширенный тип данных **Dimension**

Узел *Array Elements* используется при добавлении в расширенный тип данных более одного элемента. Первым элементом массива будет тип, который создавался, как стандартный EDT. Все последующие элементы будут добавлены как подузлы *Array Elements*. Для них можно указать только метки `Label` и `Help`

Text. Все остальные свойства наследуются от первого элемента массива. Для каждого из элементов массива могут быть настроены отдельные отношения к таблицам.

Наиболее распространенным расширенным типом, использующим эту функциональную возможность, является EDT Dimension, который повсеместно используется в приложении при группировке данных. Расширенный тип Dimension состоит из 3 элементов массива, на каждом из которых настроено свое отдельное отношение к таблице. При необходимости добавления дополнительной финансовой аналитики в вашу компанию, необходимо всего лишь изменить расширенный тип данных Dimension. Все объекты, включая таблицы, формы и отчеты будут автоматически содержать новую аналитику без необходимости написания ни одной строки кода.

```
static void DataDic_EDTArray(Args _args)
{
    CustTable    custTable;
;

    select firstly custTable;

    info(strfmt("%1, %2, %3", custTable.dimension[1],
                                custTable.dimension[2],
                                custTable.dimension[3]));
}
```

Как видно из примера, здесь используется точно такая же нотация, как и при обращении к элементам обычного массива расширенных типов данных. В приведенном примере 3 аналитики из таблицы CustTable последовательно выводятся на экран для первой записи таблицы. При поиске этих полей в таблице CustTable, мы найдем только одно поле, Dimension. Поэтому номер элемента массива, который нужно использовать, необходимо указывать вручную. Если обращаться к фиксированному набору элементов массива, как в приведенном примере, то при добавлении нового элемента массива он уже не будет выводиться на экран. Поэтому более правильным решением было бы организовать цикл по всем элементам массива, а не только по первым трем.

Если же посмотреть на поля источника данных в запросе, то мы увидим противоположную картину. Источник данных будет отображать по полю на каждый элемент массива. Dimension всегда добавляется в одноименную группу полей, и уже они используются на формах, отчетах, поэтому при добавлении еще одного элемента массива изменения сразу будут отражены и на этих прикладных объектах системы.

### 3.5 Перечислимые типы

Для того чтобы разбить данные на категории, можно пойти двумя путями. Создать связанную таблицу, такую как Номенклатурные Группы, которая

используется для группировки номенклатур. Однако если набор возможных категорий фиксирован или же пользователи приложения не должны иметь право изменять категории, следует использовать перечислимый тип. Хорошим примером является Тип номенклатуры из номенклатурного справочника.

Перечислимый тип может содержать ограниченное количество элементов, которое не должно превышать 255. Несколько перечислимых типов в системе имеют большое количество элементов, к примеру, перечислимый тип LedgerTransTxt. Однако большинство перечислимых типов содержат всего пару элементов. В базе данных элементы перечислимых типов хранятся в виде целочисленных значений. Значения перечислимых типов обычно начинаются с 0 и имеют последовательную нумерацию. Свойство **EnumValue** отображает то значение, которое будет сохранено в БД для текущего элемента перечислимого типа. При добавлении нового элемента к одному из стандартных перечислимых типов, следует пропустить несколько значений, указав в свойстве EnumValue значение больше того, которое предлагается по умолчанию. Примером такой практики является перечислимый тип NumberseqModule. Новое значение этого перечислимого типа, добавленное при обновлении приложения на слое SYS, может пересекаться с добавленным Вами в текущем слое. Чтобы предотвратить использование одного и того же значения при обновлении приложения, следует пропустить несколько значений.

В коде X++ всегда используются названия перечислимых типов и их элементов. При использовании перечислимых типов в отношениях, используются цифровые значения. При изменении кода элемента в перечислимом типе при обновлении приложения, необходимо также изменить значения в отношениях, которые используют этот перечислимый тип. Изменение значения элемента приведет к установке значения свойства **UseEnumValues** в значение «Yes». Изменение значения этого свойства назад в значение «No» приведет к перенумерации всех значений, начиная с нуля, с учетом последовательной нумерации.

```
static void DataDic_BaseEnum(Args _args)
{
    InventTable inventTable;
;

    while select inventTable
        where inventTable.itemType == ItemType::BOM
            || inventTable.itemType == ItemType::Service
    {
        info(inventTable.itemId);
    }
}
```

Для ссылки на элементы перечислимого типа, в коде указывается название перечислимого типа, за которым следует двойное двоеточие. При вводе второго двоеточия система выдаст перечень доступных элементов перечислимого типа во всплывающем окне. В приведенном примере выбираются все номенклатуры с типом Спецификация [BOM] и Услуга [Service]. Вместо указания названия,

можно использовать числовые коды соответствующих элементов, но это приведет к тому, что Ваш код будет сложно читать и поддерживать.

```
while select inventTable  
  where inventTable.itemType >= ItemType::BOM
```

Вместо перечисления всех элементов `itemType`, которые необходимо отобрать, можно использовать лексему «больше или равно» типа номенклатуры BOM, что приведет к выборке того же набора записей (это касается Стандартной поставки Ахарта – в русской локализации был добавлен еще один тип номенклатуры, Основные средства). Однако так использовать перечислимые типы не рекомендуют, так как такой код сложнее поддерживать. При создании нового элемента перечислимого типа `ItemType`, записи с новым типом номенклатуры *также* были бы выбраны, хотя, возможно, изначально это не планировалось.

---

**Примечание:** Первый элемент перечислимого типа обычно имеет значение 0 и вернет значение `false`, если будет анализироваться в условии `if`. Это является дополнительной причиной, по которой поля перечислимого типа не рекомендуется делать обязательными для заполнения, так как значение первого элемента будет считаться при проверке неудовлетворительным.

---

Системные перечислимые типы можно найти в АОТ в узле *System Documentation/Enums*. Все эти перечислимые типы используются в различных панелях свойств. При необходимости изменения свойств объекта из кода X++ следует использовать системные перечислимые типы. К примеру, системный перечислимый тип `TableGroup` содержит перечисления всех возможных значений, которые можно выбрать в свойстве `TableGroup` таблиц.

## 3.6 Функциональные ключи

До появления версии Ахарта 3.0, функциональные ключи использовались для настройки ограничений и полномочий пользователя, для определения списка таблиц, синхронизацию с БД которых необходимо проводить, а также для управления лицензиями. Настройка системы являлась камнем преткновения многих споров и требовала больших трудозатрат и времени. Начиная с версии 3.0, настройками безопасности управляют, используя конфигурационные ключи и ключи контроля доступа, а лицензиями управляют лицензионные ключи. Функциональные ключи больше не могут быть использованы в версии 3.0. Функциональные ключи в версии 3.0 были сохранены только для совместимости с предыдущими версиями. При обновлении приложения с версии 2.5 вы сможете наглядно увидеть набор Ваших функциональных ключей, заменив их конфигурационными ключами и ключами контроля доступа.



### 3.7 Лицензионные коды

При приобретении Ахарта Вам необходимо будет определиться касательно большого числа параметров, таких как количество пользователей, количество серверов, доступа к MorphX и X++. Также необходимо будет выбрать те прикладные модули, которые будут использоваться. Для каждого из параметров системы, а также для каждого выбранного модуля, вы получите свой лицензионный код. Все лицензионные коды будут скомпилированы в один текстовый лицензионный файл. Эти лицензионные коды контролируют ту функциональность системы Ахарта, к которой у Вас будет доступ. В Главном Меню будут доступны только те модули, лицензионные коды на которые вы приобрели. Попытка выполнения объекта без наличия действительного лицензионного кода из АОТ приведет к ошибке.

Партнеры обычно имеют файл лицензий с доступом ко всей функциональности системы Ахарта. При покупке системы клиент приобретет только те модули, которые необходимы ему для бизнеса. При написании модификаций для клиента, настройки системы будут, скорее всего, отличаться. Это может привести к ошибкам в дальнейшем, так как ваши модификации могут неправильно работать у клиента, который не имеет доступа ко всей функциональности. Перед вводом Ваших модификаций в реальную систему следует проверить их на системе с теми же настройками и правами, что и реальное приложение.

Вы можете самостоятельно создавать лицензионные коды и назначать их Вашим модификациям. Однако, для использования ваших лицензионных кодов Вам необходимо будет связаться с Microsoft, так как им необходимо будет сгенерировать файл лицензионных кодов от Вашего имени. Функция создания новых лицензионных кодов используется компаниями, которые создают модули для слоя GLS системы, или же партнерами, которые стремятся продавать свои собственные разработки.

В свойствах конфигурационных ключей присутствует свойство **LicenseCode**, которое служит для назначения лицензионного кода Вашим модификациям. Для настройки пользовательских прав лицензионные коды не нужны, так как это выполняется с помощью ключей контроля доступа. Лицензионные коды используются лишь в случае необходимости в лицензионном коде, как у большинства коммерческих продуктов в сфере программного обеспечения.

### 3.8 Конфигурационные ключи

В Ахарта есть два уровня настройки доступа. Конфигурационные ключи являются наивысшим уровнем, а ключи контроля доступа – вторым уровнем. Если соответствующий конфигурационный ключ отключен, связанные с ним объекты не будут отображаться в меню, формах и отчетах, и никто в системе не будет иметь к ним доступа. Конфигурационные ключи настраиваются в древовидной иерархии, в которой верхним уровнем каждого конфигурационного ключа

является лицензионный код. Форма SysConfiguration отображает иерархию конфигурационных ключей. Включены могут быть только те конфигурационные ключи, для которых был введен правильный лицензионный код, а конфигурационные ключи верхнего уровня могут быть отключены только удалением лицензионного кода. Даже если у вас введены все лицензионные ключи, это не значит, что автоматически будут включены все конфигурационные ключи. Некоторые конфигурационные ключи по умолчанию выключены. Это касается определенной расширенной функциональности, а также функциональности, специфичной для страны.

При внесении изменений в настройки конфигурационных ключей необходимо провести синхронизацию базы данных. С помощью конфигурационных ключей определяется необходимость синхронизации таблицы с базой данных. Следует с осторожностью отключать конфигурационные ключи, так как данные в тех таблицах, для которых выключен конфигурационный ключ, будут утеряны.

Не стоит создавать большое количество конфигурационных ключей для одного модуля. Можно создать один конфигурационный ключ для модуля и несколько – для подмодулей. Обычно конфигурационные ключи назначаются каждой таблице, карте соответствия, представлению и пункту меню во всех модификациях. После назначения конфигурационных ключей все объекты, в свойствах которых проставлен отключенный конфигурационный ключ, не будут отображаться в меню. Более того, при попытке пользователя активировать этот объект из АОТ будет выдаваться на экран сообщение об ошибке, так как у этого пользователя отсутствует доступ к соответствующим объектам БД.

Большинство объектов в АОТ содержат свойство для указания конфигурационного ключа. Вы, скорее всего, замечали, что некоторые расширенные типы данных и перечислимые типы из стандартной поставки имеют связанный с ними конфигурационный ключ. Настройка конфигурационных ключей на уровне всех объектов нерациональна, так как ее было бы очень сложно поддерживать. Однако иногда имеет смысл добавить конфигурационные ключи на определенные расширенные типы данных и перечислимые типы. В таблице SalesTable есть поле ProjId. В расширенном типе данных поля ProjId указан конфигурационный ключ. При наличии правильного лицензионного кода на модули Расчеты с Клиентами и Управление проектами у Вас будет возможность указать Код проекта в поле ProjId для каждого заказа. Если же у Вас куплена только лицензия на Расчеты с Клиентами, то такой возможности у Вас не будет, так как поле ProjId уже будет недоступно.

Указание конфигурационного ключа для элемента перечислимого типа также иногда встречается в стандартном приложении. К примеру, в перечислимом типе FormTextType указаны конфигурационные ключи для всех элементов, добавленных на слое GLS. При отсутствии лицензии на соответствующий модуль нет смысла выбирать эти значения в перечислимом типе. Это свойство следует использовать и в собственных модификациях. Если Вы хотите запретить выбор определенных значений перечислимого типа, то следует создать

конфигурационный ключ и назначить его соответствующим элементам перечислимого типа.

```
static void DataDic_ConfigurationKey(Args _args)
{
    SalesTable salesTable;
    ProjTable projTable;
;

    select firstly salesTable;

    info(salesTable.salesId);
    info(salesTable.custAccount);

    if (isConfigurationKeyEnabled(configurationkeynum(ProjBasic)))
    {
        info(ProjTable::find(salesTable.projId).name);
    }
}
```

Из кода X++ также можно сделать проверку конфигурационного ключа. Глобальный метод `isConfigurationKeyEnabled()` используется для проверки конфигурационного ключа. Нет причины выполнять код, который отключен. В приведенном выше примере, название проекта связанного с заказом будет выведено на экран только в случае, если включен конфигурационный ключ верхнего уровня для модуля Управления проектами. Без такой проверки, если он был отключен, на экран выводилось бы пустое значение. А что, если в коде происходило бы обновление или удаления записи? Упустив такую проверку, можно было бы позволить выполняться коду, который может привести к изменению записей.

---

**Примечание:** Несколько таблиц и полей в названии содержат префикс `DEL_`. Эти объекты больше не используются, и будут находиться в приложении до выхода следующего релиза. Поля и таблицы вместо удаления переименовываются с добавлением вышеуказанного префикса, и данные из предыдущей версии будут утеряны при обновлении версии приложения. Для всех удаляемых объектов проставляется конфигурационный ключ `SysDeletedObjects30`. После завершения обновления приложения этот ключ можно отключить.

---

### 3.9 Ключи контроля доступа

Тогда как с помощью конфигурационных ключей устанавливается доступ на всех пользователей, с помощью ключей контроля доступа можно настроить права, как для группы пользователей, так и для отдельного пользователя. Обычно настройки делаются именно на группы пользователей, так как это намного проще. Ключи контроля доступа, как и конфигурационные ключи, построены в виде иерархии. В стандартном приложении имеется 9 ключей контроля доступа для каждого модуля. Один ключ высшего уровня, связанный с конфигурационным ключом, один ключ для таблиц и еще 7 ключей контроля доступа используются для групп

объектов модуля так, как они сгруппированы в Главном меню. Иерархию ключей контроля доступа можно увидеть из формы SysUserGroupSecurity.

Конфигурационный ключ может быть связан с ключом контроля доступа любого уровня. Однако следует указывать конфигурационный ключ только для ключа контроля доступа наивысшего уровня. При отключении конфигурационного ключа ключ контроля доступа также будет недоступен, а это повлечет за собой отключение всех ключей ниже по иерархии ключей контроля доступа.

Ключи контроля доступа необходимо указывать для всех создаваемых таблиц, карт соответствия, представлений и пунктов меню. Если на каком-то из объектов не будет указан ключ контроля доступа, то не будет и возможности настройки прав доступа к этому объекту, и, соответственно, этот объект будет доступен всем пользователям. Конфигурационные ключи можно и не создавать, если ваши модификации не будут представлены как отдельный модуль, который будет использоваться на более чем одном проекте, так как при установке приложения только в одной компании необходимости отключения конфигурационных ключей нет.

При настройке прав доступа группы пользователей ключи контроля доступа не только включаются или выключаются. Помимо этого можно указать уровень доступа. Доступными значениями уровня доступа являются: Нет доступа, Просмотр, Правка, Создание или Полный доступ. Эти уровни доступа устанавливаются на словаре данных и на пунктах меню. На таблицах, представлениях и картах соответствия для указания уровня доступа используется свойство **MaxAccessMode**. По умолчанию, на таблицах свойство MaxAccessMode установлено в значение “Delete”, что дает пользователям полный доступ к этой таблице. Если таблица используется для хранения проводок, то следует установить свойство MaxAccessMode в значение “Просмотр”, так как проводки не подлежат изменению.

На пунктах меню существует схожее свойство, **NeededAcccesLevel**, которое ведет себя противоположно свойству таблицы, так как в нем необходимо указывать необходимый для его открытия уровень доступа. Значением по умолчанию для свойства NeededAccessLevel является значение “View”. Изменять это значение следует только для пунктов меню с типом Action. Многие из пунктов меню этого типа выполняют обработки, которые обычный пользователь не имеет права выполнять. Или же хотя бы для того, чтобы администратор системы дважды подумал перед тем, как просто открыть к этому пункту меню доступ.

Создание ключей контроля доступа следует отложить на самый последний период в создании Ваших модификаций. Следует, по крайней мере, создать ключи до финальной проверки функциональности. Само собой разумеется, что проверить все комбинации для конфигурационных ключей возможности не будет. Но следует попробовать настроить права доступа для конечного пользователя и поработать с использованием этих прав, для лучшего понимания, как будут вести себя Ваши модификации у пользователей.

### 3.10 Табличные коллекции

При использовании более одной компании часто необходимо использовать некоторые общие данные в таблицах из всех компаний. Примерами таких таблиц в Ахартa являются таблицы, содержащие информацию об адресе, такие как таблица индексов или стран. Для того чтобы использовать данные таблицы из всех компаний, необходимо установить свойство **SaveDataPerCompany** на таблице в значение «No». Это приведет к тому, что ядро системы удалит системное поле `dataAreaId` из этой таблицы, и объединит данные, сделав их доступными во всех компаниях.

Для того, чтобы использовать некоторые данные только для некоторого количества компаний, необходимо использовать табличные коллекции. Плюсом этого подхода является то, что не придется менять никаких свойств на таблицах. Табличная коллекция создается в АОТ, как подузел узла Table Collections, после чего в него перетаскиваются с помощью drag&drop те таблицы, которые содержат общие для компаний данные. Табличную коллекцию можно себе представить просто как шаблон, которым может пользоваться неограниченное количество компаний. Определение компаний, которые будут использовать табличную коллекцию, осуществляется из Главного Меню.

Форма `SysDataAreaVirtual` используется для настройки и создания виртуальных компаний. Виртуальные компании используют табличные коллекции. Виртуальную компанию нельзя выбрать из списка как любую другую компанию. Виртуальная компания – это всего лишь специальный термин, который используется для совместного использования общих данных набором компаний. В указанной форме Вы сможете выбрать, какие именно компании будут принадлежать выбранной виртуальной компании, и какие табличные коллекции они будут использовать. Перед созданием виртуальной компании следует экспортировать данные из таблиц, которые входят в табличную коллекцию, которая будет принадлежать создаваемой виртуальной компании, так как все данные будут удалены из этих таблиц при включении их в виртуальную компанию.

При настройке табличных коллекций для справочников, таких как Группы клиентов, Вы не испытаете трудностей при создании виртуальной компании. Если же Вы будете использовать данные из основных таблиц, таких как номенклатурный справочник, придется копать немного глубже, так как добавление только таблицы `InventTable` в табличную коллекцию будет недостаточным. Необходимо помимо таблицы `InventTable` добавить все таблицы, связанные с ней.

Следует соблюдать осторожности при настройке совместного использования данных. Будет лучше сначала протестировать сделанные настройки на тестовых данных, убедившись, что настройки `SaveDataPerCompany` и табличные коллекции не приводят к возникновению ошибок в объектах, которые используют эти таблицы.

### 3.11 Расширенные возможности использования таблиц

В этом разделе были рассмотрены все базовые инструменты, доступные из словаря данных. Далее будут рассмотрены несколько примеров использования более сложных механизмов работы с таблицами, доступных в MorphX .

#### Использование системных классов

В АОТ в узле *System Documentation/Classes* можно найти несколько классов с префиксом Dict\* в названии. С помощью этих классов можно обратиться к любому объекту словаря данных или его свойствам. При написании универсального кода, когда вы не знаете, какая таблица будет использоваться до момента его выполнения, Вы можете использовать эти системные классы. К примеру, Вам может понадобиться сгенерировать всплывающий список выбора полей из таблицы.

Ниже приведены два примера использования системных классов. В первом примере будет произведен перебор всех полей таблицы. Во втором примере будет произведен перебор всех методов таблицы.

#### Элементы проекта MORPHXIT DataDictionary

- Job, DataDic\_SystemClassesFields
- Job, DataDic\_SystemClassesMethods

```
static void DataDic_SystemClassesFields(Args _args)
{
    SysDictTable      dictTable;
    DictField          dictField;
    Counter            counter;
;

    dictTable = new SysDictTable(tableNum(MyTable));

    for (counter=1; counter<=dictTable.fieldCnt(); counter++)
    {
        dictField = new DictField(dictTable.id(), dictTable.fieldCnt2Id(counter));

        if (dictField.isSystem())
            info(strfmt("System field: %1", dictField.label()));
        else
            info(strfmt("User field: %1", dictField.label()));
    }
}
```

На экран будут выведены все поля таблицы MyTable, как системные, так и обычные, будет выведен текст, информирующий об этом.

Класс SysDictTable используется для инициализации таблицы. С помощью этого класса вы получаете доступ к любому из свойств таблицы и ее подузлам.

Обратите внимание, что SysDictTable – это прикладной класс, наследуемый от системного класса DictTable. В системе присутствует много классов, которые наследуются от системных, и в Ваших модификациях следует использовать именно их, так как по сравнению с базовыми классами в них будут учтены дополнительные проверки.

```
static void DataDic_SystemClassesMethods(Args _args)
{
    SysDictTable      dictTable;
    MethodInfo         methodInfo;
    Counter            counter;
    CustTable          custTable;
;

    select firstly custTable;

    dictTable = new SysDictTable(tableNum(custTable));

    for (counter=1; counter<=dictTable.objectMethodCnt(); counter++)
    {
        methodInfo = dictTable.objectMethodObject(counter);

        if (methodInfo.returnType() == Types::UserType)
        {
            info(strfmt("Method: %1, return value: %2",
                        methodInfo.name(),
                        dictTable.callObject(methodInfo.name(), custTable)));
        }
    }
}
```

В этом примере показано, как получить доступ к узлу методов таблицы и выполнить те из них, которые возвращают какое-либо значение. Этого же результата можно было бы добиться, вызвав метод непосредственно табличной переменной CustTable, но что, если таблица, метод которой нужно вызвать, не известна до вызова, и вместо нее используется системная таблица Common. Производится перебор методов таблицы CustTable. Выбирается первая запись таблицы CustTable для того, чтобы иметь курсор для вызова метода. Если метод возвращает расширенный тип данных или перечислимый тип, которые ядром интерпретируются как пользовательские типы, то этот метод будет выполнен для выбранного курсора и результат его выполнения будет выведен на экран с указанием названия выполненного метода.

## Внешние базы данных

Связь с внешними системами, куда нужно передать какие-то данные, никогда не была простой задачей. Проблема состоит в том, что модификация для интеграции с внешней системой часто делается разными командами. Для того, чтобы

убедиться, что все правильно работает, необходимо выбрать общую платформу для интеграции. Одним из вариантов является использование business connector, который часто называют СОМ, из внешней системы для получения доступа ко всей функциональности Ахapta. Business connector можно рассматривать как клиент Ахapta без интерфейса, для него понадобится постоянно доступное соединение.

Другим вариантом является хранение данных для обмена во внешней базе данных, после чего будет произведен доступ к ним из каждой системы. Из Ахapta, может быть настроено пакетное задание для обработки данных из внешней базы данных. При использовании этого способа Вы получаете простой способ интеграции двух систем, и Вам не придется разбираться с тонкостями интеграции их напрямую.

#### Элементы проекта MORPHXIT DataDictionary

##### ➤ Job, DataDic\_ExternalDatabase

```
static void DataDic_ExternalDatabase(Args _args)
{
    LoginProperty      loginProperty;
    ODBCConnection     odbccConnection;
    Statement           statement;
    ResultSet           resultSet;
    ResultSetMetaData   resultSetMetaData;
    Counter             counter;
;
    loginProperty = new LoginProperty();
    loginProperty.setDatabase("Northwind");
    loginProperty.setDSN("AX30SP4"); // Datasource name for the Axapta database
    loginProperty.setUsername("sa"); // Database login name
    loginProperty.setPassword(""); // Database password

    odbccConnection = new ODBCConnection(loginProperty);

    statement = odbccConnection.createStatement();
    resultSet = statement.executeQuery("select * from Employees");
    resultSet.next();
    resultSetMetaData = resultSet.getMetaData();

    for (counter=1; counter <= resultSetMetaData.getColumnCount(); counter++)
    {
        switch (resultSetMetaData.getColumnType(counter))
        {
            case 0,1 :
                info(resultSet.getString(counter));
                break;
            case 3 :
                info(date2StrUsr(resultSet.getdate(counter)));
                break;
        }
    }
}
```



```
}  
}
```

Этот пример будет работоспособным только при использовании СУБД Microsoft SQL Server. Используется демо база данных "Northwind", которая создается по умолчанию при установке Microsoft SQL Server. Необходимо помимо этого указать источник данных, логин и пароль для доступа к этой базе данных Microsoft SQL Server. Будет прочитана первая запись из таблицы Employees базы данных Northwind и каждое ее поле будет выведено в InfoLog. При необходимости обработки всех записей таблицы нужно это делать в цикле `WHILE resultSet.next()`. Сначала устанавливается соединение с базой данных, после чего производится выборка из таблицы. Перед выводом на экран производится проверка типа поля, так как его нужно знать для выбора метода, который будет использоваться для возвращения значения поля.

### 3.12 Резюме

В этом разделе были рассмотрены возможные причины и способы изменения словаря данных. К этому моменту, Вы должны были освоить, как правильно хранить данные в таблицах для обеспечения оптимальной производительности, как использовать расширенные типы данных для упрощения поддержки модификаций в будущем. Вы обучились использованию отношений и действий при удалении на таблицах для обеспечения безопасности и целостности данных, а также использованию конфигурационных ключей и ключей контроля доступа для настройки прав доступа пользователей к объектам системы.

Было продемонстрировано использование обозревателя таблицы для просмотра и редактирования данных таблиц. Этот инструмент обычно не доступен обычным пользователям. В следующем разделе будут показаны шаги для создания пользовательского интерфейса для работы с данными таблиц.



## 4 Макросы

Макросы используются в качестве констант в приложения Ахарта и широко используются препроцессором. Препроцессор не поддерживает классы, а использует макросы для хранения констант. Это причина того, что макрос на сегодняшний день - часть MorphX. В MorphX макрос не совсем широко используется. Макрос задействован только в нескольких местах таких, как хранение списка полей при использовании диалога. Рекомендуется использовать макрос только для хранения констант. Макрос не предполагается использовать для хранения кода. Повторное использование кода из макроса - менее гибкое решение по сравнению с использованием дополнительного метода.

Макрос может быть создан как в узле Macros AOT, так и локальным макросом в методе или единственной строчкой, определяющей константу. Главное отличие между макросом и методом в том, что макрос не имеет области определения переменной, и код в макросе не проверяется на ошибки перед исполнением из метода. Это основная причина, почему не надо использовать код в макросе, помимо всего код становится трудно читаемым.

Объявлять макрос в коде необходимо после определения переменных. Обычно определение переменных производится в ClassDeclaration класса, формы или отчета. Это делает его доступным в любом методе объекта.

### 4.1 Команды макроса

Для написания макроса используется набор простых команд. Для объявления начала и конца макроса вы устанавливаете начало и конец, как в методе. Можно управлять процессом выполнения макроса с помощью оператора if. Макрос с if оператором используется для проверки, определен ли параметр для макроса или нет. Список команд макроса смотри на **Рисунок 17: Обзор команд макроса.**

Команда	Описание
#define	Используется для определения константы. Смотри макрос HRMConstants.  Пример #define.myConstant100('100')
#endif	Завершает оператор #if.empty или #if.notempty оператор.
#endmacro	Завершает #LOCALMACRO или #GLOBALMACRO.
#globalmacro	Нет отличия, объявлять ли макрос как #localmacro или как #globalmacro. #globalmacro не используется в стандартной конфигурации, лучше использовать #localmacro.
#if.empty	Возвращает истина, если макрос был вызван с пустым параметром.

	Пример <pre>#if.empty(%3)     %3 = %2; #endif</pre>
#if.notempty	Возвращает истина, если макрос был вызван с определенным параметром. Смотри макрос InventDimJoin.  Пример <pre>#if.notempty(%3)     print %3; #endif</pre>
#linenumber	Возвращает текущий номер строки макроса. Может использоваться в целях отладки, но не рекомендуется.
#localmacro	Объявляет начало локального макроса. Смотри classDeclaration для класса SalesReport_Heading.  Пример <pre>#localmacro.MyLocalMacro     print %1; #endifmacro</pre>
#macrolib	Используется для загрузки AOT макроса из кода. Смотри метод класса BOMHierarchy.searchDownBOM().  Пример <pre>#macrolib.MyMacro</pre>
#undef	Отменяет объявление константы с командой #DEFINE. Объявленные константы уже не могут быть использованы, после вызова #undef с объявленной переменной.  Пример <pre>#define.MyConstant(100)  print #MyConstant;  #undef.MyConstant  print #MyConstant; // пусто так , как #MyConstant не определена.</pre>

Рисунок 17: Обзор команд макроса

## 4.2 Определение констант

Константы - это макросы в простейшем виде. Вместо использования текста в вашем коде, жестко рекомендуется определять текст, как константы. Часто вам необходимо определять значение константы, как целочисленное или как текст. Если вы будете устанавливать цвет RGB, то не легко прочитать следующее:

```
myStringColor(255, 255, 255)
```

Вместо этого вам следует рассмотреть определение константы с описательным названием:

```
myStringColor(#RBGColorWhite)
```

Частое использование макроса для определения константы вместо ввода каждый раз значения в коде делает разработку проще. Если позднее вам понадобится изменить значение, то вы просто измените макрос. Правила хорошего тона программирования [best practice] проверяют целочисленные и текстовые значения, использованные в коде, на предмет замены их значениями констант. Практичный подход – это объединение используемых констант вашей модификации и создание макроса в АОТ для их хранения в одном месте. Просмотрите макросы, расположенные в АОТ и вы обнаружите, что некоторые макросы имеют префикс модуля, перед словом Constants.

Для определения макроса используется команда `#define`. Упомянутое выше RGB значение определяется как следующее:

```
#define.RBGColorWhite(255, 255, 255)
```

### 4.3 Создание макроса

При просмотре классов в АОТ вы, наверное, замечали макрос с названием `CurrentList`, используемый во многих местах приложения. Это пример использования самого обыкновенного макроса. Макрос `CurrentList` используется для хранения списка полей диалога.

```
#define.CurrentVersion(2)
#localmacro.CurrentList
    FromDate,
    ToDate,
    Interest,
    CategoryA,
    CategoryB,
    CategoryC,
    Model
#endmacro
```

Это кусок кода из `ClassDeclaration` класса `InventReport_ABC`. Константа `CurrentVersion` используется для хранения номера версии макроса `CurrentList`. Если изменялись поля в макросе `CurrentList`, то константа `CurrentVersion` увеличивается. Это часть интерфейса используемого для хранения значений в диалоге, которая объясняется далее в главе **Классы**.

```
static void Macros_LocalMacro(Args _args)
{
```

```

CustTable custTable;
;
#localmacro.selectCustTable
  #ifnot.empty(%1)
    while select %1
      #ifnot.empty(%3)
        order by %3
      #endif
      {
        info(queryValue(%1.%2));
      }
    #endif
  #if.empty(%1)
    info("No table specified.");
  #endif
#endmacro

#selectCustTable(CustTable, accountNum)
#selectCustTable
}

```

Макрос можно создавать или прямо в коде, или в узле АОТ, а затем объявлять в коде. Последний пример показывает создание макроса #localmacro в коде. Макрос перебирает записи в таблице и выводит поле из неё. Таблицу, которую необходимо вывести, и поле следует определять в параметрах. Выбранные записи из таблицы при вызове макроса выведутся в InfoLog. Так как вы не можете определять переменные в макросе, используется префикс целочисленного значения %1 вместо переменной. Значения параметров последовательно нумеруются и по ссылке в профиле параметров передаются при вызове макроса. Здесь макрос имеет 3 параметра. Порядок сортировки данных является дополнительным и условие #ifnot.empty проверяет наличие третьего параметра. Конечно, должно быть большее число проверок, но это самый простой путь использования макроса. Проверки должны делаться перед вызовом макроса. Отметьте, что вы можете вызвать макрос без ввода круглых скобок после имени макроса.

---

**Примечание:** При изменении макроса в АОТ, вам необходимо перекомпилировать все объекты АОТ, которые используют данный макрос. Изменения в макросе АОТ распознаются объектами, содержащими макрос при компиляции.

---

Если ваш макрос будет вызываться в нескольких местах системы, то имеет смысл создания макроса в АОТ, так как затем вы можете повторно его использовать. Для создания нового макроса в АОТ разверните узел Macro и через контекстное меню выберите *Создать Macro*. Новый пустой узел макроса будет создан. Вам необходимо определить имя нового макроса Macros\_MyMacro, открыв лист свойств. Теперь поместите код #localmacro (последний пример) в ваш новый макрос.

```

static void Macros_MacroLib(Args _args)
{

```

```
CustTable custTable;  
;  
#macrolib.Macros_MyMacro  
  
#selectCustTable(CustTable, accountNum)  
#selectCustTable  
}
```

Использование созданного макроса АОТ производится использованием команды `#macrolib`. Здесь имя макроса АОТ - `Macros_MyMacro`. Отметьте, что макрос может содержать столько `#localmacro`, сколько потребуется. Вы так же можете добавить определение константы и `#localmacro` в тот же самый макрос. Но лучше разделить константы и `#localmacro` на два макроса, делая более наглядным ваш код.

---

**Примечание:** Если имя макроса то же самое, что и имя `#localmacro` в одном АОТ макросе, то на `#localmacro` можно ссылаться без определения макроса в АОТ, а используя `#macrolib`. Это не будет работать должным образом так, как параметры для `#localmacro` не распознаются.

---

## 4.4 Резюме

В этой главе объяснялось использование макросов в MorphX. Вы теперь способны определять константы, используя команды макроса и создавать макросы, как в коде, так и в узле АОТ, а так же использовать макро библиотеку.

Вы должны знать, для чего употребляются макросы и в каких случаях их использовать, но более всего важно, почему писать код в методах лучше, чем в макросах.





## 5 Классы

X++ является объектно-ориентированным языком программирования. Это означает, что код расположен в методах объекта. Используя данный профиль, объект может связываться с другими объектами. Одна из сильных сторон языка – это наследование некоторых объектов. Класс может иметь наследников [subclass], позволяя повторно использовать код родительского класса, так же называемого super классом. Это облегчает разработку модификаций, так как вы можете расширять существующую функциональность, создавая наследников в вашей модификации.

Если вы хотите поближе познакомиться с ООП, то можете найти полезные ресурсы в интернете. Базовые знания по ООП помогут вам при разработке ваших классов. Не обязательно иметь знания в ООП для чтения этой главы, однако они помогут вам для понятия некоторых терминов.

Класс – это коллекция методов. По сравнению с табличными методами, использование классов позволит использовать код повторно. Главное отличие в том, что классы могут наследоваться, а также метод класса может ссылаться на другие методы внутри класса. Поэтому нельзя сказать, что лучше помещать код в классы, чем в табличные методы, так как табличные методы служат для одного, а классы для другого. По умолчанию табличные методы используются для выполнения действий над записями таблицы. Если код должен работать с другими объектами, а не с одной таблицей, то вам следует использовать класс. С другой стороны можно объявить класс и выполнять метод класса из метода таблицы.

Классы – это один из сложных типов данных в MorphX. Базовый тип хранит одно значение переменной, а класс может хранить несколько значений нескольких переменных. Подобно базовым типам, классы используют переменные для хранения значений. На переменные в классе можно ссылаться из других методов класса. Методы класса так же используются для получения значений переменных класса вне класса. Это методы доступа (обычно с префиксом `get`) и называется инкапсуляцией. Рассматривайте класс, как прикладной объект, где методы класса – это инструменты взаимодействия с классом. Класс не имеет предопределенных установленных значений, как таблица, а только класс как объект, проинициализированный в коде, может содержать значения, заключенных в нем, переменных. При инициализации класса, создается экземпляр класса, обычно это делается с использованием метода `new`. Класс объявляется также как табличная переменная, но только табличная переменная может содержать курсор.

### 5.1 Основы класса

В Ахapta классы разделены на классы приложения и системные классы. Вы используете классы приложения для создания своих модификаций. Только классы

приложения могут создаваться и модифицироваться. Системные классы преимущественно используются для непосредственной обработки значений и их код закрыт от редактирования и просмотра. Вы найдете объяснение системных классов в этой главе.

Классы приложения располагаются в АОТ в узле *Classes*.

## Методы

Одна из концепций ООП - это разделение кода на маленькие блоки, где каждый блок осуществляет определенную операцию. Класс – это как раз такой блок. Классы разделены на методы, где каждый метод выполняет определенную задачу. Вам следует учитывать повторное использование кода в создаваемых методах. Вы можете написать код в одном единственном методе класса, но это сделает ваш класс бесполезным для других целей. Некоторые используют за правило помещать в метод определенное количество строчек кода. Это будет хорошим решением, если вы будете держать в голове, что ваш метод используется для определенного расчета и может быть вызван из другого места.

### Компоненты метода

Код в методах состоит из 4 блоков: идентификация, определение переменных, непосредственно строчек кода и возвращаемого значения. Идентификация располагается в самом верху метода и имеет следующий синтаксис:

`< modifiers> <return type> <method name>(parameter profile)`

Модификатор [`modifiers`] – ключевое слово, определяющее поведение метода.

Объяснение этих ключевых слов вы можете найти в разделе **Модификаторы**.

Возвращаемый тип должен быть определен заранее. Это может быть любой из простых типов или сложный тип. Если вы не желаете, что бы метод возвращал что-либо, то используйте модификатор *void*, как возвращаемое значение. *Void* используется в некоторых предопределенных методах на таких объектах, как таблицы и формы, которые не имеют возвращаемого значения.

Имя метода необходимо начинать с прописной буквы. Используйте краткое описание метода в имени. Имена такие, как `calcInventQty()` или `isFormDatasource()` поясняют гораздо больше, чем просто `calcQty()` или `formDatasource()`.

После имени метода следует описание профиля параметров [`parameter profile`].

Параметры определяются подобно переменным. Нормальная практика использовать префикс в параметрах "\_" для избегания путаницы параметров и переменных, используемых в методе. Не обязательно метод должен иметь параметры. Однако вы можете устанавливать параметры по умолчанию при вызове метода. Это делается присвоением переменной параметра определенного значения. Отметьте, что рационально такую переменную параметра ставить на последнее место в профиле параметров.

При вызове метода, ваш метод имеет тот же самый профиль параметров.

Второй блок - это определение переменных. Вы можете поподробнее ознакомиться с определением переменных в главе **Введение в X++**.

Строчки кода добавляются после определения переменных. Если метод имеет возвращаемый тип, то ключевое слово *return* используется для возвращаемого значения. Вызов *return* происходит в конце метода и возвращает значение определенного типа. Вы можете использовать *return* любое количество раз в методе. Однако правилами хорошего программирования (best practice) рекомендуется использовать один раз *return* в методе так, как использование нескольких *returns* делает код трудно читаемым. Используйте создание переменной для хранения возвращаемого значения и вызывайте *return* как последнюю строчку кода.

```
public AmountMst sumCustTrans(CustAccount    _custAccount,
                             TransDate      _transDate = systemdateget())
{
    CustTrans    custTrans;
    AmountMst    sumAmountMst;
;






    if (!_custAccount)
        throw error("Customer account not specified.");

    sumAmountMst = (select sum(amountMst) from custTrans
                    where custTrans.accountNum == _custAccount
                    && custTrans.transDate   >= _transDate).amountMst;

    return sumAmountMst;
}
```

Этот метод подсчитывает сумму покупок клиента и возвращает её, используя переменную *sumAmountMst*. Метод имеет два параметра и один по умолчанию. Если первый параметр *customer account* не определен, то появится сообщение об ошибке, а если метод будет вызван только с одним параметром *customer account*, то второй параметр дата покупки будет определен текущей датой.

### Иконки

Иконки метода определяются модификатором. Предопределенные методы имеют иконку со стрелочкой , методы созданные в текущем классе имеют иконку , защищенные [protected] методы имеют иконку с ключиком , частные [private] методы класса представлены иконкой с замочной скважиной  и статические методы имеют иконку с крестиком .

Иконки упрощают обзор методов таблицы или класса. Посмотрите на класс *CustBillExchangeClose*. Легко определить тип метода. Отметьте, что предопределенные и другие динамические методы сортируются по алфавиту, а

статические методы представлены списком после них. Это правильная причина сортировки так, как статические методы имеют отличное от не статических методов использование.

Отметьте, что иконка имеет "светофор" в правом углу иконки. Цвета красный или желтый показывают об ошибке или предупреждении в методе. Зеленый цвет показывает, что метод компилируется без ошибок.

## Компоненты класса

Класс приложения имеет 3 узла: *ClassDeclaration*, *new* и *finalize*. Смотри **Рисунок 18: Узлы класса**. В *ClassDeclaration* определяются глобальные переменные класса. Переменные в *ClassDeclaration* только определяются, но не иницируются значениями. Только локально определенные переменные метода могут быть иницированы значениями по умолчанию.

Макрос, если используется для всего класса, так же определяется в *ClassDeclaration*.

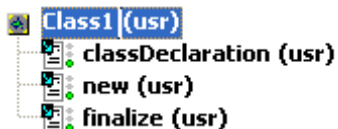


Рисунок 18: Узлы класса

При объявлении класса, вы объявляете класс как объект. Перед тем как ссылаться на объект класса, вам следует инициировать объект класса. Компилятор выдаст ошибку, если вы будете ссылаться на объект класса, не иницировав его прежде. Ключевое слово *new* используется для инициирования объекта класса. Синтаксис следующий: *new <Class name>()*. Объект класс и иницируемый класс должны быть одного типа. Это означает, что объект класс и иницируемый класс должны быть равны или наследник равен родительскому классу.

```
MyClass      myClassObject;
;
myClassObject = new MyClass();
```

Инициализация класса производится методом *new()*. Как и другие методы, метод *new()* может иметь параметры. Переменные, объявленные в *ClassDeclaration* часто иницируются со значениями параметров, которые берутся из метода *new()*. Вызвав *new()*, вы вызываете конструктор класса. Конструктор создает экземпляра класса в оперативной памяти. Некоторые языки программирования имеют такие же названия методов в классе. Этот метод в них тоже может использоваться, как конструктор. В *MorphX* *new()* используется только как конструктор. Обычно в *MorphX* используется метод *construct()*, если вам необходимо инициировать ваш класс, используя его подклассы, наследники. Нормально, если

метод `construct()` имеет один параметр, который используется в `construct()` для определения какой наследник инициировать, используя `new()`. Класс `NumberSeqReference` который является частью системы, формирующей номерные серии в Ахapta, использует метод `construct()`.

Метод `finalize()`, используется для удаления класса из памяти. После вызова `finalize()` вы не сможете ссылаться на объект класса. Это метод не вызывается автоматически, как сборщик мусора для удаления не используемых объектов из памяти. Не имеет смысла вызывать метод `finalize()`, если объект более не используется. Если вы в цикле иницируете каждый раз класс, вам не следует вызывать `finalize()`, как сборщик мусора при превышении определенного количества не используемых объектов.

Имеет смысл использовать `finalize()`, если ваш объект больше не предполагается использовать, а так же для принудительного удаления объекта. Пример использования можно найти в классах, использующих COM, для разрыва сессии.

---

### Пример 1: Создание класса

#### Элементы MORPHXIT проекта Classes

##### ➤ Класс, MyClass

В этом примере создадим класс с одним методом. Это простой пример покажет вам, как создавать и использовать класс.

1. Перейдите к узлу *Classes*, через контекстное меню правой кнопки мыши выберите *Создать Class*. Вновь созданный класс имеет имя "Class1". Откройте класс двойным кликом мыши.
2. Проверьте левое окно, для того чтобы убедиться, что вы находитесь в *ClassDeclaration*. Переименуйте класс, изменив "Class1" на "MyClass".
3. В *ClassDeclaration* объявите переменную с расширенным типом *ItemId*. *ClassDeclaration* должен выглядеть следующим образом:

```
class MyClass
{
    ItemId itemId;
}
```


4. Теперь добавьте новый метод к классу, нажав `ctrl+n`. Новый метод получит имя "Method1" и автоматически откроется в редакторе. Измените имя метода на "parmItemId".
5. Добавьте расширенный тип *ItemId* как параметр в новом методе `parmItemId()`. Параметр будет инициироваться глобальной переменной класса `itemId`. Установите переменную параметра равной глобальной переменной класса `itemId`. Метод должен возвращать `itemId`.

```

itemId parmItemId(ItemId _itemId = itemId)
{
;
    itemId = _itemId;

    return itemId;
}

```

6. Нажмите на иконке сохранить  в панели инструментов редактора для сохранения всех изменений в классе.
- 

Так как на переменную класса невозможно ссылаться вне класса, то вы можете создать метод для установки или получения значений переменных в классе. Такие методы часто используют с префиксом `parm*`. Класс `MyClass` содержит один метод с именем `parmItemId()`, который возвращает значение глобальной переменной класса `ItemId`, при вызове метода без параметра. Так же параметр в методе `parmItemId()` может использоваться для установки значения `ItemId` в классе. Такие методы особенно часто используются в диалогах для передачи значения поля диалога вызывающему объекту. Вариант установки значения переменной класса предоставляет возможность использования данного класса без интерфейса диалога.

```

static void Classes_TestMyClass(Args _args)
{
    MyClass    myClass;
;
    myClass = new MyClass();
    myClass.parmItemId("Item100");

    info(myClass.parmItemId());
}

```

Простейший путь проверки класса – это создание задания [job]. Создайте задание, которое будет тестировать `MyClass`. Задание иницирует `MyClass` и запускает метод `parmItemId()` с параметром. Метод `parmItemId()` вызывается еще раз и уже без параметра и выводит предыдущее значение в `Infolog`.

```

void methodWithFunction()
{
    void methodWithFunction(CustAccount _custAccount)
    {
        ;
        info(_custAccount);
    }

    methodWithFunction("4000");
}

```

Любую функцию можно написать внутри метода. Функция может быть использована методом, но на нее нельзя ссылаться вне метода. Синтаксис функции в методе похож на один из методов. Здесь функция объявлена с тем же самым именем, что и метод. Это позволительно, если профиль параметров не равен профилю параметров метода. Функция вызывается без определения имени объекта. Отметьте, что нет ни одной проверки на правильное число входящих параметров в функцию. Вам следует рассматривать функцию в методах, как дополнительную возможность X++. Это не рекомендуемая возможность, так как код не подлежит повторному использованию. Вместо этого вам следует рассмотреть создание метода класса.

## Модификаторы

Предпочтительно определять модификатор для класса или метода. Модификаторы особенно полезны, если вы собираетесь разделить выполнение метода или всего класса (например, на клиенте или сервере), выполнение зависит от наследования класса.

### Модификаторы доступа

Вы можете добавить модификатор доступа к классу или методу для отдельного использования. И динамические и статические методы могут использовать модификаторы доступа, и разделение может быть установлено на разных уровнях. Если вы желаете, что бы метод использовался только внутри класса или только для наследников этого класса. По умолчанию используется открытый уровень [public], который дает доступ к классу и всем его методам. Модификаторы доступа показаны на **Рисунке 19: Обзор модификаторов доступа**. MorphX поддерживает модификаторы доступа только для методов и классов. Переменные, объявленные в ClassDeclaration, будут доступны в наследниках [subclasses].

Модификатор	Описание
Public	Устанавливается по умолчанию для класса или метода. Public class может наследоваться и методы класса могут быть переопределены в классах наследниках и могут вызываться вне класса.
Protected	Только методы могут иметь этот модификатор. Protected method может переопределяться в наследниках, но использоваться может только внутри классов наследников.
Private	И классы и методы могут быть установлены, как private. Однако это действует только на методах. Private method может использоваться только внутри текущего класса.

**Рисунок 19: Обзор модификаторов класса**

Модификаторы доступа в MorphX часто забываются, так как по умолчанию всегда используется `public`. Это позволяет любой части класса быть использованной отовсюду. Не использование разделения доступа для класса, может быть в том, что класс не предполагается наследовать. Вам следует рассматривать модификаторы доступа для понимания работы вашего класса. Класс часто имеет несколько методов подсчитывающих некоторые значения внутри класса. Такие методы следует отделять от перекрытия в классах наследниках.

---

### Пример 2: Модификаторы доступа

#### Элементы MORPHXIT проекта Classes

- Класс, `MyClass_AccessModifiers`
- Класс, `MyClass_AccessModierers_sub`

Мы создадим Родительский класс [super class] и класс наследник [sub class], чтобы показать использование модификаторов доступа.

1. Создадим новый класс и переименуем в “`MyClass_AccessModifiers`”.
2. Добавим метод, с названием `publicMethod()` в класс. Установим модификатор доступа в `public`. Добавим строчку, выводящую ключевое слово модификатора доступа в `Infolog`.

```
public void publicMethod()
{
;
    info("public");
}
```

3. Повторим шаг 3, добавляя похожие методы, для модификаторов `protected` и `private`. Не забудьте установить модификатор доступа для каждого метода.
4. Сохраните класс `MyClass_AccessModifiers`.
5. Создайте класс наследник с именем `MyClass_AccessModifiers_sub`. `ClassDeclaration` для класса наследника должен выглядеть следующим образом:

```
class MyClass_AccessModifiers_sub extends MyClass_AccessModifiers
{
}
```

6. Через контекстное меню на узле класса `MyClass_AccessModifiers_sub` выберите *Переписать Method*. Выберите метод `protectedMethod()`. Этим вы создадите новый метод в классе наследнике.



## 7. Сохраните класс MyClass\_Modifiers\_sub.

---

Класс родитель содержит 3 метода, но при перекрытии методов в классе наследнике, только методы `public` и `protected` будут доступны. MorphX определяет доступность методов по модификаторам доступа, и показывает список методов, которые можно перекрыть. Даже если созданные методы по умолчанию, определяются как `public`, имеет смысл использовать модификаторы. Методы с определенными модификаторами не могут быть доступны в классах наследниках. Отметьте, что перекрытый метод содержит вызов `super()` для вызова родительского метода. `Super()` вызывает исполнение кода, написанного в классе родителе.

---

**Примечание:** Инструменты Visual MorphXplorer и Application Hierarchy Tree, вызываемые из меню add-ins, помогают просматривать иерархию класса. Не забывайте обновлять перекрестные ссылки.

---

При создании класса наследника хорошей практикой служит использование префиксов в имени класса наследника после имени родительского класса "\_". Если класс родитель `SalesFromLetter`, то имя класса наследника может быть `SalesFormLetter_Confirm`.

```
static void Classes_Modifieres(Args _args)
{
    MyClass_AccessModifiers_sub modifiers_sub;

;
    modifiers_sub = new MyClass_AccessModifiers ();
    modifiers_sub.publicMethod();
}
```

Это задание тестирует класс `MyClass_AccessModifiers_sub`. Только метод с модификатором `public` будет выполнен, так как методы с модификаторами `private` и `protected` не могут выполняться вне иерархии класса. При установке точки после имени класса все методы родительского класса и классов наследников будут показаны списком. Компилятор, однако, может сообщить об ошибке, если попытаться использовать метод с не надлежащим уровнем доступа.

---

**Примечание:** Перед версией 3.0 модификаторы доступа были недоступны компилятору. Модификаторы доступа были определены, но не имели эффекта.

---

Использование модификатора `private` для класса не будет иметь эффект. Если вы желаете изменить доступ для класса, то вам следует объявить `new()` как `private`. Это предотвратит объявление класса обычным путем, через использование `new()`. Вместо этого вы можете создать метод `construct()` для контроля объявления класса. Метод `construct()` всегда создается, как статический метод, и будет доступен в зависимости от модификатора доступа для метода `construct()`.

Модификатор доступа может так же использоваться в формах и отчетах для предотвращения вызова методов, вызванных вне объекта. Действительно, вы можете использовать различные модификаторы в формах и отчетах, однако, это в первую очередь имеет смысл для использования модификаторов доступа.

### Модификатор Static

По умолчанию создаваемые методы – динамические. Это означает, что вам следует объявить объект класса перед возможностью использования его методов. Если добавляется ключевое слово *static* как модификатор в метод, то вы получаете доступ к методу без объявления объекта класса.

```
static EmplTable find(EmplId _emplId,
                    boolean _forUpdate = false)
{
    EmplTable emplTable;

    if (_emplId)
    {
        emplTable.selectForUpdate(_forUpdate);

        select firstonly emplTable
            index hint EmplIdx
            where emplTable.emplId == _emplId;
    }

    return emplTable;
}
```

Приведенный выше фрагмент кода показывает метод `find()` из `EmplTable`. Метод возвращает запись `EmplTable`, основанную на коде сотрудника, переданном в параметре. Табличные методы такие, как `find()` и `exist()` всегда создаются как статические. При поиске записи в таблице не имеет смысла объявлять табличную переменную так, как метод вернет её (`find` возвращает табличную переменную). Следующий фрагмент кода показывает, как вызывается статический метод `find()` на `EmplTable`. Отметьте, что в синтаксисе статического метода используется двойное двоеточие. Аналогичный синтаксис используется и для перечислимого типа данных.

```
static void Class_CallStaticMethod(Args _args)
{
    ;
    info(EmplTable::find("АМО").name);
}
```

Модификатор `static` часто используется в методах классов, к которым необходим быстрый доступ. Класс `WinAPI` – хороший пример использования статических методов. Класс `WinAPI` содержит список статических методов таких, как доступ к

файлам системы для выполнения операций. Часто используется WinAPI метод проверка наличия файла.

Методы, созданные для цели объявления объекта класса имеют модификатор `static`. Самые распространенные – это методы `main()` и `construct()`. Отметьте, что `main()` используется для вызова класса через пункт меню или AOT.

### Модификатор Final

Установка для класса модификатора `final` защищает его от перекрытия. Причина может быть в том, что вы хотите использовать только сам класс без наследования. Возможно, класс может предназначаться для использования его другим классом или класс может использоваться для передачи данных в другой класс. Класс `InventOnhand` – пример этого.

Метод с модификатором `final` не может быть перекрыт в классе наследнике. Только динамические методы могут иметь модификатор `final` так, как статические методы не могут быть перекрыты в наследнике по определению. Определяя метод как `final`, вы предотвращаете наследование. У такого метода будет уровень доступа как `public`. Однако вы можете использовать `final` вместе с одним из модификатором уровня доступа, например:

```
final protected void protectedMethod()
{
;
    info("protected");
}
```

### Модификаторы Display и Edit

MorphX имеет два специальных модификатора: `display` и `edit`, которые часто используются в пользовательском интерфейсе. Они используются для полей форм и отчетов, которые не связаны с источником данных. Модификаторы `display` и `edit` могут использоваться с модификаторами `static` и `final`. Вы найдете детальное объяснение по модификаторам `display` и `edit` в главе **Формы**.

### Модификатор Abstract

Класс с модификатором `abstract` или `abstract` метод действуют противоположно по отношению к действию модификатора `final`. Использование `abstract` класса – это планирование наследования и последующее создание класса наследника для использования класса, так как отдельно метод в классе не может быть объявлен как `abstract`. Это часто используется в стандартной конфигурации для родительских (`super`) классов для запрета ошибочного объявления объекта родительского класса. Класс `SalesFormLetter` создающий документы такие, как подтверждение и счет на оплату используют эту практику. Класс `SalesFormLetter`

определен как `abstract` для предотвращения объявления объекта класса через `new()` и имеет метод `construct()`, который следует использовать.

```
abstract CustAccount myAbstractMethod()
{
    // Код в этом методе никогда не выполнится

    #if.never
        select firstly custTable
        where custTable;
    #endif
}
```

Методы могут быть объявлены, как `abstract`, только если класс объявлен, как `abstract`. Метод `abstract` должен быть переопределен в наследнике, поэтому `abstract` не может иметь кода. Методы `abstract` содержат только профиль параметров. Однако вы можете использовать вызов макроса `#if.never` для добавления «кода» в абстрактный метод и в дальнейшем использовать «код», как некий шаблон. Это поможет в будущем понять, почему этот метод должен быть переопределен в классе наследнике. Возможно, вам интересно, почему бы просто не добавить комментарии в `abstract` метод? Ответ в том, что код внутри макроса имеет цвет такой же, как и другой код в редакторе, делая легким отличие комментариев и кода. Отметьте, что не сделано ни одной проверки в коде макроса `#if.never`, поэтому можно писать туда что угодно. Рекомендуется дополнительно определять методы абстрактного класса, как `abstract`. Поэтому вы также можете добавлять переменные в `ClassDeclaration` абстрактного класса.

Модификатор доступа `protected` может использоваться в методах `abstract`. Абстрактные методы не могут быть `static` так, как `static` методы существуют непосредственно в классах.

### Интерфейс класса

В `MorphX` имеет место только единственное наследование, это означает, что допускается только один класс родитель. Это не так плохо так, как в языках, поддерживающих множественное наследование только с помощью ухищрений можно получить обзор иерархии классов.

Существует еще одна возможность – это создание интерфейса `[interface]` класса и использование его в классе наследнике. Подобно абстрактным классам и методам `interface` класс не может быть объявлен и методы `interface` класса не могут содержать код. Так в чем же отличие между `abstract` классом и `interface`? Абстрактный класс используется для получения инструкций по содержимому в классах наследниках. А интерфейс не является частью иерархии. Возможно, сделать более чем один интерфейс для класса. Интерфейсы используются для решения простых задач как класс интерфейс `SysPackable`, который контролирует, что бы были созданы методы, для хранения последних значений диалога.

```
interface MyClass_Interface
{
}
```

Интерфейс в действительности не реальный класс. При создании интерфейса вы заменяете ключевое слово *class* на *interface*. Интерфейс может наследоваться другим интерфейсом, но класс не может наследовать interface, класс реализовывает интерфейс.

```
public abstract class Runbase extends Object implements sysSaveable, sysRunnable
{
}
```

При написании класса, реализующего interface, используется ключевое слово *implements*. Здесь приводится заголовок из класса Runbase. Отметьте, что в заголовке слово *implements* ставится после наследуемого класса.

---

**Примечание:** Вызовом глобального метода `pickInterface(true)` из задания [job] все interface классы будут показаны списком. Класс Global имеет несколько методов с префиксом `pick*`, которые могут использоваться для просмотра узлов AOT.

---

Любой модификатор доступа может использоваться в интерфейсе. Помните, что интерфейс не родительский [super] класс, а некий шаблон для вашего класса. В классе, реализующем интерфейс, обязательно должны быть переопределены все методы интерфейса.

## Передача значений

Профиль параметров в методе может содержать любое количество параметров. И базовые типы данных и сложные типы могут предаваться параметрами. Это главная цель параметров - передача значений переменных в метод. Это делает ваш метод кирпичиком, который легко прикрепить к другому кирпичику.

### Передача значений

В MorphX переменные передаются по значению. Это означает, что переменные, переданные в метод, не изменятся, модифицированием переменной параметра внутри метода. Передача временной таблицы в качестве параметра отличается от правила вызова временной таблицы по ссылке. Если передается временная таблица в качестве параметра в метод и параметр изменяется в методе, то значение параметра тоже меняется.

Не имеет значения, какой тип переменной в параметре используется, только рекомендуется создавать локальную переменную для переменной параметра вместо модифицирования самого параметра так, как это делает ваш код легким в понимании.

---

**Пример 3: Передача значений**
Элементы MORPHXIT проекта Classes

## ➤ Класс, MyClass\_PassingValues

Класс будет содержать два созданных метода. Цель – показать, как MorphX работает при передаче переменных в метод. Параметр в методе callByValue() меняет свое значение внутри метода. Этот пример не рассматривается, как хороший стиль программирования.

1. Создайте новый класс MyClass\_ParameterValues.
2. Добавьте метод с названием callByValue() с одним параметром расширенного типа Counter. Увеличиваем значение Counter на 10 и затем возвращаем результат.

```
Counter callByValue(Counter _counter)
{
    _counter += 10;

    return _counter;
}
```

3. Добавьте метод с названием callByReference() с временной таблицей TmpAccountSum в качестве параметра. Метод должен выводить последнюю запись из временной таблицы в Infolog.

```
void callByReference(TmpAccountSum _tmpAccountSum)
{
    TmpAccountSum tmpAccountSum;
    ;
    tmpAccountSum = _tmpAccountSum;

    select firstonly tmpAccountSum order by accountNum desc;
    {
        info(tmpAccountSum.accountNum);
    }
}
```

4. Сохраните класс.
- 

Для тестирования класса MyClass\_PassingValues создайте следующее задание:

```
Static void Classes_CallByValue(Args _args)
{
    MyClass_ParameterValues    passingValues    = new MyClass_ParameterValues ();
    Counter                    counter          = 100;
    ;
}
```

```

info("Исходное значение параметра "+strFmt("%1", counter));
info("Значение, получаемое из метода "+strFmt("%1", passingValues.callByValue(counter)));
info("Значение параметра не изменилось"+strFmt("%1", counter));
}

```

Задание иницирует переменную счетчик и выводит значение в Infolog перед вызовом callByValue(). Далее задание возвращает значение из метода callByValue() и выводит в Infolog, заключительно задание выводит значение переменной счетчика, показывая тем самым, что метод callByValue() не изменил значение параметра переменной счетчика.

```

static void Classes_CallByReference(Args _args)
{
    CustTable          custTable;
    TmpAccountSum      tmpAccountSum;
    MyClass_ParameterValues passingValues = new MyClass_ParameterValues ();
    Counter            counter;
;

    while select custTable
    {
        counter++;

        if (counter > 5)
            break;

        tmpAccountSum.accountNum = custTable.accountNum;
        tmpAccountSum.insert();
    }

    select tmpAccountSum;

    info("Исходное значение"+ tmpAccountSum.accountNum);
    passingValues.callByReference(tmpAccountSum);
    info("Значение после обработки в методе"+ tmpAccountSum.accountNum);
}

```

В следующем задании производится тестирование метода callByReference(). Сначала во временную таблицу необходимо вставить данные. Первые 5 записей из CustTable выбираются в цикле и вставляются в таблицу. Первая запись из временной таблицы выводится в Infolog. CallByReference() вызывается с параметром временной таблицы, и выводит последнюю запись из временной таблицы. Отметьте, что параметр временной таблицы после вызова метода CallByReference() изменился на значение переменной этой временной таблицы в методе CallByReference().

Вызов по ссылке может вас запутать и не совсем подходит вам. Но чтобы не менялось значение переменной временной таблицы в задании, измените, код в методе callByReference(), как показано ниже.

```

void callByReference(TmpAccountSum _tmpAccountSum)

```

```

{
    TmpAccountSum tmpAccountSum;
;
    tmpAccountSum.setTmpData(_tmpAccountSum);

    select firstonly tmpAccountSum order by accountNum desc;
    {
        info(tmpAccountSum.accountNum);
    }
}

```

Метод setTmpData() инициализирует локальную переменную TmpAccountSum а, устанавливая TmpAccountSum равной переменной параметра \_tmpAccountSum, будет аналогичен вызову по ссылке. Отметьте, что только временные таблицы передаются по ссылке. Обычная таблица, как CustTable, будет передаваться по значению.

### Рекурсивные вызовы

Рекурсивный метод [recursive method] - это метод, вызывающий сам себя. Проверяйте при вызове рекурсивного метода, чтобы в вашем коде не было бесконечного цикла. Рекурсивный метод превосходит для повторного использования кода и делает ваш код простым.

Добавьте следующий метод в класс MyClass\_ParameterValues:

```

void recursiveCall(ProjId _projId = "")
{
    ProjTable projTable;
;

    while select projTable
        where projTable.parentId == _projId
    {
        setprefix(projTable.parentId);
        info(projTable.projId);

        this.recursiveCall(projTable.projId);
    }
}

```

Таблица ProjTable, главная таблица проектов обрабатывается в цикле в методе recursiveCall(). Проекты в Ахарты строятся иерархически, поэтому метод сортирует записи в дереве иерархии. Вам бы потребовалось написать много кода, по меньшей мере два метода, если бы не было рекурсивных вызовов. Для каждого цикла в методе recursiveCall() выводится строчка в Infolog. Метод setprefix() используется для показа уровня проекта в Infolog. Создайте ещё одно задание для тестирования метода:

```

static void Classes_RecursiveCall(Args _args)

```



```
{
    MyClass_ParameterValues    passingValues    = new MyClass_ParameterValues ();
;

    passingValues.recursiveCall();
}
```

### Возврат значений

Метод часто возвращает одно значение или одну переменную. Когда вызывается `return`, метод заканчивает работу. Иногда у вас возникает ситуация, когда необходимо возвращать более, чем одну переменную. В этом случае вам следует пересмотреть ваше решение, так как вы слишком много обрабатываете в методе.

```
Container returnContainer()
{
    CustAccount custAccount;
    custName    custName;

    return [custAccount, custName];
}
```

Возвращаемый тип может быть базовым или расширенным, конечно, можно использовать и контейнер `[container]`. Вы можете явно не объявлять переменную контейнер в методе, если возвращать значения в квадратных скобках `[]`, как показано в примере выше.

MorphX имеет набор базовых [фундаментальных] классов, которые решают эту проблему. Базовый класс может содержать необходимое число переменных. Вы можете подробнее ознакомиться с базовыми классами в разделе **Базовые классы**.

## 5.2 AOS

Ахарта может быть сконфигурирована для работы с 2 уровневой или 3 уровневой клиентом. В 2 уровневой конфигурации система состоит из клиента и сервера, а 3 уровневая ещё имеет сервер приложений [Application Object Server], так же называемый AOS. Без сервера AOS выполнение логики приложения происходит на клиенте, которая служит результатом большого объема пакетов данных, перемещаемых между сервером и клиентом. AOS существенно уменьшает трафик между базой данных и клиентом, так как сам AOS преимущественно связывается с базой данных, кэширует приложение и уже связь с клиентом осуществляется только через сервер приложения.

Вы можете настроить любое количество AOS серверов для распределения рабочей нагрузки оборудования. За дальнейшее информацией о AOS смотрите руководства в стандартной поставке. Объяснение работы AOS не рассматривается в этой книге. Наша цель – это оптимизация вашего кода, используя AOS.

---

**Примечание:** Оптимизация вашего кода при использовании AOS является дополнительной возможностью. Вам не следует оптимизировать весь код на использование AOS, так как потратите много времени на оптимизацию. Но если у вас тяжелые модификации или ограниченная пропускная способность канала связи, то вы получите лучшие результаты, настраивая AOS.

---

## Установка места выполнения

При выполнении табличного метода или класса вы можете определить, где у вас будет выполняться ваш код на AOS или на клиенте. По умолчанию табличные методы обращаются к базе данных для вставки, обновления или удаления. Эти методы всегда выполняются на сервере. Формы всегда выполняются на клиенте. По умолчанию объект выполняется на том уровне, откуда он вызван. Решение по реализации выполнения объекта по умолчанию, является решением оптимизации.

Ваш код будет работать и в 2 уровневой и в 3 уровневой конфигурации, так как настройка выполнения объекта может быть осуществлена только для 3 уровневой системы. Это так же означает, что выполнение настроек оптимизации для AOS будет только в 3 уровневой конфигурации. Изменение выполнения формы на сервере будет допустима в 2 уровневой конфигурации, но при работе в 3 уровневой конфигурации форма никогда не будет показана, так как выполнение формы было определено на сервере.

Определение места выполнения определяется свойством **RunOn** объекта и использованием модификатора *client* или *server*. Пункты меню и классы имеют свойство RunOn. По умолчанию выполнение пунктов меню происходит на клиенте. Классы по умолчанию выполняются из того места, откуда вызваны. Свойство RunOn может быть установлено на клиенте, сервере или месте вызова [called from]. Устанавливая свойство RunOn в called from означает, что объект выполнится на уровне вызывающего объекта. Ключевое слово *client* или *server* используется для определения места выполнения метода. Установка места выполнения метода происходит добавлением модификаторов *client* или *server*. Отметьте, что хорошим тоном программирования считается установка модификаторов самым первым среди других модификаторов.

```
server AmountMST balancePerDate(TransDate _transactionDate = systemdateGet())
{
    return this.CustVendTable::balancePerDate(_transactionDate);
}
```

Свойство RunOn в листе свойств класса используется для определения места выполнения класса. Все динамические методы класса выполняются в том месте, которое определено в свойстве RunOn класса. Только статические методы класса могут иметь свой отдельный AOS модификатор. Метод, показанный выше, установлен на выполнение на AOS.

Таблицы отличаются несильно, но и динамические и статические методы таблицы могут иметь свои AOS модификаторы.

Если вы установили свойство RunOn для пункта меню, вызывающего класс, то класс по умолчанию будет выполняться на том же самом уровне, при установке свойства класса RunOn в called from. Установка свойства для пункта меню, однако, может быть отклонена установкой свойства класса. Место исполнения определяется так же и объявлением класса, где вызывается new(). Это так же означает, что свойство RunOn родительского [super] класса будет так же определять место исполнения класса потомка [subclasses].

## Объекты для оптимизации

Итак, какие объекты следует оптимизировать для выполнения на AOS и когда это следует делать? Правило такое, что табличные методы и классы с большим количеством вызовов базы данных – кандидаты на выполнение на сервере приложения AOS. Отчеты, использующие структуру классов runbase лучше так же запускать на сервере. Это может звучать неправильно, что отчеты выполняются на сервере, в то время как формы не могут выполняться на сервере. Структура классов runbase распределяет вызовы между клиентом и AOS так, что пользователь просматривает диалог отчета, а отчет выполняется на сервере приложения AOS.

Инструменты MorphX такие, как системный монитор [System Monitoring] и профайлер кода [Code Profile] могут использоваться для просмотра вызовов между клиентом и сервером. Вам следует ознакомиться с ними перед тем, как оптимизировать ваш код для исполнения на AOS.

## 5.3 Структура класса Runbase

Задание пакетной обработки [batch job] – это задача, выполняемая в определенное время или повторяющаяся через определенные интервалы времени. Сервер пакетной обработки [batch server] выполняет задания пакетной обработки [batch jobs] и обычно исполняется на сервере. Клиент Ахapta запускает сервер пакетной обработки, но также сервер пакетной обработки альтернативно может запускаться, как Microsoft Windows Service. Вы можете найти подробное описание конфигурирования batch servers и batch jobs в руководстве пользователя в стандартной поставке.

Тяжелые задачи выполняемые классами или комплексными отчетами работают со множеством записей и обычно распределяются по batch jobs. Это может быть ежедневная задача такая, как выгрузка заказов во внешнюю систему или периодическая распечатка баланса покупателей.

Batch jobs не взаимодействует с пользователем. При установке выполнения класса в batch job значения введенные пользователем в диалоге пакетной обработки хранятся особо так, как batch server избегает любых диалогов.

## Использование структуры класса Runbase

Структура класса runbase имеет две основных функции. Это вывод диалога, представляемого пользователю для заполнения и возможность выполнения процесса пакетной обработки. Классы с префиксом RunBase\* используются в структуре runbase. Некоторые из этих классов вызываются только структурой. В ежедневном использовании вам необходимо знать только некоторые из них. Смотрите **Рисунок 20: Часто используемые классы структуры runbase**.

Название класса	Описание
RunBase	Класс RunBase используется для задач, которые можно не использовать в пакетной обработке.
RunBaseBatch	RunBaseBatch используется, что бы дать возможность пользователю назначать расписание batch job для выполнения задачи.
RunBaseReport	Используется для тяжелых отчетов и поэтому является наследником RunBaseBatch.

**Рисунок 20: Часто используемые классы структуры runbase**

Часть структуры класса runbase используется для отчетов и объясняется более детально в главе **Отчеты**.

---

### Пример 4: Создание класса пакетной обработки

#### Элементы MORPHXIT проекта Classes

- Class, MyClass\_RunBaseBatch
- Menu item action, MyClass\_RunBaseBatch

В этом примере будет создан класс, использующий структуру runbase. Класс будет иметь опцию выполнения в batch job. Наша цель в том чтобы разобраться, как конструировать класс, использующий структуру runbase.

Класс выведет всех покупателей, чьи проводки совершены после определенной даты. Последняя введенная пользователем дата для поиска проводок будет сохранена.

Создадим новый класс и назовем его “MyClass\_RunBaseBatch”.

```
class MyClass_RunBaseBatch extends RunBaseBatch
{
    FromDate    fromDate;
    DialogField dialogFromDate;

    #define.CurrentVersion(1)

    #localmacro.CurrentList
        fromDate
    #endmacro
}
```

Новый класс должен быть наследником класса RunBaseBatch для осуществления пакетной обработки. Переменная с расширенным типом FromDate создается для хранения значения последней введенной даты. Переменная dialogFromField относится к классу DialogField, который используется для добавления полей в диалоге. Константа CurrentVersion содержит запись последнего изменения диалога. CurrentList – макрос, содержит переменные для хранения последних значения полей диалога.

```
public container pack()
{
    return [#CurrentVersion, #CurrentList];
}
```

Метод pack() должен быть перекрыт. Метод возвращает два макроса, определенные в ClassDeclaration.

```
public boolean unpack(container packedClass)
{
    container    base;
    boolean      ret;
    Integer      version = conPeek(packedClass,1);

    switch (version)
    {
        case #CurrentVersion:
            [version, #CurrentList, base] = packedClass;
            ret = true;
            break;
        default:
            ret = false;
    }
    return ret;
}
```

Unpack() тоже должен быть перекрыт. Переменные в макросе CurrentList инициализируются значениями при выполнении unpack().

```
static void main(Args _args)
{
    MyClass_RunBaseBatch runBaseBatch = new MyClass_RunBaseBatch();
    ;

    if (runBaseBatch.prompt())
        runBaseBatch.run();
}
```

```
}
```

Это основной статический метод, объявляющий и исполняющий класс. Main() используется для выполнения класса из АОТ или пункта меню. Если в диалоге пользователь нажал ОК, выполнится метод run().

```
protected Object dialog()
{
    DialogRunBase dialog = super();
    ;

    dialog.addGroup("Date");
    dialogFromDate = dialog.addFieldValue(typeId(FromDate), fromDate);

    return dialog;
}
```

В этом перекрытом методе объявляется диалог. Поле для ввода даты добавляется в диалог.

```
public boolean getFromDate()
{
    boolean ret;

    ret = super();

    if (ret)
    {
        fromDate = dialogFromDate.value();
    }

    return ret;
}
```

GetFromDate() вызывается, если нажата кнопка Ok в диалоге. Значение даты, введенное в соответствующее поле формы диалога, хранится в переменной fromDate.

```
client server static ClassDescription description()
{
    return "Testing batch able class";
}
```

Все классы, наследуемые от структуры runbase должны иметь этот метод. Так как метод статический, то вы не можете перекрыть его из родительского класса, поэтому description() следует создавать вручную. Метод добавляет текст в заголовок формы диалога.

```
public void run()
{
    CustTable    custTable;
    CustTrans    custTrans;
    Counter      totalRecords;
    ;

    select count(recId) from custTable
```

```

    exists join custTrans
      where custTrans.accountNum == custTable.accountNum
        && custTrans.transDate    >= fromDate;

totalRecords = custTable.recId;

startLengthyOperation();
this.progressInit("List customers with transactions", totalRecords, #AviSearch);

while select custTable
  exists join custTrans
    where custTrans.accountNum == custTable.accountNum
      && custTrans.transDate    >= fromDate
  {
    progress.incCount();
    progress.setText(sprintf("%1, %2", custTable.accountNum, custTable.name));
    sleep(500);
  }
endLengthyOperation();
}

```

Run() выполняет необходимый алгоритм, используя значения, введенные в диалоге. Первый select вычисляет число покупателей, у которых есть проводки, используя агрегатную функцию. Полученное число используется счетчиком для формы прогресса [progress bar], которое будет увеличиваться с каждым циклом обработки проводок покупателя. Глобальные методы startLengthyOperation() и endLengthyOperation() используются для установки экранного курсора в форму песочных часов. Вызванный макрос используется, как параметр при инициализации progress bar. Это - константа из библиотеки макросов AviFiles, определяющая форму прогресса. Вместо вывода информации об обработке в Infolog, progress bar обновляет её на форме с каждым циклом. Функция sleep() устанавливает задержку ½ секунды, делая возможным видимость progress bar.

Окончательно, создайте пункт меню для класса, просто перетаскив класс в узел *Output* узла MenuItems.

---

При выполнении класса MyClass\_Runnable появляется диалог. Диалог имеет две закладки. Первая закладка имеет группу полей «**Date**» для определения даты. Введенное значение сохраняется, и выводится по умолчанию, при следующем вызове класса. Метод prompt() в main() вызывает метод dialog(). Метод run() вызывается при нажатии кнопки Ok в диалоге и метод getFromDialog() возвращает true. Вам следует всегда начинать с перекрытия методов pack() и unpack() при создании класса наследника runbase. Если вы попытаетесь создать статический метод main() перед определением этих методов, то получите сообщение об ошибке, так как pack() и unpack() - часть интерфейса структуры класса runbase. Вторая закладка служит для установки параметров пакетной обработки. Даже если создаваемый класс не будет выполняться в пакетном задании, то вам лучше использовать наследование от класса RunBaseBatch. Если метод canGoBatch() перекрыт и возвращает false, закладка «Пакет» не появится.

Отметьте, что если вам необходимо установить проверку введенных значений в диалоге, то вам следует переопределить метод `validate()`. Диалог не может быть закрыт нажатием кнопки `Ok`, пока метод `validate()` не вернет `true`.

Вы можете запустить класс `MyClass_Runnable` через контекстное меню, выбрав *Открыть*. Так как класс имеет статический метод `main()` с параметром `Args`, то он будет запускаться без дополнительного кода. Это также называется исполнимый [runable] класс. Все классы с статическим методом `main()` – это исполнимые классы и могут выполняться через активацию пункта меню. Отметьте, что структура `runbase` не требует создание еще чего-либо для запуска класса. Вам следует запускать класс через пункт меню. Рекомендуется создавать пункт меню для выполнения вашего класса так же, как это будет делать пользователь. `Runnable` классы всегда запускаются через пункт меню.

В нашем примере была использована форма прогресса [progress bar]. Для задач, выполнение которых занимает более чем несколько минут, вам следует добавлять `progress bar`. Это делает реализацию задания через ваш класс более понятной так, как конечный пользователь видит - сколько времени осталось до конца. Так как `progress bars` – это часть структуры `runbase`, вы можете легко добавить `progress bar` в ваш класс наследник. Вы можете использовать `progress bars` для любого тяжелого блока кода. Классы с префиксом `SysOperationProgress*` используются для построения `progress bars`. Форма `Tutorial_Progress` покажет вам множество возможностей использования `progress bars`.

Для обзора возможностей анимации в `progress bars` посмотрите класс `Tutorial_ShowAviFiles`.

Параметр `Args` в `main()` используется для получения параметров вызывающего объекта. Это часто используется при вызове класса из формы, потому что при использовании `Args` вы можете получить курсор записи или идентифицировать вызывающий объект. Глава **Формы** покажет вам использование `Args`.

## Диалог

Диалог – это пользовательский интерфейс класса. Если вам необходим диалог для ввода определенных значений пользователем, то используйте структуру `runbase` так, как диалог в нем – это интегрированная часть структуры. Классы с префиксом `Dialog*` используются структурой. Простейшие возможности формы такие, как группировка полей, добавление формы выпадающего списка [lookups] по связанным полям содержатся в классах `dialog`. Вы можете добавить дополнительные клавиши в ваш диалог, такие как вызов запроса. Если ваш класс наследуется от класса `RunBaseReport`, то автоматически добавляются в диалог опция выбора полей из запроса отчета и настройки принтера.

---

### Пример 4: Диалог класса



Элементы MORPHXIT проекта Classes

- Class, MyClass\_RunBaseDialog
- Menu item action, MyClass\_RunBaseDialog

В этом примере будут показаны некоторые простые возможности диалога класса. Для упрощения кода методы description() и run() будут опущены.

Продублируем класс MyClass\_RunBaseBatch и переименуем в “MyClass\_RunBaseDialog”. Pack() и unpack() останутся без изменений. Метод run() следует удалить. Другие методы следует модифицировать, как показано ниже:

```
class MyClass_RunBaseDialog extends RunBaseBatch
{
    NoYesId      selectDate;
    FromDate     fromDate;
    ToDate       toDate;
    NoYesId      selectCustGroup;
    CustGroupId  custGroupId;
    DialogField  dialogFromDate, dialogToDate, dialogSelectCustGroup, dialogCustGroup;
    DialogGroup  dateGroup, countryGroup;

    #define.CurrentVersion(1)

    #localmacro.CurrentList
        fromDate,
        toDate,
        selectCustGroup,
        custGroupId
    #endmacro
}
```

Добавлены некоторые дополнительные поля. Созданы переменные для хранения значения каждого поля диалога. Макрос CurrentList расширен для хранения новых переменных.

```
protected Object dialog()
{
    DialogRunbase dialog = super();
;
    dialog.allowUpdateOnSelectCtrl(true);

    dateGroup = dialog.addGroup("Date");
    dateGroup.frameOptionButton(FormFrameOptionButton::Check);
    dateGroup.columns(2);

    dialogFromDate      = dialog.addFieldValue(typeId(FromDate), fromDate);
    dialogToDate         = dialog.addFieldValue(typeId(ToDate), toDate);

    countryGroup = dialog.addGroup("Customer");
    countryGroup.columns(2);

    dialogSelectCustGroup = dialog.addFieldValue(typeId(NoYesId), selectCustGroup,
```

```

        "Select customer group");
    dialogCustGroup    = dialog.addFieldValue(typeId(CustGroupId), custGroupId);

    return dialog;
}

```

Группа диалога содержит дату начала и дату конца. По умолчанию все поля диалога расположены в одной колонке. Метод `columns()` используется для установки даты начала и даты конца в одну строчку. Метод диалога `frameOptionButton()` используется для вставки поля типа галочки [checkbox] в заголовке группы поля. Другая группа полей `Country` имеет два поля. Если поле `dialogSelectCustGroup` не отмечено, то поле `dialogCustGroup` не доступно. Это происходит, если метод диалога `allowUpdateOnSelectCtrl()` установлен в `true`.

```

public void dialogSelectCtrl()
{
;
    if (dialogSelectCustGroup.value())
    {
        dialogCustGroup.allowEdit(true);
    }
    else
    {
        dialogCustGroup.allowEdit(false);
    }
}

```

Каждый раз при изменении в полях диалога выполняется этот метод, если метод диалога `allowUpdateOnSelectCtrl()` установлен в `true`.

```

public boolean getFromDialog()
{
    boolean ret;

    ret = super();

    if (ret)
    {
        fromDate      = dialogFromDate.value() ? dialogFromDate.value() : systemdateget();
        toDate        = dialogToDate.value() ? dialogToDate.value() : systemdateget();
        selectCustGroup = dialogSelectCustGroup.value();
        custGroupId    = dialogCustGroup.value();
    }

    return ret;
}

```

Значения, полученные из полей диалога присваиваются переменным, объявленным в `ClassDeclaration`. Поле проверки типа галка [checkbox] заголовка группы полей дата в этом случае не инициализируется значением.

```

static void main(Args _args)
{
    MyClass_RunBaseDialog runBaseDialog = new MyClass_RunBaseDialog();
;
}

```

```
if (runBaseDialog.prompt())
    runBaseDialog.run();
}
```

Метод `main()` должен быть изменен для выполнения нового класса.

Все что остается добавить – это новый пункт меню для созданного класса.

---

Класс ничего не выполняет так, как метод `run()` отсутствует. Метод `run()` – член родительского класса, поэтому его отсутствие при компиляции не выдаст ошибки. Выполнение диалога класса показано на **рисунке 21: Пример диалога**.

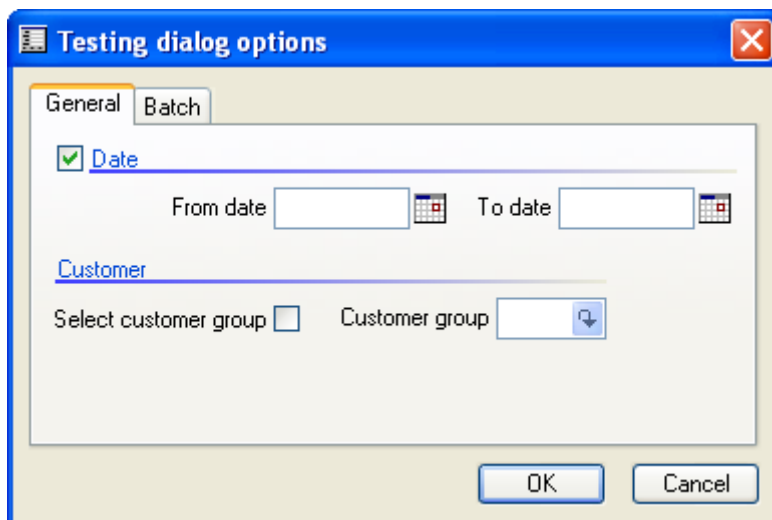


Рисунок 21: Пример диалога

Диалог показывает два возможных пути реализации зависимости изменения в одном поле от другого. Если снять галку в заголовке группы полей дата, то поля `from date` и `to date` могут правиться. В нашем случае можно было бы вводить диапазон между датами и при условии выключения перебирались бы все записи. Группа полей `Customer` использует сходный функционал. Если стоит галка в поле проверки `Select customer group`, то можно редактировать поле `Customer`. Метод `dialogSelectCtrl()` выполняется только для полей типа `DialogField`, это часто используется в стандартном функционале. Однако `dialogSelectCtrl()` не может обновлять диалог, поэтому диалог не обновляется новыми настройками после снятия или установки галки в поле `Select customer group`.

Альтернатива использованию диалога – это создание собственной формы. Структура `runbase` поддерживает использование стандартной формы в качестве диалога. Форма будет выводиться так же как диалог. Это более предпочтительное решение для реализации сложных диалогов. Если вам необходимо построить сложный диалог, то использование собственной формы значительно сохранит вам время. Сначала у вас будет простая форма диалога, но затем будет проще её доработать. Вы найдете более подробную информацию по использованию форм в диалогах в главе **Формы**.

В нашем примере можно просто добавить поле диалога, использующее запрос [query]. Используя метод диалога addMenuItemButton() кнопка с именем Выбор будет добавлена на форму диалога. Запрос будет выведен, если нажата кнопка выбор и перекрыт метод queryRun(). Добавьте следующий код в ваш метод dialog() и появится кнопка выбора, выводимая в форме диалога:

```
dialog.addMenuItemButton(MenuItemType::Display,
                        menuItemDisplayStr(RunBaseQueryDialog),
                        dialogMenuItemGroup::BottomGrp);
```

Перекрывая метод queryRun(), кодом, приведенным ниже, появится запрос, содержащий таблицу CustTable, связанную с таблицей CustTrans.

```
public QueryRun queryRun()
{
    QueryRun ret;

    ret = new QueryRun(QueryStr(Cust));

    return ret;
}
```

В деталях, как строить запрос и получать информацию из запроса объясняется в главе **Запросы**.


Полезная возможность использования структуры runbase в том, что последнее введенное значение поля легко сохраняется до следующего вызова. Обычно это используется для хранения значений полей диалога, однако, можно сохранять любую переменную. Вам только следует добавить такую переменную в макрос CurrentList. При изменении CurrentList в текущем классе, вам следует увеличить константу макроса CurrentVersion на один так, как может возникнуть ошибка из-за использования той же самой версии при изменении списка сохраненных переменных. Последние значения хранятся в системной таблице SysLastValue для пользователя и компании. Увеличение CurrentVersion создаст новую запись в SysLastValue.

Pack() и unpack() должны быть перекрыты так, как они являются частью интерфейса структуры runbase. Так же стоит помнить об объявлении двух макросов в ClassDeclaration. Возможно, у вас будут ситуации, когда получение последнего значения не требуется. Для этого pack() должен вернуть пустой контейнер connull(), а unpack() должен вернуть false.

Если вам просто необходим модальный диалог, для выбора продолжить ли операцию или нет, то существует более простое решение, чем использование структуры runbase. Класс Vbox имеет коллекцию статических методов, который поддерживает все простые комбинации. Вы можете посмотреть вызов метода из класса vbox, используя, оператор if:

```
static void Classes_Box(Args _args)
{
    if (Box::yesNo("Continue", DialogButton::Yes, "Test of Box") == DialogButton::Yes)
        info("Here goes.");
}
```

## 5.4 Фундаментальные классы

Последние классы в списке АОТ в узле *Classes* – специальный тип классов. Это фундаментальные классы, которые не являются обычными классами и не могут быть унаследованы. Вы можете узнать фундаментальные классы по иконке .

Вы будете использовать фундаментальные классы для ваших модификаций, даже не подозревая об этом так, как эти классы не объявляются, как обычные классы. Самое важное, что фундаментальные классы автоматически объявляются ядром при старте клиента. Базовые знания в фундаментальных классах, помогут вам понять некоторые элементарные процессы в MorphX. Ниже вы найдете объяснение основных важных моментов в фундаментальных классах.

### ClassFactory

Всегда, при вызове объекта пользовательского интерфейса, такого как форма, отчет или диалог, активируется класс ClassFactory, по меньшей мере, при исполнении объекта, следует вызывать ClassFactory. При вызове формы, отчета или запроса из X++, объект инициализируется с использованием ClassFactory. Это автоматически делается при вызове объекта через пункт меню или использовании структуры runbase.

Цель использования класса ClassFactory – некоторый фундамент. Он применяется при общих изменениях форм или отчетов таких, как добавление заголовков или полей в формы. Примеры использования ClassFactory для форм или отчетов смотрите в главах **Формы** и **Отчеты**.

Отметьте, что при работе в 3-уровневой конфигурации используется две сущности ClassFactory. Одна определяется на AOS, а другая на клиенте. При выполнении объекта, используется сущность ClassFactory текущего уровня. Если вы используете ClassFactory для открытия формы, выполняемой на клиенте, и запускаете класс на AOS вы не получите один и то же результат так, как две сущности обрабатывают отдельно. Цель данной технологии – уменьшить число вызовов между клиентом и сервером.

## Global

Все методы в классе Global статические. Вы можете создать и динамический метод в классе Global. Однако что бы затем его использовать, вам придется объявлять класс. Идея использования класса Global – это коллекция функций, созданных в X++, которые можно использовать, не определяя объект класса. Методы global можно также использовать, как и системные функции.

Некоторые методы – дополнительные функции, использующие операции с базовыми типами, например, метод класса Global `date2StrUsr()`, который конвертирует переменную из типа дата в тип строку, форматируя согласно пользовательским настройкам.

Вы можете добавлять ваши собственные методы в класс Global. Это может быть полезно, если у вас имеется часто используемый блок кода. Перед добавлением своего метода следует проверить существующие методы и существующие системные функции, расположенные под узлом *System Documentation/Functions*.

## Info

К системе Infolog можно обратиться, используя класс Info. Сущность InfoLog класса Info, инициализируется при старте Axapta. Вам следует использовать класс Info, при выводе сообщений в Infolog. Вам никогда не следует ссылаться на класс Info напрямую, так как это уже сделано при старте системы, вместо этого используйте глобальные методы-функции `info()`, `warning()` и `error()`.

Класс Info отрабатывает и другие задачи, так как класс используется и при вызове справки, и для расчета лицензии пользователя, и для производства импорта объектов.

Вы можете найти более подробную информацию по использованию Infolog в главе **Введение в MorphX**.

## 5.5 Системные классы

Системные классы используются также как классы приложения. Системные классы могут быть унаследованы также как и классы приложения. Некоторые классы приложения в стандартной поставке являются наследниками системных классов. Так как вы не можете изменять системные классы, то наследованием от системного класса становится возможным добавлять дополнительную логику в работу системного класса. Такие классы как `SysDictTable` или `SysReportRun` являются наследниками системных классов.

Типы наиболее часто используемых системных классов описаны в следующем разделе.

## Object

Системный класс Object это основа всех классов в MorphX. Вы можете сравнить Object с системной таблицей Common. Любой класс может быть объявлен, используя класс Object. Так как MorphX проверяет методы объекта во время компиляции, то использование класса Object очень выгодно, так как тип объекта может быть не известен. Это следует использовать при компиляции ваших методов формы или отчета, когда их имя еще не известно. Пример такого использования вы можете найти в главе **Формы**.

## Изменения во время исполнения

Одно из основных предназначений системных классов – это возможность обратиться к любому узлу АОТ. Это бывает особенно необходимо для форм и отчетов так, как вы имеете возможность обращаться к любому свойству или методу объекта. В данной книге вы найдете примеры использования системных классов для обращения к объектам АОТ тем самым, делая ваш код более динамичным. Системные классы, используемые для обращения к определенным узлам АОТ, обычно имеют префикс – ссылку на узел АОТ подобно всем системным классам для обращения к узлам формы имеющим префикс Forms\*.

## Args

Вы, может быть, заметили системный класс Args используемый для передачи параметров. Любой исполнимый класс, как правило, имеет Args в профиле параметров метода main(). Цель использования Args – передача параметров между объектами. Если вы вызываете класс из формы, то вы можете использовать Args для передачи объекта formRun, делая возможным получать данные из формы и даже выполнять методы вызывающей формы.

---

### Пример 5: Args

#### Элементы MORPHXIT проекта Classes

- Class, MyClass\_Args
- Menu item action, MyClass\_Args
- Form, CustTable

Вам придется добавить новый пункт меню в форму клиентов. Класс будет запускаться этим пунктом меню, и сообщать о выбранных записях в форме.

1. Создайте новый класс и переименуйте в “MyClass\_Args”.
2. Класс имеет только метод main(), который запускает класс. Args используется для проверки вызывающего объекта, в данном случае есть ли у него в

источнике данных таблица CustTable. Вызывающий объект может быть любой формой, только использующий указанный источник данных CustTable. Если проверка вернула true, табличная переменная CustTable инициализируется вызывающей записью и выводится результат.

```
static void main(Args _args)
{
    MyClass_Args      myClassArgs = new MyClass_Args();
    CustTable         custTable;
;

    if (_args.dataset() == tablenum(CustTable))
    {
        custTable = _args.record();
        info(strfmt("Customer selected: %1", custTable.name));
    }
}
```

3. Сохраните класс и создайте пункт меню для MyClass\_Args.
  4. Перейдите к узлу *Forms* в AOT найдите форму CustTable. Раскройте узел формы и перейдите к дизайну. Перетащите пункт меню MyClass\_Args в узел *Design/ButtonGroup:ButtonGroup* и сохраните форму.
- 

При открытии формы CustTable вы увидите, что на форме есть дополнительная кнопка с таким же именем, что и класс. Так как метка не была определена для пункта меню, то показывается имя класса. Запустите класс, нажав на кнопку, и название клиента, на котором стоит курсор, будет выведен в Infolog. Это был простой пример использования Args. Вы можете использовать Args для вызова отчета и добавления фильтров к нему, например, вывода только выделенных записей. Класс Args часто используется для передачи параметров в класс, при выполнении им тяжелых задач. Так как форма всегда выполняется на клиенте, становится возможным помещать рабочую нагрузку по выполнению задачи на AOS. Просматривая дерево AOT, вы найдете множество примеров использования Args.

## Базовые классы

В MorphX есть группа системных классов, которые можно использовать как альтернативу сложным типам данных. Они называются foundation классами. Основное свойство foundation классов – хранение динамического списка переменных. Значения хранятся в памяти, так что их можно быстро использовать. По сравнению с временной таблицей, которая хранит данные на диске, фундаментальные классы работают гораздо производительнее.

В MorphX представлены пять foundation классов: Array, List, Map, Set и Struct. Foundation классы разработаны для использования любого типа данных. Полезная



вещь в том, что вы можете использовать фундаментальные классы в качестве параметра для метода или использовать метод, возвращающий foundation класс.

Вы можете найти примеры использования foundation классов при просмотре АОТ или нажать F1 для просмотра справки.

## Оптимизация операции записи

Если у вас есть задача, в которой необходимо выводить некоторые записи несколько раз, то вам следует использовать класс отсортированного списка для лучшего представления. В этом случае выводится большое количество записей, но перед выводом класс обновляет те же самые записи. Обычно вам необходимо вывести записи в классе при вызове отчета, но запрос отчета может вывести записи второй раз. Для предотвращения вывода записей второй раз, вы можете использовать системный класс `RecordSortedList`. Разноска и распечатка счетов фактур – это одно из мест в стандартной конфигурации, использующее `RecordSortedList`.

Если вам нет необходимости сортировать ваши записи, или если они уже отсортированы в заказе, то лучше использовать `RecordLinkList`.

Системный класс `RecordInsertList` - это альтернатива ключевого слова в запросе `insert_recordset`. Однако `insert_recordset` использовать предпочтительнее.

За примером использования оптимизации операции записи смотрите [online справку](#).

## Обработка файлов

Импорт данных из простого текстового файла – это очень просто, так как MorphX имеет специальный системный класс, работающий с файлами. Базовый класс, обрабатывающий файлы – это системный класс `Io`. Системный класс `AsciiIo` и `CommaIo` - наследники `Io`, работающие с текстовыми файлами.

```
static void Classes_CommaIo(Args _args)
{
    CommaIo      fileOut;
    FileName      fileName = "c:\\Customers.csv";
    CustTable      custTable;
;
    #File

    fileOut = new CommaIo(filename, #io_write);

    if (fileOut)
    {
        while select custTable
```

```

    {
        fileOut.write(custTable.accountNum,
                     custTable.name,
                     custTable.custGroup,
                     custTable.currency);
    }
}
}

```

Пример показывает использование CommaIo. Клиенты из CustTable выбираются в цикле, и записываются в отдельный comma файл. Константа из библиотеки макросов используется для установки записи при объявлении класса CommaIo. Метод write() в классе CommaIo использует любое число полей в качестве параметров. Отметьте, что необходимо использовать двойные слэши при ссылке на путь файла. В MorphX вам не обязательно закрывать файл, так как это будет сделано автоматически.

---

**Примечание:** Если вам необходимо экспортировать сложные типы данных такие, как контейнеры, то это можно сделать, используя системный класс BinaryIo. Альтернативно вы можете экспортировать ваши данные через XML.

---

## 5.6 Специальное использование классов

В этом разделе показываются примеры использования классов для специальных нужд. Некоторые главы этой книги имеют примеры, как эта, показывающая использование классов для интеграции с внешними приложениями.

### Использование COM

Так же, как работает Business Connector, COM connector может использоваться для интеграции Ахарта с внешними продуктами. В этом примере получаем доступ к Microsoft Outlook, используя COM. Для использования этого примера, у вас должен быть, как минимум, один зарегистрированный COM пользователь.

#### Элементы MorphxIt проекта Classes

##### ➤ Job, Classes\_ReadFromOutlook

```

static void Classes_ReadFromOutlook(Args _args)
{
    SysOutlookApplication      sysOutlookApplication;
    SysOutlook_NameSpace       sysOutlookNameSpace;
    SysOutlookMapiFolder       sysOutlookMapiFolder;
    SysOutlook_Folders         sysOutlookFolders;
    SysOutlook_Items           collection;
    COM                         message;
    Notes                      messagebody;
;

```

```

#sysOutLookComDef

sysOutlookApplication = new sysOutlookApplication();
sysOutlookNameSpace = sysOutlookApplication.getNameSpace("MAPI");

sysOutlookNameSpace.logon();
sysOutlookFolders      = sysOutlookNameSpace.Folders();
sysOutlookMapiFolder   =
sysOutlookNameSpace.getDefaultFolder(#OlDefaultFolders_olFolderInbox);

collection      = sysOutlookMapiFolder.Items();
message         = collection.GetFirst();

while (message)
{
    info(message.subject());

    message = collection.GetNext();
}
}

```

Классы с префиксом SysOutlook\* используются для доступа к приложению Microsoft Outlook. Класс SysOutlookApplication открывает COM connection и SysOutlookNameSpace логинится в Microsoft Outlook. Если появится предупреждение, то вам следует принять доступ к вашему почтовому клиенту. Макро библиотека SysOutlookComDef содержит список констант используемых для Microsoft Outlook. По умолчанию выбирается папка Входящие в Microsoft Outlook и темы всех писем из Входящих выводятся в Infolog.

В этом примере рассматривается только чтение из Microsoft Outlook. Однако вы можете записывать или синхронизировать вашего почтового клиента с данными из Ахарта. Форма HRMInterviewTable использует Microsoft Outlook для создания встреч в календаре, основанном на данных из Ахарта.

---

**Примечание:** Если вы собираетесь использовать интерфейс внешней системы, то можете использовать Мастер Оболочек Для COM Объектов [COM Class Wrapper Wizard] для создания COM оболочки вашей внешней системы. Классы SysOutlook\* сделаны при помощи мастера.

---

## X++ Compiler

MorphX имеет специальный системный класс XppCompiler, который используется при компиляции X++ кода, написанного, например, в текстовом файле. Вы, может быть, хотите знать, для чего использовать этот класс, если всё интегрировано в MorphX? Это, может быть, путь предоставления администратору системы написания своих собственных проверок на X++ не связанных с изменениями в АОТ. На тестовом оборудовании с использованием этого класса вы можете делать модификации более гибкими, позволяя пользователю делать некоторые настройки.

Существуют места в Ахарт, где пользователь имеет некоторые жестко предопределенные переменные, которые могут использоваться в текстовом поле таком, как форма TransactionsTexts. А что, если вместо использования жестко предопределенных переменных, вы определите ваши собственные переменные, написанные в X++? Это будет достаточно красиво, и использованием класса XppCompiler вы сможете этого достигнуть.

#### Элементы MorphxIt проекта Classes

- Class, Classes\_XppCompiler
- Menu item output, Classes\_XppCompiler

У созданного класса будет два метода. Класс должен быть исполнимым [runable].

```
Notes buildFunction()
{
    TextBuffer    textBuffer = new TextBuffer();
    ;

    textBuffer.appendText("static void test(Counter _counter, CustTable _custTable)");
    textBuffer.appendText("{");
    textBuffer.appendText("while select _custTable");
    textBuffer.appendText("{");
    textBuffer.appendText("info(_custTable.name);");
    textBuffer.appendText("_counter++;");
    textBuffer.appendText("}");
    textBuffer.appendText("info(strfmt(\"Customers printed: %1\\", _counter));");
    textBuffer.appendText("}");

    return textBuffer.getText();
}
```

Этот метод класса создает код функции для последующего исполнения классом XppCompiler. Эта функция имеет два параметра счетчик и таблицу CustTable. Все записи из CustTable перебираются в цикле функции. Имена клиентов и их общее количество затем выводится в Infolog.

Системный класс TextBuffer используется для построения строки кода функции. Добавление строк, с использованием символа + очень медленно. Следует всегда использовать TextBuffer при добавлении нескольких строчек.

```
static void main(Args _args)
{
    Classes_XppCompiler classXppCompiler= new Classes_XppCompiler ();
    xppCompiler          compiler = new xppCompiler();
    Notes                codeString;
    CustTable            custTable;
    ;

    codeString = strfmt(classXppCompiler.buildFunction());

    if (compiler.compile(codeString))
    {
```

```
    runbuf(codeString, 0, custTable);  
  }  
  else  
  {  
    info(compiler.errorText());  
  }  
}
```

Метод `main()` объявляет класс `XppCompiler`. Если код, построен в `buildFunction()` без ошибок, то функция `runbuf()` выполнит код. Первый параметр `runbuf()` это строка кода для исполнения, а следующие параметры определены для функции. Все переменные, используемые в созданной функции, должны быть предопределены, и передаваться параметрами, их вам не следует объявлять в теле функции.

## 5.7 Резюме

Классы – это фундаментальная часть в MorphX. Классы приложения используются для создания бизнес логики. Системные классы связывают ядро с приложением. В этой главе объяснялись отличия между различными классами особенность их использование . Теперь у вас есть необходимые знания об использовании классов в MorphX. В следующей главе вы узнаете, как комбинировать классы для разработки пользовательских интерфейсов.



## 6 Формы

Формы – это наиболее важная часть пользовательского интерфейса, так как формы являются связующим звеном между пользователем и базой данных. При вставке, обновлении или удалении данных пользователем, используются как раз формы. Посмотреть и изменять данные в базе можно в обозревателе таблицы, но вам не следует рассматривать эту возможность для рабочей системы, так как пользователь может испортить данные. По сравнению с обозревателем таблицы в функционал формы обычно входят дополнительные проверки, помимо этого форма структурирует данные при выводе, делая легким их использование.

Форма имеет множество функций для пользователя такие, как сортировка, наложение фильтра и идентификация появления формы. Некоторые из этих возможностей будут упомянуты в этой главе. Однако мы будем рассматривать, использование MorphX для построения формы.

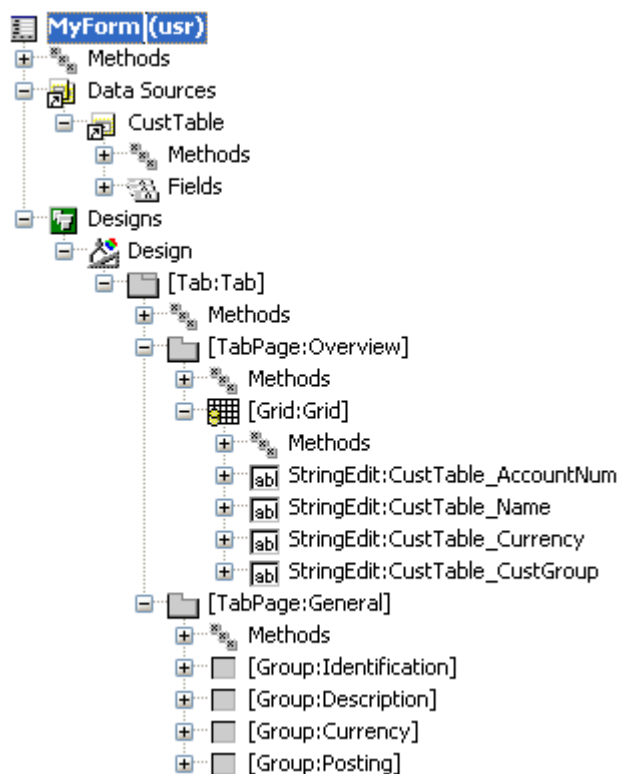


Рисунок 22: Обзор формы

### 6.1 Создание форм

Форма в Ахарта создается в узле AOT *Forms*. Форма состоит из двух частей, запроса, обрабатывающего данные, и дизайна, который определяет вывод данных.

Дизайн обычно используется для представления полей таблиц, связанных запросом.

---

### Пример 1: Моя первая форма

#### Элементы проекта MORPHXIT Forms

- Form, MyForm
- Menu item display, MyForm

Для получения опыта в создании форм, создайте форму, как показано на **Рисунке 22: Обзор формы**. Это простая форма, которая научит вас базовым шагам создания форм. В следующих примерах мы поподробнее рассмотрим эту форму и добавим новые возможности. Для упрощения мы не будем использовать метки, однако метки всегда следует создавать в ваших модификациях.

1. В контекстное меню узла *Forms* выберите *Создать Form*. Переименуйте форму в "MyForm", используя лист свойств.
2. Раскройте узел новой формы и перейдите к новым узлам формы таким, как *Data Sources* и *Designs*, которые теперь видимы. Откройте еще одно окно AOT, найдите таблицу SalesTable и перетащите SalesTable в узел *Data Sources* формы.
3. Перейдите к узлу *Designs/Design* и откройте лист свойств. В свойстве **Caption** введите текст "Sales orders". Выберите источник данных SalesTable в свойстве **TitleDatasource**.
4. В контекстном меню узла *Designs/Design* выберите *Создать Control/Tab*. Перейдите к листу свойств поля tab и установите свойство **Width** в "Column width" и **Height** в "Column height".
5. Теперь добавим закладку [tab page], для этого в контекстном меню узла *Designs/Design/[Tab:Tab]* выберем *Создать Control/TabPage*. Переименуем новую закладку в "Overview", используя лист свойств. Введите новое название для закладки, просто напечатав "Overview" в свойстве **Caption**.
6. Добавим поле табличного документа [grid control] на закладку Overview, для этого в контекстном меню узла *Designs/Design/[Tab:Tab]/[TabPage:Overview]* выберем *Создать Control/Grid*.
7. Перейдем к новому полю табличного документа и установим свойства Width и Height в "Column width" и "Column height".
8. Раскроем источник данных SalesTable и выберем поля SalesId, CustAccount, CurrencyCode и SalesStatus. Перетащите эти поля в поле табличного документа



в упомянутом порядке. Эти простейшие операции можно сделать в контекстном меню источника данных SalesTable, а можно открыть новое окно АОТ и перетащить необходимые поля из нового окна.

9. Добавим вторую закладку [Tab:Tab] через контекстное меню узла. Переименуйте её в “General”, установив свойство Caption в "General".
  10. Перейдите к источнику данных SalesTable, раскройте узел *Fields* источника данных. Группы полей [field groups], определенные на таблице SalesTable представлены списком после самих полей таблицы источника данных. Выберите группы полей Identification, Currency, Customer и Status. Перетащите их в упомянутой последовательности в узел дизайна *Designs/Design/[Tab:Tab]/[TabPage:General]*.
  11. Сохраните форму. Теперь вы создали свою первую форму!
  12. Создайте пункт меню [menu item] для формы, просто перетащив форму MyForm в узел *Menu Items/Display*.
- 

Созданная форма – это пример простой формы в Ахapta. Не было создано необходимых настроек по связи источников данных. Просто выбрали необходимую таблицу и перетащили в источник данных, который является запросом [query] формы. Создание интерфейса формы действительно просто, так как MorphX авто позиционирует поля формы [controls]. При перетаскивании поля в дизайн, автоматически создается поле необходимого типа и автоматически располагается между другими полями.

Самое замечательное, что не было написано ни одной строчки кода при создании этой формы. Мы только настроили несколько свойств. Вам часто будет необходимо добавлять код в ваши формы, но этот пример показывает насколько просто представлять данные из таблицы. Дизайн всегда авто позиционируется, даже для сложных форм, а это существенно сокращает время разработки.

Свойства, установленные на дизайне этого примера почти всегда установлены и на стандартных формах. Текст заголовка [caption] формы был определен на узле Design. Свойство Caption определяет название формы (в заголовке крайний левый текст), а свойство TitleDatasource показывает в заголовке формы так же значение полей, определенных в свойствах уже таблицы TitleField1 и TitleField2 и располагается после названия формы.

Устанавливая свойства Width и Height в значение Column width и Column height, вы делаете возможным изменение размеров формы пользователем (появляется бегунок). Свойства Width и Height устанавливаются для обеих закладок и поля табличного документа [grid] делая возможным наглядную подгонку к размерам формы также и поля табличного документа.

Поля на таблице всегда должны быть оформлены в группы, так как это упростит разработку: при изменении поля в группе меняются все формы, где эта группа используется или же надо менять на каждой форме, где это поле используется. Поле группы [group control] имеет свойство **AutoDataGroup**. Изменения сделанные в группе полей на таблице автоматически отобразятся на форме или отчете при установке свойства AutoDataGroup в Yes. Упрощается добавление нового поля в существующую группу полей [field group] на форме, так как изменения следует вносить только в словаре данных [data dictionary]. Для упрощения мы не использовали свойство группы полей в поле табличного документа. Вы можете перетащить группу полей и в поле табличного документа или определить свойство **DataGroup** у поля табличного документа. Стандартная таблица, как правило, уже имеет группу полей с названием Overview, которая содержит поля для использования в поле табличного документа.

---

**Примечание:** Если вы хотите узнать больше о возможностях форм, то поищите в АОТ формы с префиксом tutorial\*.

---

## 6.2 Запрос формы

Запрос формы, по сути это источник данных, содержащий таблицы, используемые в форме. Не обязательно иметь запрос в вашей форме. Вы можете выводить данные в форму, используя выборку [select]. Однако использование запроса должно быть вашим основным решением, так как он обладает некоторыми возможностями для пользователя такими, как фильтр, сортировка и печать данных.

Узлы, используемые для построения запроса формы в АОТ, немного отличаются от построения запроса в других местах АОТ, например, запрос в отчете. Запрос отчета имеет логическое построение с представленными таблицами и уровнем соединения [join level] в дереве, а запрос формы представляет только список всех таблиц на одном уровне дерева узла *Data Sources*. При добавлении таблицы в источник данных [data source] формы, вы добавляете таблицу в запрос формы источника данных. Это может немного смущать, при переключении разработки с формы на отчет. Скоро вы поймете эти отличия, так как свойства похожи. Формирование запроса в X++ происходит тем же самым способом вне зависимости от типа используемого объекта.

За дальнейшей информацией об основах запроса смотрите главу **Запросы**.

### Объединение источников данных

Самый простой путь организации фильтра записей по таблицам - это объединение источников данных [joining data sources]. Этого можно достичь объединением источников данных формы или, например, вызвав эту форму из другой формы, у которой имеется связь по источнику данных.

---

**Пример 2: Форма, использующая внешнее объединение (outer joined)**

---

Элементы проекта MORPHXIT Forms

- Form, MyForm\_OuterJoin
- Menu item display, MyForm\_OuterJoin

Форма, созданная в примере 1, будет модифицирована для вывода получателя из связанной таблицы сотрудников. Мы также для упрощения не будем создавать метки в этом примере.

1. Продублируйте форму MyForm и переименуйте в “MyForm\_OuterJoin”.
  2. Раскройте узел *Data Sources* новой формы, откройте еще одно окно AOT и найдите таблицу EmplTable. Перетащите EmplTable в узел *Data Sources*.
  3. Откройте лист свойств узла *Data Sources/EmplTable*. Подставьте SalesTable в свойство **JoinSource**. Установите свойство **LinkType** в OuterJoin.
  4. Выберите поле SalesTaker из SalesTable и перетащите его в узел *Designs/Design/[Tab:Tab]/[TabPage:Overview]/[Grid:Grid]*.
  5. Повторите шаг 4 добавлением поля Name из EmplTable в поле табличного документа [grid].
  6. Выберите группу полей Name из EmplTable и перетащите эту группу на закладку General.
  7. Перейдите к закладке General и установите свойство **Columns** в 2.
  8. Сохраните форму и создайте пункт меню [display menu item] для неё.
- 

Форма MyForm\_OuterJoin теперь имеет два дополнительных поля в поле табличного документа, показывающих получателя и имя сотрудника. Имя сотрудника выводиться из связанной таблицы EmplTable. Выводится все еще такое же количество записей, так как источники данных связаны по outer join. Изменив свойство на inner join, в форме уже будут отражаться только те заказы, у которых заполнено поле получатель в заказах. Попробуйте изменить свойство LinkType у источника данных EmplTable для просмотра, как изменится вывод данных.

При добавлении источника данных в форму перетаскиванием таблицы из словаря данных, имя его будет такое же, что и имя таблицы. Однако вы можете изменить имя источника данных. Если вы объединяете одну и ту же таблицу более чем один раз в запросе, то следует менять имя. Это подобно объявлению двух переменных одной и той же таблицы в X++.

Свойство Columns, измененное для закладки General разделит группы полей на две колонки. По умолчанию все группы полей представляются списком в одну колонку, так что количество колонок можно изменить, если ваша форма имеет несколько групп полей.

---

**Примечание:** Если вы не уверены в том, каким образом связаны таблицы, то попробуйте создать запрос в АОТ с объединением этих таблиц. В запросе АОТ связи будут автоматически показаны, установкой свойства Relations в true в подчиненном источнике данных.

---

Объединение таблиц для вывода данных из них на форму очень наглядно и его следует использовать для таблиц, где существует связь один к одному (1-1 relation). Для пользователя все равно, откуда берутся поля: из одной таблицы или нескольких. Пользователь даже может не подозревать, что данные выводятся из нескольких таблиц.

Так как нам необходимо получать данные из SalesTable и из EmplTable в поле табличного документа, то в нашем примере способ объединения [join mode] должен быть inner join или outer join. При использовании exist join или not exist join не будут показываться данные из подчиненной таблицы.

#### Типы связи источников данных на форме

Способ объединения источников данных в форме устанавливается настройкой свойства источника данных **LinkType**. Помимо стандартных способов объединения, формы имеют еще и 3 дополнительных способа, которые используются только на форме для более наглядного отображения данных: Passive, Delay и Active. При использовании inner join или outer join выводятся все записи из подчиненного источника данных. Это необходимо для отображения данных в поле табличного документа. Если данные из подчиненного источника данных не входят в одно и то же поле табличного документа, то вам следует использовать один из специальных способов объединения источников данных формы.

Значение этого свойства LinkType по умолчанию - Delayed. Это самый распространенный способ, при объединении источников данных и выводе их в одном поле табличного документа. Способ объединения delayed делает вывод данных из подчиненного источника с задержкой, то есть возникает эффект наглядного перестроения строчек заказа при переходе на следующий заказ. Это очень наглядно при ускоренной прокрутке по записям в поле табличного документа. Это дело нескольких миллисекунд и пользователь с трудом будет знать, что данные выводятся с задержкой. Форма SalesTable использует эту технологию.

---

**Примечание:** При использовании способов объединения Delayed, Active и Passive таблицы всегда объединяются, как outer join. Все записи выводятся из подчиненного источника данных.

---

Способ связи Active очень похож на Delayed. Единственное отличие в том, что Active не имеет задержки перед выводом данных из подчиненного источника данных. Active не часто используется так, как это не предоставляет пользователю той гибкости которая есть при использовании способа delayed.

Способ Passive – прямая противоположность Active. При Active данные сразу выводятся и обновляются из подчиненного источника данных, а при Passive данные не обновляются. Имеет смысл использовать Passive, если вы хотите контролировать вывод данных, используя X++.

### Связи в форме

Отношения [relations], определенные на таблицах или расширенных типах данных автоматически используются для связи источников данных на форме. Например, вы создали новую форму, вызываете её из формы клиентов и выводите в ней данные о клиентах. Если таблица для новой формы имеет поле с расширенным типом CustAccount, то ваша новая форма будет автоматически использовать связь с формой клиентов. Расширенный тип CustAccount имеет настроенное отношение [relation] к таблице CustTable и используемой форме клиентов. При вызове вашей формы из формы клиентов, MorphX создает связь между двумя формами, через расширенный тип данных. Этот тип связи называется динамическая связь [dynalinks].

Попробуйте открыть форму клиентов CustTable и открыть форму проводок по кнопке Проводки в форме клиентов. При переходе в форму проводок, проводки автоматически открываются только для текущей (активной) записи клиента в форме клиентов. Это происходит без написания кода и это поведение формы по умолчанию отрабатывает по установленным отношениям на связанных таблицах.

В некоторых случаях необходимо удалять динамическую связь [disabling dynalinks]. Например, вам необходимо выводить все записи в связанной форме или настроить альтернативную связь между формами.

```
void init()
{
    super();

    this.query().dataSourceNo(1).clearDynalinks();

    criteriaOpen = this.query().dataSourceNo(1).addRange(fieldnum(CustTrans,Closed));

    custTransDetails = new CustTransDetails(custTrans);
}
```

Для удаления динамической связи формы проводок покупателя, найдите форму CustTrans в АОТ и исправьте метод init(), расположенный в узле *Data Sources/CustTrans*. После вызова super() в init() удалите динамическую связь, как показано в верхнем блоке кода. Теперь при открытии формы проводок

покупателей из формы покупателей записи не фильтруются по клиенту, а выводятся все.

Причина для удаления связи может быть в том, что вам необходимо показывать все записи в связанной форме, так как связанная форма используется для просмотра всех записей. Это может быть случай вызова формы просмотра. Вместо использования динамической связи, основанной на связи в словаре данных, вы можете создать свою собственную связь. Попробуйте дополнить функционал в методе `init()` как это сделано в следующем блоке кода:

```
void init()
{
    super();

    this.query().dataSourceNo(1).clearDynalinks();
    this.query().dataSourceTable(tablenum(CustTrans)).addDynalink(
        fieldnum(CustTrans, CurrencyCode),
        element.args().record(),
        fieldnum(CustTable, Currency));

    criteriaOpen = this.query().dataSourceNo(1).addRange(fieldnum(CustTrans,Closed));

    custTransDetails = new CustTransDetails(custTrans);
}
```

После вызова `clearDynaLinks()` настраивается новая динамическая связь, используя метод `addDynalink()`. Вы должны определить поле в текущем источнике данных для связи с вызывающим объектом и вызывающей записью. Здесь `element.args().record()` используется для получения курсора из вызывающего объекта.

Добавляя эту динамическую связь, проводки покупателя теперь фильтруются по коду валюты клиента вместо клиента. Это, конечно, не имеет смысла в рабочем приложении. Это просто пример, показывающий, как можно перестроить динамическую связь по умолчанию.

---

**Примечание:** Если вы не уверены, какие поля используются в динамической связи для фильтрации, то посмотрите заголовок формы. Поля динамической связи представлены списком в последней части текста заголовка, после названия формы и полей `title1 title2`. Иногда эта связь не видна, так как поля, по которым осуществляется связь, включены в `title1` и `title2`

---

## Установка доступа

Общие права доступа настраиваются на таблицах и полях в словаре данных. Специфические права для форм следует определять на источнике данных формы. Вы не можете переопределить сделанную настройку в словаре данных, но вы можете добавить дополнительные ограничения по доступу. Ограничение по доступу может быть установлено и на уровне источника данных и на уровне поля

источника данных. Для всех полей источника данных вы можете определить максимальный уровень доступа, который определяется свойствами AllowEdit, AllowCreate и AllowDelete источника данных. На одном поле источника данных вы можете установить ограничение только для этого поля, например, может ли поле правиться или редактироваться. Ограничение доступа можно установить также и для поля в дизайне формы, не связанного с источником данных. Однако ограничение доступа следует всегда устанавливать в источнике данных так, как одно и то же поле может быть использовано более одного раза в дизайне формы. Предпочтительно не писать код в дизайне формы и не модифицировать настройки дизайна по умолчанию. Программирование в MorphX подразумевает не писать код в форме так, как это делает легким переключение пользовательского интерфейса на web интерфейс. Если вы не можете ограничить доступ к полю дизайна по причине того, что поле дизайна не связано с источником данных, то вам следует установить ограничение доступа в дизайне.

Некоторые ограничения доступа обычно настраиваются в свойствах. Форма CustTrans показывает проводки покупателя, которые нельзя править. Часто вам бывает нужно устанавливать ограничения доступа в зависимости от вызывающего объекта или в зависимости от одной записи в форме.

---

### Пример 3: Установка доступа

#### Элементы проекта MORPHXIT Forms

- Form, MyForm\_SettingAccess
- Menu item display, MyForm\_SettingAccess

Пример показывает, как настроить доступ для источника данных формы на X++. Установка доступа в X++ обычно базируется на каком-нибудь условии. Для упрощения кода не будем добавлять условие.

Продублируйте форму MyForm и переименуйте новую форму в “MyForm\_SettingAccess”. Переопределите метод init() формы и добавьте следующий код:

```
public void init()
{
    super();

    salesTable_ds.allowCreate(false);

    salesTable_ds.object(fieldnum(salesTable, CurrencyCode)).allowEdit(false);
    salesTable_ds.object(fieldnum(salesTable, SalesStatus)).visible(false);
}
```

Сохраните форму и создайте пункт меню для этой формы.

---

В открытой форме, вы не можете добавлять новую запись. Поле CurrencyCode не может редактироваться, а поле SalesStatus не показывается в дизайне формы.

На любое свойство источника данных или можно ссылаться из X++, используя название источника данных с суффиксом `_ds`. При ссылке на поле источника данных, вам следует использовать метод источника данных `object()`. Используя идентификатор поля таблицы как параметр для `object()` вы можете ссылаться на любое поле источника данных. В стандартной конфигурации чаще устанавливается ограничение доступа на поле в дизайне формы, чем ограничение доступа на поле источника данных. Это не оптимально, но причина в том, что использование обращения к полям источника данных была представлена в последней версии Ахпта.

## 6.3 Дизайн

Некоторые средства разработки заставляют тратить время впустую, делая дизайн формы, то есть, добавляя поля на форму вручную. Но это не случай MorphX. В действительности, производство дизайна формы, в MorphX достаточно просто. Правила хорошего тона программирования [Best practice] рекомендуют не помещать код в дизайн и по возможности авто позиционировать поле. Это ускоряет разработку. После создания запросной части формы (источника данных), вы можете формировать выход дизайна, перетаскивая группы полей из источника данных в дизайн.

Бывают случаи, когда вам необходимо помещать код в дизайн. И производить некоторые базовые настройки в свойствах поля дизайна. Но это минимальные настройки.

Авто позиционирование поля в дизайне будет лучшим решением по нескольким причинам. При добавлении поля или группы полей между уже расположенными полями в дизайне будет соответствующее расстояние. Если поле удаляется или добавляется в группу полей в таблице, то связанная группа полей в дизайне автоматически обновляется и авто позиционируется. Установка свойства поля невидимым, также авто позиционирует соседние поля. Если поля из группы полей не показываются, то и вся группа также не будет показана.

Используя авто настройки, становится трудно разрабатывать специфический дизайн для формы. Однако вы можете отключить любую авто настройку и позиционировать каждое поле в дизайне вручную. Идея авто настройки – это скорость и простота разработки дизайна формы, а так же стандартный вид форм.

### Создание дизайна

Необходимо следовать некоторым базовым правилам для представления и проектирования вашей формы. Стандартная форма показана на **Рисунке 23:**



**Стандартная форма.** Поля формы следует разделять на закладки [tab pages], где первая закладка обычно называется Обзор (Overview), которая представляет список доступных полей. Поля, показываемые на первой закладке, должны быть определены в группе полей таблицы. Если свойство поля установлено в обязательное заполнение [mandatory], то это обычно означает, что его следует располагать на первой закладке. На остальных закладках оставшиеся поля из источника данных следует группировать и давать логически объяснимые названия. Поля, не подходящие по смыслу к отдельной закладке, обычно располагают на второй закладке с названием Общие [General]. Поля следует, по возможности, добавлять в группах. Обычная практика это добавление полей, показываемых как на первой закладке, так и на других. Первая закладка часто используется, как список, наиболее важных полей таблицы, а эти поля обычно могут быть членами другой группы, поэтому эти поля расположены в дизайне дважды.

Даже если вы создаете простую форму с несколькими полями, вам следует придерживаться этой практики. Вы сэкономите немного времени, добавляя просто одно поле в форму, для показа его из источника данных. Однако это приведет к тому, что ваша форма будет выглядеть отличной от других, и не будет попадать в стандарт расположения элементов на форме, что будет затруднительно для дальнейшей модификации.

Кнопки [buttons] обычно располагаются с правой стороны формы.

Предпочтительно использовать пункт меню [menu items], так как для пункта меню становится возможным добавление ключей доступа и конфигурационных ключей. За подробной информацией по созданию пункта меню смотрите главу **Меню и Пункт Меню**.

Item number	Item name	Search name	Item group	Item type
B-R14	Battery Baby R14	BatteryBabyR14	Parts	Item
B-R6	Battery Penlight R6	BatteryPenlightR6	Parts	Item
ESB-005	Energy Saving Bulb 5 Watt	EnergySavingBulb5Wat	Bulbs	Item
ESB-007	Energy Saving Bulb 7 Watt	EnergySavingBulb7Wat	Bulbs	Item
ESB-009	Energy Saving Bulb 9 Watt	EnergySavingBulb9Wat	Bulbs	Item
ESB-011	Energy Saving Bulb 11 Watt	EnergySavingBulb11Wa	Bulbs	Item
ESB-013	Energy Saving Bulb 13 Watt	EnergySavingBulb13Wa	Bulbs	Item
ESB-015	Energy Saving Bulb 15 Watt	EnergySavingBulb15Wa	Bulbs	Item
FL-Penlight	Flash Light Penlight	FlashLightPenlight	Parts	Item
FL-Standard	Flash Light Standard	FlashLightStandard	Parts	Item
FLL-2500	Floor Lamp 2500 Color	Floorlamp2500Color	Lamps	BOM
FLL-MeasureConfig	Floor Lamp w MeasurementConfig	FLL-MeasureConfig	Lamps	BOM
FT-018	Fluorescent Tube 18 Watt	FluorescentTube18Wat	Bulbs	Item
FT-036	Fluorescent Tube 36 Watt	FluorescentTube36Wat	Bulbs	Item

Рисунок 23: Стандартная форма

Дизайн формы создается под узлом *Designs*. Не смущайтесь названием узла, так как форма может иметь только один дизайн.

Вы можете просмотреть дизайн формы двойным щелчком по узлу design. Это приведет к открытию редактора правки дизайна, как показано на **Рисунке 24: Режим правки Формы**. Режим правки формы [edit mode] предназначен для создания и правки дизайна формы. Однако вам следует рассматривать эту графическую презентацию только для обзора дизайна, так как этот инструмент, достаточно, общий и не гибкий. Используйте узлы формы вместо использования этого редактора для создания и изменения дизайна.

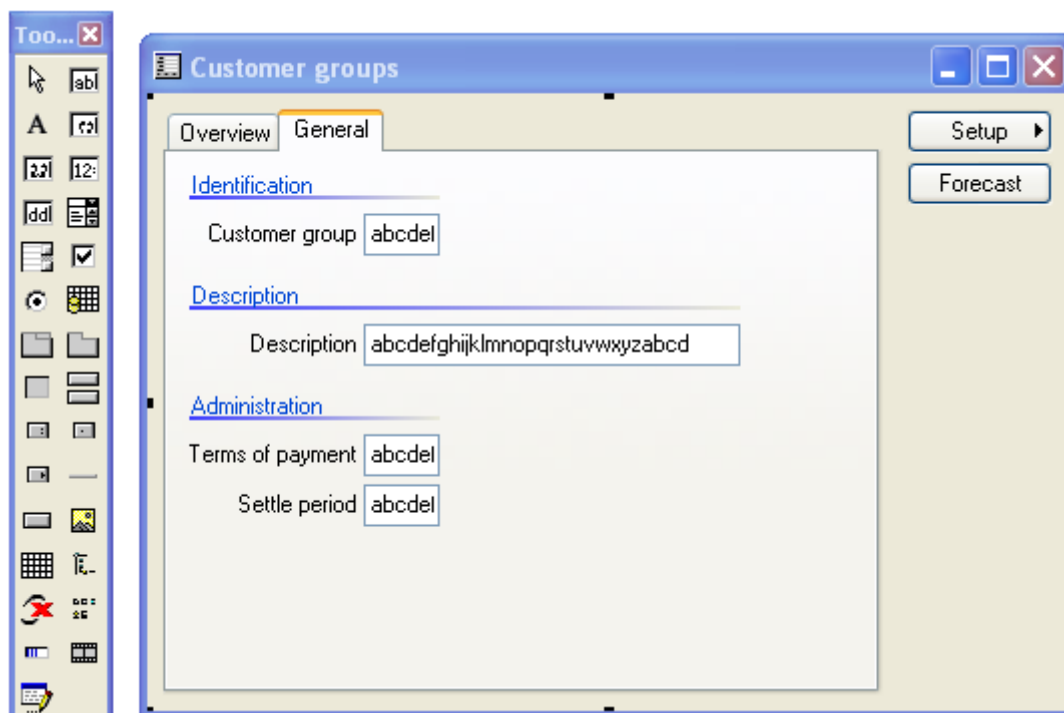


Рисунок 24: Режим правки формы

При открытии дизайна формы в режиме правки открывается панель инструментов рядом с вашей формой. Вы можете использовать панель инструментов для добавления поля, выбором нужной опции в панели инструментов и перетягивая её в дизайн. Для удаления поля формы из дизайна, выберите поле и нажмите кнопку delete.

Просмотр дизайна формы в режиме правки очень необходим для получения информации о конкретных частях дизайна формы, так как вы можете передвигаться по дизайну и просматривать все поля вне зависимости от условий видимости. Это ценно для просмотра форм с большим количеством полей, например, SalesTable. При выборе поля в дизайне, лист свойств активируется для выбранного поля. Если вы знакомы с полями формы, то это может быть быстрый путь просмотра свойств определенного поля.

## Поля в дизайне

Создание дизайна завершается добавлением полей в него, перетаскивая поля и группы полей из источника данных в дизайн или создавая поля вручную. При перетаскивании поля из источника данных, автоматически создаётся поле в дизайне соответствующего типа с установленными свойствами источника данных. Это самый быстрый путь для построения вашего дизайна и вы можете быть уверены в том, что ваши поля правильно созданы. Поля можно создавать и вручную. Это обычно делается при создании полей, которые не связаны с источником данных. Отметьте, что поля и группы полей не могут быть перемещены напрямую из таблицы, как в дизайне отчета.

Вам следует использовать описательное название полей. Создавая поле вручную, вы добавляете его в дизайн с надлежащим типом и последовательным номером. Меняйте дизайн, где поля созданы с именами по умолчанию, так как это делает дизайн трудным в понимании и вам необходимо раскрывать все узлы для просмотра. Посмотрите на **Рисунок 25: Плохое наименование полей**. А ведь это совсем простая форма.

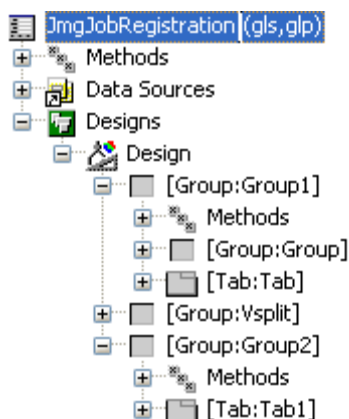


Рисунок 25: Плохое наименование полей

Метки, используемые в полях и группах полей дизайна по умолчанию не определяются, а имя подставляется из метки расширенного типа данных, базового перечисления [base enums] или поля таблицы. Вам только остается определить метки для полей, содержащих другие поля формы такие, как группы полей, закладки или поля табличного документа. Если вам необходимо другое название поля или группы полей, то вы можете переопределить название по умолчанию (берется из свойства поля таблицы или свойства расширенного типа данных), вставив текст, в свойство label. Это не рекомендуется, так как вы теряете автонастройку, и это потребует большего времени на настройку вашей формы.

Для обзора доступных полей дизайна формы, смотрите **Рисунок 26: Поля формы**.

Название	Описание
ActiveX	Используется для интеграции поля ActiveX в форму. Смотрите форму KMKnowledgeAnalogMeter.
Animate	Используется для проигрывания клипов.
Button	Простая кнопка. Используйте MenuItemButton по возможности, так как затем вам потребуется переопределять метод clicked() на кнопке для выполнения задания.
ButtonGroup	Поле группы кнопок, используется для группировки любых типов кнопок.
CheckBox	Возвращает значение true или false. Часто используется для установки значения перечислимого типа NoYes.
ComboBox	Поле выбора, используется для показа введенного значения. Показывает вводимые значения.
CommandButton	Специальный тип кнопки. Это поле имеет свойство Command, где определяется возможное действие. Обычно используется для добавления кнопок Ok, Cancel и Apply в форму. Смотрите форму ReqTransPoCreate.
DateEdit	Используется для ввода даты. Дата форматируются согласно региональной настройке Windows.
Grid	Используется для представления данных в таблице на форме. Поле табличного документа обычно добавляется на первую закладку. Свойства Width and Height следует всегда устанавливать в Column width и Column Height, делая возможным добавление бегунка в grid при изменении размеров формы.
Group	Поле группы обычно используется для показа групп полей таблицы. Поле group может так же использоваться для включения других group для представления полей с определенным количеством колонок.
HTML	Может использоваться для вывода содержимого HTML в форму, смотрите форму ForecastItemAllocationDefaultDataWizard.
IntEdit	Используется для целочисленного значения.
ListBox	Простой вывод поля списка ListView. Может использоваться для перечислений. Это поле редко используется.
ListView	Часто используется при выборе фиксированного числа значений, где одно поле ListView содержит все возможные значения, а другое поле ListView содержит

Название	Описание
	выбираемые значения. Данные должны вручную заполняться в ListView. Смотрите форму KMActionType.
MenuButton	Меню кнопок MenuButton используется для организации кнопок в под меню. Код на изменение доступа к кнопкам в подменю часто располагается в методе clicked() у MenuButton.
MenuItemButton	Используется для создания кнопки пункта меню. MenuItemButton – это предпочтительный тип кнопки, так как не используется код для выполнения связанного пункта меню. Свойства MenuItemType MenuItemName определяют объект для выполнения. Обычно создается перетаскиванием пункта меню в дизайн формы.
Progress	Обычно форма прогресса создается с использованием класса приложения SysOperationProgress. Это поле может использоваться для интеграции формы прогресса в форме. Используйте метод pos() этого progress поля для обновления счетчика.
RadioButton	Альтернатива полю ComboBox для вывода enums. Каждое значение enum будет показываться как отдельно выбираемое значение.
RealEdit	Используется для значений типа real.
Separator	Это поле используется в поле menuButton. Будет разделять кнопки в подменю добавлением горизонтальной черты.
StaticEdit	Показывает не редактируемый текст. Это поле может быть связано с полем источника данных. Обычно, текст этого поля устанавливается в листе свойств или в X++. Отметьте, что это поле не может использоваться в поле табличного документа.
StringEdit	Используется для строковых значений. Если использовать тип мемо, то свойство Width и Height следует устанавливать в Column width и Column height.
Tab	Это обычно самый верхний уровень из полей стандартного дизайна формы. Это поле используется для добавления на него закладок tab pages. Свойства Width и Height всегда следует устанавливать в Column width и Column Height, делая возможным изменение размера закладок (tab pages) на поле tab.
Table	Не используется. Может показывать множество записей, как и поле табличного документа. Данные должны вноситься вручную.
TabPage	Закладки добавляются на поле tab. На закладке возможно располагать любой тип поля, содержащий данные. Дизайн

Название	Описание
	формы обычно состоит из одной или более закладок.
TimeEdit	Это поле использует вывод времени. Время форматируется согласно региональным настройкам Windows.
Tree	Используется для вывода записей в виде дерева. Логика построения дерева располагается в классе приложения. Смори форму ProjTable.
Window	Может использоваться при добавлении bitmap к форме. Смотрите форму KMAction.

Рисунок 26: Поля формы

### Поле табличного документа

Поле табличного документа [grid] имеет несколько возможностей, которые полезно знать, так как это делает вашу модификацию более наглядной. Вы можете пометить некоторые записи в нем так же, как пометить файлы в файловой системе Windows. Это очень наглядно, например, вам необходимо сделать некоторую комбинацию с записями, вызвать исполнимый класс и обновить только отмеченные записи. Поле типа галки [checkbox] часто используется в табличном поле для контроля, какие записи отмечены. Если вам нет необходимости сохранять информацию о помеченных записях, то используйте edit метод.

---

### Пример 4: Множественная выборка

#### Элементы проекта MORPHXIT Forms

- Form, MyForm\_MultiMark
- Menu item display. MyForm\_MultiMark

Этот пример показывает, как получить выбранные записи в табличном документе.

1. Продублируйте форму MyForm, и переименуйте форму в “MyForm\_MultiMark”.
2. Перейдите к узлу *Designs/Design*, через контекстное меню выберите *Создать Control/ButtonGroup*.
3. Добавьте кнопку в ButtonGroup через контекстное меню на поле *ButtonGroup* и выберите *Создать Control/Button*. Перейдите к листу свойств новой кнопки и установите метку “MultiMarkTestButton” в свойстве **Name**. Установите свойство **MultiSelect** в Yes. В свойстве **Text** введите “Show marked”.

4. Теперь переопределите метод clicked() на кнопке [ButtonGroup:ButtonGroup]/Button:MultiMarkTextButton/Methods и добавьте следующий код:

```
void clicked()
{
    salesTable salesTableMarked;
;
    super();

    if (salesTable_ds.anyMarked())
    {
        salesTableMarked = salesTable_ds.getFirst(1,false);

        while (salesTableMarked)
        {
            info(salesTableMarked.salesId);
            salesTableMarked = salesTable_ds.getNext();
        }
    }
}
```

5. Сохраните форму и создайте пункт меню.
- 

Откройте новую форму MyForm\_MultiMark. Отметьте несколько записей в табличном поле и нажмите кнопку «Show marked». Номера заказов [sales id] выбранных записей будут показаны в Infolog. В реальном случае вам следует использовать кнопку, активирующую пункт меню и вызвать исполнимый класс, а затем производить уже классом обработку записей. Только кнопки с свойством MultiSelect, установленным в Yes могут использоваться при обработке более чем одной записи. Это здорово, так как нет необходимости в проверке, вызван ли пункт меню с одной записью или нет. Форма SalesTable в стандартной поставке использует множественную отметку для разности.

---

**Примечание:** Табличное поле показывает только часть записей доступных в форме. Это означает, что позиция вертикального бегунка в табличном поле основана на числе заэкшированных записей. Это сделано для лучшего представления.

---

Еще одна прекрасная возможность использования табличного поля - это выделение записи, копирование и вставка прямо в Microsoft Excel. Однако только поля связанные с источником данных могут быть скопированы таким образом.

Как одно поле, так и множество записей могут быть раскрашены в табличном поле. Использование цвета делает вашу форму более наглядной, так что это может использоваться для просмотра определенных значений определенных полей. Вы можете найти примеры использования цвета на форме в разделе **Специальные Формы**.

## Поле дерева

Если записи в вашем источнике данных связаны иерархически, то вам следует рассмотреть использование поля дерева. Это позволит пользователю иметь логическое представление записей. Такие формы как ProjTable и HRMOrganization используют поле дерева [Tree].

Просмотр данных в дереве – это небольшой трюк и вам следует добавить немного кода, чтобы это заработало. Просто продублируйте существующую форму, использующую поле дерева вместо попытки построения собственной с нуля. Используйте класс для построения данных в поле дерева. Вы можете найти две версии этого класса в стандартной поставке. Простейший класс называется FormTreeDatasource и используется в форме ProjTable. Форма HRMOrganization использует более продвинутый класс с названием CCFormTreeDatasource. Продвинутая версия поддерживает, помимо прочего, поддерживает drag and drop.

Форма, использующая поле дерева, как правило, большая и при изменении размеров формы, поле дерева может некорректно отображаться. Это можно разрешить добавлением разделителя [‘splitter’] между полем дерева и другими полями формы. Далее уже разделителем вы можете изменять размеры либо поля дерева или других полей формы. Разделитель это не обычное поле и для построения используется класс. Если вы хотите использовать разделитель, скопируйте код из формы уже использующей разделитель. **Рисунок 27: Поле разделитель** показывает, где добавлено поле разделитель в форме HRMOrganization. Вы используете группы полей при создании разделителя, а также перекрываете методы mouseUp(), mouseMove() и mouseDown(). Проверьте свойства группы полей SplitControl, так как вам необходимо будет изменить некоторые из них.

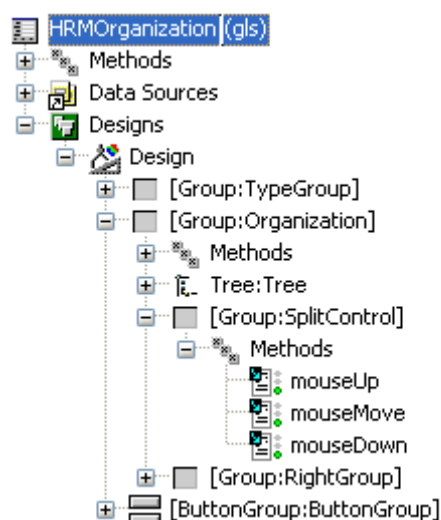


Рисунок 27: Поле разделитель



## Модификаторы Display и Edit

MorphX имеет два специальных модификатора для методов, используемых в интерфейсе форм. Они применяются если вам необходимо показывать значения полей, которые не являются частью запроса (нет в источнике данных). У вас может быть задача, которая потребует построения комплексного запроса, объединяющего с определенным полем связанной таблицы или с полем таблицы, с которой нет вообще никакой связи. Для получения такого результата самым простым будет использование либо модификатора display, либо edit.

Эти два типа модификаторов обычно используются в display и edit методах. Display и edit методы могут возвращать любой тип значений. Модификаторы display и edit просто означают, что они будут использованы в пользовательском интерфейсе.

Перед выбором использования display или edit методов вам следует знать, что вы не сможете сортировать в поле табличного документа, используя их. Щелкая по полю в табличном документе, вы сортируете по этому полю записи в табличном документе, и этот функционал необходим пользователю. Пользователь не знает, связано ли поле с источником данных или нет, или основано на использовании display или edit методов.

---

### Пример 5: Методы display и edit

#### Элементы проекта MORPHXIT Forms

- Table, SalesTable
- Form, MyForm\_DisplayEdit
- Menu item display, MyForm\_DisplayEdit

Этот пример покажет вам использование модификаторов display и edit в интерфейсе пользователя. Модифицируем форму MyForm для использования полей с модификаторами display и edit.

1. Продублируйте форму MyForm и переименуйте в “MyForm\_DisplayEdit”.
2. Откройте еще одно окно AOT и перейдите в методы таблицы SalesTable.
3. Перетащите метод customerName() в группу полей Customer.
4. Создайте новый метод в SalesTable со следующим кодом:

```
display LineAmount salesTotal(SalesTable _salesTable)
{
    return (select sum(lineAmount) from salesLine
            where salesLine.salesId == _salesTable.salesId).lineAmount;
}
```

5. Убедитесь, что изменения сохранены в SalesTable и перетащите метод salesTotal() в форму MyFrom\_DisplayEdit в дизайн *Designs/Design/[Tab:Tab]/[TabPage:Overview]/[Grid:Grid]*. Расположите поле последним в табличном документе. Откройте лист свойств нового поля и выберите SalesTable как значение свойства **DataSource**.
6. Перейдите к таблице SalesTable и добавьте новый метод со следующим кодом:

```
edit CustName editCustomerName(boolean _set,
                                CustName _name)
{
    CustName    name = _name;
    CustTable    custTable;

    if (_set)
    {
        if (name)
        {
            ttsbegin;
            custTable      = CustTable::find(this.custAccount, true);
            custTable.name  = name;
            custTable.update();
            ttscommit;
        }
    }
    else
    {
        name = CustTable::find(this.custAccount).name;
    }

    return name;
}
```

7. Сохраните изменения в SalesTable и перетащите новый метод в дизайн формы *Designs/Design/[Tab:Tab]/[TabPage:Overview]/[Grid:Grid]*. Расположите новое поле в поле табличного документа после поля SalesTable\_CustAccount. Откройте лист свойств нового поля и выберите SalesTable, как DataSource для этого поля.
8. Перейдите к узлу *Data Sources/SalesTable/Methods* и переопределите метод init() следующим кодом:

```
public void init()
{
    super();

    this.cacheAddMethod(tablemethodstr(SalesTable, salesTotal));
}
```

9. Сохраните форму и создайте новый пункт меню для этой формы.

Два различных display метода были добавлены в форму. Метод `customerName()` - это display метод, возвращающий название покупателя текущего заказа. И display методы и edit методы могут добавляться в группу полей. Это прекрасная возможность, так как в этом случае вам не придется менять форму для вывода названия покупателя. Использование display и edit methods в табличных группах полей имеет один недостаток. При добавлении или удалении табличного метода, id метода меняется. Это означает, что вы должны проверить вашу табличную группу полей, содержащую display и edit методы, каждый раз при изменении номера табличного метода, так как ваша группа полей будет содержать неправильный display или edit метод.

---

**Примечание:** Если ваш display метод выводит данные в форме без использования метки, вы можете забыть про установку источника данных вашего поля. При перетаскивании display или edit метода из методов таблицы это необходимо сделать вручную.

---

Во-вторых, display метод, используемый в форме `salesTotals()` делает простое суммирование итогов выбранных заказов. При использовании display метода в поле табличного документа, вам необходимо использовать текущую запись в качестве параметра. При передаче текущей записи параметром, display метод может быть создан как табличный метод или как метод на источнике данных.

Display и edit методы вызывают задержку производительности, особенно при использовании в поле табличного документа, так как display методы выполняются несколько раз, например, при прокрутке записей формы. Для оптимизации представления display методы следует кэшировать. Перекрыв метод `init()` источника данных методом `cacheAddMethod()` можно кэшировать табличные display методы. Только табличные методы можно кэшировать и кэширование необходимо делать после `super()` в методе `init()` источника данных. Display методы обычно создаются как табличные методы и это не вызывает никаких проблем. Edit методы не могут кэшироваться, так как edit методы не такие простые, как display методы, итак кэшированием display методов можно улучшить производительность.

В нашем примере был за кэширован только display метод поля табличного документа. В этом случае будет не большой выигрыш (кэширование display метода, показывающего имя покупателя) так как этот результат будет выведен для всех записей.

Edit метод - это расширение функционала display метода. Display метод обычно применяется для вывода дополнительной информации или результата, произведенных вычислений, edit метод применяется где объединение не может быть использовано и значение в связанной таблице может правиться. Edit methods имеет два параметра, первый – true, если значение поля изменено, а второй параметр содержит значение поля. Вам следует добавлять код в edit метод для получения значения в связанной таблице при обновлении. Вы, может быть,

догадались, что это стоит дополнительных ресурсов и вам всегда следует учитывать, насколько будет меняться ваш дизайн перед применением `edit` метода.

Расширенный тип данных, используемый в возвращаемом значении `display` или `edit` метода, устанавливает метку и формат поля. Если вам необходимо другое название поля, то вам следует рассмотреть создание собственного расширенного типа данных для применения в вашем методе, это лучше, чем изменять метки в дизайне.

---

**Примечание:** Альтернатива `edit` методу – если только вам необходимо выводить одну запись – это просто определить расширенный тип данных и в листе свойств поля установить свойство `AutoDeclaration` в `Yes`. Таким образом, вы можете получать и выставлять значение поля формы. Форма *KMAAction* использует подобный функционал для фильтрации в верхней части формы.

---

## 6.4 Методы на форме

Базовые формы могут быть созданы без единой строчки кода. Даже сложные формы могут создаваться без написания кода в форме. В действительности, старайтесь писать код вне форм и реализуйте логику в таблицах или классах, так как это упростит пользовательский интерфейс. При использовании внешней системы и выполнении логики через СОМ вам придется переписывать код формы, так как формы не могут выполняться через СОМ.

Вы не можете избежать написания кода в форме. Просто держите в голове, при проектировании формы, что если возможно, добавлять код в другом месте. Например, вам необходимо реализовать определенные задачи после ввода значения в поле формы. Если поле связано с источником данных, то будет лучше добавить код в табличный метод `modifiedField()`, чем в метод поля источника данных формы `modified()`.

В следующей секции вы найдете обзор методов, которые могут быть перекрыты в форме.

### Методы формы

При выполнении формы иницируется сущность системного класса `FormRun`. В узле *MyForm/Methods* вы можете перекрыть методы объекта `FormRun`. Обычно на эти методы ссылаются как на методы формы. Через контекстное меню узла метода формы выберите *Перекрыть Method*, все методы формы, класса `FormRun` будут показаны списком. Будут показаны только методы, которые можно перекрыть, а методы, определенные, как `final`, вообще не будут показаны в списке.

На методы формы можно ссылаться, используя ключевые слова *this* и *element*. Ключевое слово *this* может использоваться только для ссылки на методы формы, а

element используется как ссылка на форму, как объект и может быть использовано при ссылке, как из методов источника данных, так и полей формы.

Имя	Параметры	Описание
activate		Вызывается каждый раз при фокусировании на форму. Если переключаться между формами метод формы будет вызываться при выборе активной формы. Смотрите форму PBATable.
canClose		Вызывается после метода closeCancel() или closeOk(). Часто используется для проверки возможности закрытия формы. Смотрите форму InventTransPick.
close		Это последний метод, выполняемый при закрытии формы. Смотрите форму ProdParameters.
closeCancel		При нажатии кнопки типа CommandButton с установленным свойством Command в Cancel, вызывается метод closeCancel(). Если closeCancel() вернул true, вызывается метод canClose(). Смотрите форму AssetChangeGroup.
closed		Closed() возвращает true после super() вызванного в close(). Может использоваться для проверки вызова close(). Не часто используется.
closedCancel		Возвращает true после вызова closeCancel(). Часто используется из close() для проверки закрытия формы cancel или ok. Смотрите форму CustParameters.
closedOk		Возвращает true после вызова closeOk(). Возвращаемое значение может использоваться для проверки возможности выполнения метода источника данных write(). Смотрите форму SalesCreateOrder.
closeOk		При нажатии кнопки типа CommandButton с установленным свойством Command в ОК, вызывается метод closeOk(). Если closeOk() вернул true, вызывается метод canClose(). Смотрите форму AssetChangeGroup.

closeSelect	str_selectString	Используется для определения возвращаемого значения в форме выпадающего списка [lookup form]. Часто в этом нет необходимости. Вместо использования метода перед закрытием формы выпадающего списка. Метод selectMode() класса FormRun устанавливает форму в режим просмотра [lookup mode] и определяет возвращаемое значение выбранного поля формы. Смотрите форму KMGamePlanLookup.
controlMethodOverloadObject	Object_val	Переопределяя этот метод, можно перекрывать методы полей, добавляемых на форму программно во время выполнения и создавать их в классе, а не в форме. Смотрите класс BOMChangeItem.
copy		Выполняется при нажатии пользователем ctrl+c для копирования содержимого поля формы или целой записи.
cut		Выполняется при нажатии пользователем ctrl+x для удаления содержимого поля формы.
doApply		При нажатии кнопки типа CommandButton с установленным свойством Command в Apply, вызывается метод doApply(). Может использоваться для подтверждения изменений, сделанных на форме. Смотрите форму ProdSetupRelease.
docCursor		Так как на сложной форме активная запись связана с несколькими источниками данных, то метод docCursor() используется для определения какой источник данных связать с активной записью, если форма использует более, чем один источник данных. Смотрите форму InventTable.
Finalize		Вызывается, когда форма уже закрыта. Удаляет объект формы из памяти. Этот метод обычно не переопределяется.
firstField	int_flags = 1	Устанавливает фокус на первое поле в дизайне формы. Переопределяется, если необходимо устанавливать фокус на другом поле формы. Смотрите форму CustOpenTrans.

init		Это первый метод, который можно переопределить. Этот метод инициализирует форму. Сущности, используемые во всей форме, обычно инициализируются здесь. Проверка объекта, вызывающего форму, например, при вызове из другой формой используется в init(). Смотрите форму CustOpenBalance.
lastField	int _flags = 1	Этот метод не имеет эффекта. Должен устанавливать фокус на последнее поле в дизайне формы.
loadUserSettings		Настройки, сделанные пользователем, в форме загружаются, используя loadUserSettings().
new	Args _args = NULL	New() – первый вызываемый метод при запуске формы. Этот метод не следует никогда переопределять, так как это может привести к проблемам при открытии.
nextField		Выполняется при переходе на следующее поле формы, либо используя tab, либо enter.
nextGroup		Выполняется при переходе на следующую группу полей, нажатием ctrl+pgdn.
paste		Выполняется, когда пользователь нажимает ctrl+v для ввода значения в поле формы.
prevField		Выполняется при переходе на предыдущее поле, нажатием shift+tab.
prevGroup		Выполняется при переходе на предыдущую группу полей, нажатием ctrl+pgup.
print		Выполняется при распечатке формы. Выводятся на печать поля из группы полей AutoReport таблицы. Смотрите форму ReqItemJournalSafetyStock.
printPreview		PrintPreview() вызывается, если форма выбрана из верхнего меню Файл   Предварительный просмотр. Смотрите форму SysLicenseAgreement.
reload	Args _args = NULL	Не используется.

resize	int _width, int _height	Если размеры формы изменены, то вызывается метод <code>resize()</code> .
run		<code>Run()</code> загружает форму. Вызываемый в этом методе <code>super()</code> запускает <code>run</code> запроса источника данных формы [ <code>data source query</code> ]. Инициализацию формы следует производить по возможности, в методе <code>init()</code> или в методе источника данных <code>init()</code> , это лучше чем переопределение метода <code>run()</code> . Пример использования смотрите в форме <code>ProdTable</code> .
saveUserSettings		Пользовательские настройки формы сохраняются в <code>saveUserSettings</code> .
selectControl	FormControl _control	Вызывается при переходе фокуса на следующее поле. Смотрите форму <code>LedgerTransSettlement</code> .
send		<code>Send()</code> вызывается если выбирается Файл   Послать в верхнем меню.
setApply	Object _object, Object _parm	Может использоваться в комбинации с кнопкой <code>Command</code> , если форма вызывается из класса. Смотрите класс <code>DocuActionTrans</code> .
task		Вызывается, когда пользователь нажимает правую кнопку мыши или <code>key</code> . Может использоваться для определения горячих клавиш [ <code>hot keys</code> ]. Смотрите форму <code>JmgSelectJob</code> .

Рисунок 28: Методы формы

## Методы источника данных формы

Методы источника данных формы часто перекрываются при необходимости внесения изменений в запрос формы или вставке, обновлении или удалении записей в источнике данных используя X++. Источник данных формы – это сущность системного класса `FormDataSource`.

Имя	Параметры	Описание
Active		<code>Active()</code> вызывается каждый раз при выборе новой записи. Этот метод часто используется для установки доступа к полям. Смотрите форму <code>SalesTable</code> .



create	boolean _append = false	При нажатии ctrl+n вызывается метод create(). Инициализацию значений полей следует производить, используя initValue(). Смотрите форму SalesTable.
cursorNotify	int _event	Вызывается при установке курсора на другую запись. Метод так же вызывается при обновлении кэша источника данных. Может использоваться для продвинутого кэширования. Смотрите форму projPeriodEmpl.
defaultMark	int _value	Возвратит значение 1 если все записи в поле табличного документа [grid] отмечены нажатием ctrl+a. Смотрите класс ReqTransFormPO.
Delete		Если пользователь нажал alt+F9, вызывается метод delete(). Delete() вызовет validateDelete() перед попыткой удаления записи. Смотрите форму ProjJournalTable.
deleteMarked		Метод вызывается, если пользователь пометил более чем одну запись и нажал alt+F9. DeleteMarked() вызовет delete() для каждой удаляемой записи.
displayOption		Используется для подкрашивания колонок или рядов. Смотрите форму EmplTable.
executeQuery		Выводит данные, основанные на настройках запроса формы. Метод часто переопределяется для установки значения добавляемых критериев из X++. Смотрите форму CustTrans.
filter	fieldId _field, str _value	Если кликнуть на поле и выбрать в контекстном меню Фильтр, вызовется метод executeQuery(). Не часто переопределяется, как метод executeQuery().
findRecord	Common _record	Может быть использован для установки курсора на определенной записи, используя объект Common. Отметьте, что это медленный путь для позиционирования курсора. Смотрите форму SalesTable.

findValue	fieldId _field, str _value	Может использоваться для установки курсора на запись, соответствующего значения определенного поля. Это очень медленный путь позиционирования курсора. Смотрите форму LedgerJournalTransApprove.
first		Вызывается при нажатии ctrl+home для выбора первой записи в источнике данных. Смотрите форму ContactPerson.
forceWrite	boolean _value	Помечает запись как “dirty”. Если установить true для текущей записи, то выполнится сохранение перед переходом на другую запись. Должен вызываться из X++. Смотрите форму LedgerJournalTransApprove.
init		Init() – это первый вызываемый метод источника данных. Изменения запроса формы во время выполнения добавляются обычно здесь. Смотрите форму CustTrans.
initValue		При создании новой записи в форме будет вызван метод initValue(). Этот метод используется для инициализации значениями полей. Используйте вместо этого корреспондирующий метод на таблице, так как настройки специфичны для формы. Смотрите форму BankAccountStatement.
last		Вызывается при нажатии ctrl+end для выбора последней записи в источнике данных. Смотрите форму HRMApplclicantTable.
leave		Метод вызывается при переходе курсора из текущей выбранной записи.
leaveRecord		Может вызываться из X++ для проверки возможности передвижения курсора на другую запись. Смотрите форму InventPosting.
linkActive		При вызове связанной формы[sub form], вызывается linkActive() каждый раз при переходе на новую

		запись в главной форме. LinkActive() вызывает executeQuery(). Смотрите форму MarkupTrans.
mark	int _value	Возвратит 1, если отмечена текущая запись. Может использоваться для отметки записи при вызове метода со значением 1.
next		Вызывается при выборе следующей записи. Смотрите форму EmplTable.
nextPage	int _pageSize	При нажатии пользователем кнопки page down в поле табличного документа, вызывается nextPage(). Параметр _pageSize содержит число показанных записей в поле табличного документа.
prev		Вызывается при выборе предыдущей записи. Смотрите форму EmplTable.
prevPage	int _pageSize	При нажатии пользователем кнопки up key в поле табличного документа, вызывается prevPage(). Параметр _pageSize содержит число показываемых записей в поле табличного документа.
print	PrintMedium _target	Вызывается при нажатии ctrl+p для печати авто отчета. Может использоваться для изменения поведения печати авто отчета, как, например, изменения вывода во время выполнения. Смотрите форму CCCount.
prompt		Исполняется, когда пользователь вызывает форму диалога запроса. Вызывается только для текущего источника данных. Prompt() может использоваться для защиты вызова диалога запроса при использовании формой поля дерево. Смотрите форму HRMOrganization.
refresh		Часто используется вместе с reread(). Где reread() обновляет источник данных формы для текущей записи со значениями из таблицы, refresh() обновляет форму дизайн формы для текущей записи значениями из источника данных формы. Смотрите форму WMSShipment.

refreshEx	anytype _pos	Расширенная версия refresh(). Если метод вызывается с параметром -1, то все записи источника данных обновляются. Смотрите форму CustOpenTransReverse.
removeFilter		Вызывается при удалении предыдущего фильтра. Обычно переопределяется на источнике данных, который только содержит одну запись в качестве параметра формы. Смотри форму InventParameters.
reread		Должен вызываться из X++. Выводит текущую запись из таблицы источника данных. Смотри форму SalesCreateOrderForm.
research		Должен вызываться из X++. Выводит записи, определенные в запросе. Метод похож на вызов executeQuery(), кроме того, что research() содержит настройки, сделанные в запросе. Смотри форму LedgerTable.
validateDelete		Delete() вызывает validateDelete() для проверки удаления записи. Смотри форму ProdJournalTable.
validateWrite		ValidateWrite вызывается методом write(). Возвращает true, если запись может быть вставлена или обновлена. Смотри форму InventTable.
write		Вызывается при нажатии ctrl+s. Вызывается insert() при создании новой записи, иначе вызывается update(). Проверку лучше производить в validateWrite(). Смотри форму CustOpenTrans.

Рисунок 29: Методы источника данных формы

## Методы полей источника данных

Все поля источника данных имеют определенный набор методов определенных в системном классе FormDataObject.

Имя	Параметры	Описание
context		Вызывается при нажатии

		пользователем правой кнопки мыши на поле.
filter	str _filterStr, NoYes _clearPrev	Вызывается при нажатии пользователем правой кнопки мыши на поле и выбором Фильтр. Отметьте, что super() вызванный в filter() не работает, если метод перекрывается.
find		Вызывается при нажатии пользователем правой кнопки мыши на поле и выбором Найти.
helpField		Выводит справочный текст текущего поля на экране, при наведении на него курсора. Справочный текст показывается для текущего поля, связанного с полем таблицы или EDT или Base Enum.
jumpRef		Вызывается ctrl+alt+F4 или при переходе к основной таблице контекстного меню. Метод может быть переопределен для перехода на другую форму, с которой не существует связи. Смотри форму InventItemGroup.
lookup	FormControl _formControl, str _filterStr	Используется для построения собственной формы выпадающего списка для поля, у которого нет такой формы. Смотри форму InventTable.
modified		Если значение поля, связанного с полем источником данных изменилось, вызывается modified(). Если validate() возвращает true, вызывается modified(). Обычно переопределяется для установки значения другого поля. Смотри форму PriceDiscAdm.
performFormLookup	FormRun _form, FormControl _formControl	Может использоваться для наложения критериев на существующую форму выпадающего. Смотрите форму HRMApplication.
restore		Это метод, по-видимому, не используется.
toolTip		Используется для подсвечивания желтым цветом названия при наведении курсора на поле. Метод

		может быть переопределен для изменения подсвечивания по умолчанию. Смотри форму BOMDesigner.
validate		Этот метод вызывается при изменении значения в поле, связанном с источником данных. Используется для добавления проверки измененного значения. Смотри форму InventTable.

Рисунок 30: Методы полей источника данных

Специфические проверки для одной формы делаются с использованием методов поля источника данных формы. Если делается проверка по полю, связанному с источником данных, то вам следует использовать табличные методы `validateField()` и `modifiedField()`, чем использовать методы полей источника данных `validate()` и `modified()`.

## Методы полей формы

Некоторые методы полей формы похожи на методы полей источника данных. У вас есть альтернатива перекрытию метода на поле формы - это использование метода источника данных и объявление поля формы [auto declare]. Объявление поля делается установкой свойства поля **AutoDeclaration** в Yes. Это общее свойство для всех полей формы. Объявлением поля формы создается сущность системного класса, ссылающегося на поле формы. Объявлением поля типа строка создается сущность системного класса `FormStringControl` и теперь оно становится доступным из всех методов формы.

## Последовательность выполнения методов формы

Только несколько методов формы необходимы в повседневной работе. Имея базовые знания по последовательности выполнения этих методов, вам легче будет разрабатывать свои модификации. Следующие методы выполняются в определенной последовательности при открытии и закрытии формы:

`init() ► ds init() ► run() ► ds executeQuery() ► canClose() ► close()`

1. `FormRun.init()` – это первый вызываемый метод. `Super()` вызываемый в `FormRun.init()` вызывает `FormDataSource.init()` каждого источника данных, используемого в запросе.
2. `Super()` вызываемый в `FormRun.run()` вызывает `FormDataSource.executeQuery()` каждого источника данных.
3. При закрытии формы `FormRun.canClose()` выполняется проверка, может ли форма быть закрыта и если возвращается true, вызывается `FormRun.close()`.

Это наиболее важные методы, выполняемые при открытии и закрытии формы. Другие методы, выполняемые в последовательности открытия и закрытия, загружают настройки, сделанные пользователем. Однако обычно вам придется переопределять только упомянутые методы.

---

**Примечание:** Для обследования порядка выполнения методов, вы можете установить точку останова при вызове `super`, вызываемого в перекрытом методе и проверить стек вызовов в отладчике. Другая опция выводить строку в Infolog из каждого перекрытого метода.

---

Для каждого источника данных в запросе формы, будет выполняться следующая последовательность отработки запроса `executeQuery()` при открытии формы:

`ds.executeQuery() ► refresh() ► active()`

1. Метод источника данных `executeQuery()` выводит все записи текущего источника данных и затем вызывает `refresh()`.
2. `Refresh()` обновляет выводимые записи, и обычно не перекрывается. Если вы вставляете, обновляете, удаляете записи в форме из X++, то `refresh()` следует вызывать для обновления выводимых данных.
3. При открытии формы, вызывается `active()` текущей записи. Вы можете использовать `active()` для установки ограничения доступа к полям активной записи. `Active()` вызывается каждый раз при переходе на новую запись.

---

#### Пример 6: Active

##### Элементы проекта MORPHXIT Forms

- Form, `MyForm_Active`
- Menu item display. `MyForm_Active`

Этот пример показывает, как работает метод источника данных `active()`, используемый для управления доступом.

1. Продублируйте `MyForm` и переименуйте новую форму в “`MyForm_Active`”.
2. Раскройте форму и перейдите к узлу *Data Sources/SalesTable/Methods*. Переопределите метод `active()`. Добавьте следующий код в метод `active()`:

```
public int active()
{
    int ret;

    ret = super();

    salesTable_ds.allowEdit(salesTable.salesStatus == SalesStatus::Invoiced ? false : true);

    return ret;
}
```

}

### 3. Сохраните форму и создайте пункт меню.

Откройте форму MyForm\_Active. Вы не сможете редактировать заказ со статусом invoiced. Если вы перейдете на вторую закладку формы, то увидите, что поля подкрашены серым цветом - это означает, что поля не могут редактироваться. Поля в табличном документе не изменили своего цвета. Если вам потребуется изменить вид полей в табличном документе, то это лучше реализовать подкрашиванием рядов. Как модифицировать вид поля в форме подробно объясняется в разделе **Специальные Формы**.

#### Выполнение методов источника данных

Добавление новой записи в форму вызывает следующую последовательность методов источника данных:

create() ► initValue() ► refresh() ► active() ► refresh()

1. Метод create() вызывается при нажатии ctrl+n для добавления новой записи.
2. Метод источника данных initValue() вызывается методом create(). Вам следует определить поля, которым следует передавать специфические значения в initValue() при инициализации. Табличный метод initValue() вызывается из super() метода источника данных initValue(). Если определяемое значение не связано со спецификой формы, то вам следует использовать корреспондирующий табличный метод.
3. После инициализации вызывается refresh(), а при вводе записи вызывается active(). Далее refresh() вызывается снова методом active().

При сохранении записи вызываются следующие методы:

validateWrite() ► write() ► refresh()

1. ValidateWrite() вызывает табличный метод validateWrite(). По возможности, используйте корреспондирующий табличный метод.
2. Метод источника данных write() вызывает или insert() или update() на таблице, в зависимости от того создается ли запись или обновляется. Проверки следует уже сделать перед этим. Если у вас есть условия заполнения, то это следует делать в validateWrite().
3. Refresh() вызывается для обновления поля формы.

Удаление записи имеет сходную последовательность действий:



validateDelete() ► delete() ► active()

1. ValidateDelete() вызывает корреспондирующий табличный метод. Условия по возможности удаления можно делать как в этом методе, так на источнике данных или на таблице.
2. Метод источника данных формы delete() вызывает табличный метод delete().
3. Active() вызывается при перестановке курсора на другую запись после удаления текущей.

### Методы полей формы

Все поля формы также имеют методы, которые можно перекрыть. Вам следует только в специальных случаях перекрывать методы на поле формы. Если поле формы не связано с источником данных, то вы можете использовать методы поля формы. Для связанных полей правилом хорошего тона программирования считается приоритетное использование методов поля источника данных. Некоторые специальные поля формы такие, как поле дерева [tree control] и поле списка [list control] требуют, что бы перекрывались определенные методы на самом поле. Для остальных полей базовых типов вам вряд ли следует использовать методы поля формы.

### Переопределение запроса формы

Вы часто, наверное, сталкивались с ситуацией, которая требует фильтрацию данных в форме. Так как форма использует запрос для вывода данных, то этого можно достигнуть добавлением критерия в запрос формы. При использовании запроса в АОТ или запроса отчета вы можете добавлять критерии в запрос, используя соответствующие узлы в этих объектах. Запрос формы не имеет подобных узлов для фильтрации данных. Вместо этого вам следует использовать X++ для добавления критериев.

---

#### Пример 7: Добавление критериев

##### Элементы проекта MORPHXIT\_Forms

- Form, MyForm\_QueryFilter
- Menu item display. MyForm\_QueryFilter

Этот пример показывает, как добавить фильтр в запрос формы, используя поле на форме.

1. Сделайте копию MyForm и переименуйте в “MyForm\_Query”.

2. Откройте лист свойств в узле *Designs/Design* и установите свойство *Columns* в 1.
3. В контекстном меню узла *Designs/Design* выберите *Создать Control/Group*. Поместите новое поле над полем закладки [tab] и переименуйте его в “rangeGroup”. Установите свойство *Caption* нового поля в “Filter”.
4. В контекстном меню поля rangeGroup выберите создать *StringEdit*. Переименуйте поле в “filterCurrencyCode”. Откройте лист свойств нового поля и установите *AutoDeclaration* в “Yes”, а *ExtendedDataType* в “CurrencyCode”.
5. Перейдите к *ClassDeclaration* формы и введите следующее:

```
public class FormRun extends ObjectRun
{
    QueryBuildRange    rangeCurrencyCode;
}
```

6. Теперь перейдите к *Data Sources/SalesTable/Methods* и переопределите *init()*:

```
public void init()
{
    QueryBuildDatasource    salesTableDS;
    ;
    super();

    salesTableDS             = SalesTable_ds.query().dataSourceTable(tablenum(SalesTable));
    rangeCurrencyCode       = salesTableDS.addRange(fieldNum(SalesTable, currencyCode));
}
```

7. Переопределите метод *executeQuery()* на источнике данных *SalesTable*:

```
public void executeQuery()
{
    rangeCurrencyCode.value(queryValue(filterCurrencyCode.valueStr()));

    super();
}
```

8. Перейдите к полю *filterCurrencyCode* и переопределите метод *modified()*:

```
public boolean modified()
{
    boolean ret;

    ret = super();


    SalesTable_ds.executeQuery();

    return ret;
}
```

```
}
```

## 9. Сохраните форму и создайте пункт меню для формы.

Форма `MyFrom_QueryFilter` имеет поле фильтра над табличным документом. Это поле используется для показа заказов с определенной валютой. Поле фильтр не связано с источником данных. Расширенный тип данных `CurrencyCode`, установленный в поле, добавляет метку и выводит форму выпадающего списка валют. Так как поле не связано с источником данных, то нельзя осуществить проверку по ограничению формы выпадающего списка валюты, значит, вам следует добавить критерий по полю валюты, которое не следует показывать в форме списка. Проверку следует сделать еще и для того чтобы проверять введенное значение кода валюты. Проверка опущена в целях упрощения примера.

В методе `init()` источника данных `SalesTable` был создан новый критерий запроса. Критерии всегда определяются в методе `init()`, так как этот метод выполняется однажды. Значение, используемое в критерии, определяется в методе `executeQuery()` так, как этот метод выполняется каждый раз при изменении в форме. Запрос формы можно выполнить еще раз, если пользователь вызовет диалог запроса, кликнув по иконке запроса , и изменит значение критериев. В этом примере метод `executeQuery()` вызовется при изменении значения поля `Currency`.

Отметьте, что если пользователь нажал иконку запроса, то критерий для кода валюты может быть изменен. Для защиты этого следует добавить следующую строку кода в метод запроса `init()`:

```
rangeCurrencyCode.status(RangeStatus::Locked);
```

Такой путь добавления поля фильтра в дизайн формы стандартно используется в Ахарта. Пользователь может добиться того же результата через диалог запроса фильтра. Это был упрощенный пример, поэтому для пользователя было не сложно изменить критерий отбора, используя диалог запроса. Если же у вас на форме будет несколько полей фильтров, и несколько источников данных в запросе формы, то это будет уже не такая простая задача для пользователя.

## Изменение источника данных из X++

Правила хорошего тона программирования гласят: никогда не вставляйте, не обновляйте или не удаляйте записи из методов формы. Такие операции следует всегда делать в методах таблицы или класса, в методах которого обрабатываются операций с данными формы.

Распространенная ошибка – когда для вставки, обновления или удаления записи из таблицы используется источник данных формы, но это прямое назначение таблицы, а не источника данных. Вам следует написать исполнимый класс, вызываемый из вашей формы для вставки новой записи в таблицу, используемую

в качестве источника данных. Для показа вставленной записи в форме, затем следует вызывать метод `executeQuery()`. Ваша новая запись будет показана, но курсор будет с позиционирован на первой записи в форме. Это может вас устроить, но часто требуется, что бы курсор позиционировался на новой записи. Источник данных имеет метод `findRecord()`, который следует использовать для позиционирования курсора. Не рекомендуется использовать `findRecord()`, если это будет результатом перебора большого количества записей в форме для необходимого позиционирования курсора. Вместо этого вам следует использовать методы источника данных, которые используются при вставке новой записи на форме.

---

#### Пример 8: Добавление записей

##### Элементы проекта MORPHXIT Forms

- Class, `MyForm_NewRecord`
- Form, `VendGroup`
- Menu item action, `MyForm_NewRecord`

В этом примере вы узнаете, как добавлять новую запись в источник данных из X++, используя вызовы тех же методов, как при создании, из формы пользовательского интерфейса.

1. Создайте новый класс, перейдите в `ClassDeclaration` и переименуйте в “`MyForm_NewRecord`”.
2. Сделайте класс исполнимым, добавив статический метод `main()`:

```
static void main(Args _args)
{
    MyForm_NewRecord newRecord;
;

    newRecord = new MyForm_NewRecord();
    newRecord.run(_args.record());
}
```

3. Создайте метод `run()` и добавьте следующий код:

```
void run(VendGroup _vendGroup)
{
    VendGroup vendGroup;
    VendGroup callerVendGroup = _vendGroup;
    FormDataSource formDataSource;
;

    formDataSource = callerVendGroup.dataSource();
    vendGroup.data(callerVendGroup);

    formDataSource.create(true);
}
```

```
callerVendGroup.data(vendGroup);
callerVendGroup.recId      = 0;
callerVendGroup.vendGroup  = "G1";
callerVendGroup.name       = "Group 1";
callerVendGroup.write();

formDataSource.reread();
formDataSource.refresh();
}
```

4. Сохраните класс и создайте пункт меню для вызова класса. Перейдите к листу свойств нового пункта меню и установите метку “New record”.
  5. Раскройте узел формы VendGroup *Designs/Design/[ButtonGroup:ButtonGroup]*. Перетащите пункт меню нового класса в группу кнопок.
  6. Сохраните форму VendGroup.
- 

При открытии формы VendGroup вы обнаружите новую кнопку с названием New Record. Нажав на кнопку, вы добавите новую запись в таблицу VendGroup, и курсор позиционируется на новой записи. Запись будет копией текущей выбранной записи, изменится только идентификатор группы поставщиков и описание.

В классе происходит обращение к источнику данных VendGroup. Текущая выбранная запись передается в качестве параметра в методе main(). Табличная переменная имеет метод datasource(), который инициализируется при вызове из формы. Этот метод используется для вызова методов источника данных create() и write() при добавлении и вставки новой записи. Так как текущая запись копируется, то системное поле recId следует установить в 0 для вставки новой записи.

Методы источника данных refresh() и reread() обновляют отображаемые записи в форме и читают новые из таблицы. Если вы пропустили вызовы этих методов, то вставленная запись будет не видна и станет ошибкой при использовании AOS, так как кэш не обновится.

Использование методов источника данных для вставки, обновления и удаления записей сделает интерфейс вашей формы более наглядным. Это предпочтительное решение при добавлении одной записи. Однако если вы вставляете много записей в форму, то вам следует использовать табличные методы вместо вызова методов источника данных, при вставке большого объема записей такой алгоритм будет замедлять обработку.

Системный класс Args используется в этом примере для получения вызывающего объекта. Класс Args часто используется в условиях проверки вызывающего объекта. В некоторых формах стоит условие обязательного вызова из другой

формы или класса. В этом случае следует использовать Args для проверки вызывающего объекта, и в зависимости от него накладывать различные критерии или устанавливать ограничения на поля формы. Вы найдете множество простых примеров по использованию Args в формах в стандартной поставке.

## Построение форм выпадающих списков

Добавление связи через расширенный тип данных будет являться автоматическим появлением формы выпадающего списка при использовании полем формы такого расширенного типа данных. Часто это достаточно и это прекрасно. Однако у вас могут быть случаи, когда такой связи не существует, а форма выпадающего списка необходима. Так же может понадобиться наложение фильтра на записи, представленных в форме списка, или же вам необходим более продвинутый вывод данных, при использовании нескольких закладок (tab pages), где формы выпадающих списков на этих закладках выводят различные записи в списке. Форма выпадающего списка может быть переопределена построением новой формы из X++ или заменой формы списка по умолчанию через расширенный тип данных.

Взгляните на следующий код, который взят из формы HRMApplication. Это метод performFormLookup() переопределенный на поле источника данных hrmRecruitingId:

```
public void performFormLookup(FormRun _p1, FormControl _formControl)
{
    super(hrmApplication::hrmRecruitingIdLookup(_p1), _formControl);
}
```

Для наложения фильтра на существующую форму списка из X++, вам следует использовать метод performFormLookup() на поле источника данных. И поля, связанные с источником данных, и не связанные имеют метод lookup(). Этот метод следует использовать, если ваше поле еще не имеет формы выпадающего списка или если вам необходимо построить собственную форму списка для поля формы не связанного с источником данных. Так как метод performFormLookup() был впервые представлен в Axapta 3.0, вы не найдете много примеров использования этого метода в стандартной поставке. PerformFormLookup() имеет объект FormRun, используемый как первый параметр и поле формы для построения формы списка как второй параметр. Рассмотрим метод hrmRecruitingIdLookup() на таблице HRMApplication, вызываемый для переопределения объекта FormRun, передаваемого параметром в super().

```
static formRun hrmRecruitingIdLookup(FormRun lookupFormRun)
{
    formDataSource formDataSource;
    query formquery;
;
    formDataSource = lookupFormRun.objectSet();
```

```

formQuery          = formDataSource.query();

formQuery.dataSourceNo(1).addRange(fieldNum(HRMRecruitingTable,
                                     status)).value(queryValue(HRMRecruitingStatus::Started));

return lookupFormRun;
}

```

Если раскрыть табличный метод `hrmRecruitingIdLookup()`, то вы увидите, что объект `FormRun` используется для получения формы выпадающего списка по умолчанию. Таким образом, вы получаете доступ к запросу, используемому в форме списка. Здесь добавляется новый критерий для фильтрации записей в форме списка. Запрос, используемый в форме списка – обычный запрос, который может быть объединен с любым источником данных.

Если вам необходимо создать форму выпадающего списка для поля формы, которое не имеет таковой, то вы не сможете использовать метод `performFormLookup`. Вместо этого используется класс `SysTableLookup` для построения формы выпадающего списка. Форма `InventTable` использует этот класс в некоторых методах, расположенных в таблице `InventTable`. Методы, строящие формы выпадающих списков в таблице `InventTable` имеют префикс `lookup*`.

Форма, созданная в АОТ может также использоваться как форма выпадающего списка. Формы, используемые как формы выпадающего списка, легко распознаются в АОТ по префиксу `*lookup`. Создавая форму выпадающего списка, также как и в форме по умолчанию генерируемой ядром, вам не следует использовать закладки. Просто добавьте поле табличного документа [grid control] с полями, представляемыми в списке. Устанавливая свойство **Frame** в узле *Design* в “Border” в форме, вы получите вид формы выпадающего списка по умолчанию. Метод `init()` в форме выпадающего списка должен возвращать выбираемое поле формы:

```

void init()
{
    super();
    element.selectMode(CustTable_AccountNum);
}

```

Этот код взят из формы `CustTableLookup`. Метод формы `selectMode()` устанавливает форму в ‘режим просмотра’ (`‘lookup mode’`). Когда пользователь нажимает `enter` или кликает по записи, форма закрывается и возвращается значение поля формы параметром.

```

client static void lookupCustomer(FormStringControl _formStringControl,
                                   CompanyId          _companyId = curExt())
{
    Args          args;
    FormRun        formRun;

```

```

if (! CompanyTmpCheck::exist(_companyId))
{
    throw error(strFmt("@SYS10666", _companyId));
}

changecompany(_companyId)
{
    args = new Args();
    args.name(formstr(CustTableLookup));
    formRun = classFactory.formRunClass(args);
    formRun.init();
    _formStringControl.performFormLookup(formRun);
}
}

```

Здесь форма CustTableLookup вызывается из метода lookupCustomer на таблице CustTable. Отметьте, что метод источника данных performFormLookup используется для связи формы выпадающего списка с вызывающей формой. Гораздо лучшее решение вместо вызова метода lookup напрямую из X++ - это связать вашу форму выпадающего списка через расширенный тип данных. Просто введите название формы в свойство **FormHelp** расширенного типа данных, и ваша форма станет автоматически формой выпадающего списка при использовании расширенного типа данных в форме. Это предельно просто и не требует ни одной строчки кода для вызова формы. Если необходимо, то вы можете установить разграничение в форме выпадающего списка по признаку вызывающего объекта и определять поведение формы. Расширенный тип ProjId использует такую форму. Многие расширенные типы используют специальные формы выпадающих списков, настраиваемые пользователем как, например, расширенный тип TransDate.

---

**Примечание:** Если вам необходимо создать форму выпадающего списка с объектами AOT, как таблицы или классы, посмотрите на методы класса Global с префиксом pick\*. Посмотрите форму sysDatabaseLogTableSetup на предмет использования метода pickTable() класса Global.

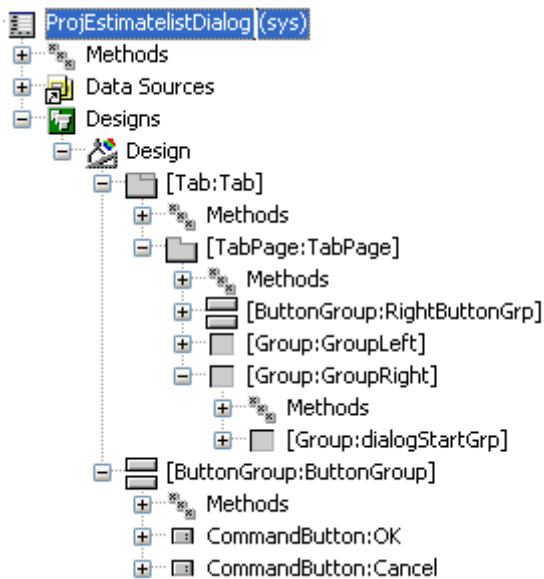
---

## Форма диалога

Стандартный путь создания диалога в классе runbase – это использование класса Dialog\*. Диалог предельно прост и все что вам остается сделать – это добавить несколько полей и расположить их в группах.

Однако если вам необходимо создать комплексный диалог с несколькими полями формы, причем сделать их недоступными при определенных режимах работы формы, то вы поймете, что диалог – это временное решение. Перед тем как зайти так далеко, вам вначале следует рассмотреть использование диалога. Структура runbase поддерживает использование простых форм в качестве диалога. Все что нужно это следовать нескольким правилам, при конструировании формы диалога.





**Рисунок 31: Форма диалога**

Взгляните на **рисунок 31: Форма диалога**. Эта форма - пример формы диалога, используемая в отчете в модуле Проектов (Project). Структура runbase проверяет, корректность создания и именования полей в дизайне. Эти обязательные для заполнения поля используются структурой runbase для расположения (позиционирования) различных частей дизайна. Если вы решите использовать эту форму диалога, то вы еще можете добавить дополнительные поля на неё, используя диалог класса. Добавляемые поля в форме диалога авто позиционируются.

При создании формы диалога быстрее копировать существующую форму диалога, чем создавать свою собственную с нуля. При использовании формы диалога, которая рассматривается выше, вам следует определиться с необходимыми полями. Теперь просто замените поля в группе полей GroupLeft полями, которые нужны именно в вашем диалоге. Все поля, используемые для ключевого ввода данных в форме диалога, следует создавать с применением edit методов. Это облегчит передачу значений в runbase класс.

Упомянутая выше форма диалога вызывается из класса ProjReport\_EstimateCheck. Метод dialog() в классе выглядит следующим образом:

```
public Object dialog()
{
    DialogRunbase dialog = Dialog::newFormnameRunbase(formstr(ProjEstimatelistDialog),this);
    ;
    dialog = super(dialog);

    // Add extra fields below

    return dialog;
```

```
}
```

Отметьте, что дополнительные поля диалога также можно добавить и в методе `dialog`. Метод доступа к значениям полей создается в форме диалога для каждого поля для установки или получения значения. Следует использовать метод `getFromDialog()`, если вы добавили дополнительные поля в форму диалога из класса.

## 6.5 Специальные формы

Эта секция показывает примеры, как создавать более продвинутые формы. Примеры должны дать вам представление о возможностях форм Ахарт. Если формы определены, и вывод данных с позиционирован, то у вас еще остается несколько возможностей по модификации вашей формы для более наглядного интерфейса без усложнения кода.

### Вызов определенных методов формы

Вы, может быть, замечали, что при выполнении формы из X++, вы не можете вызывать ваших собственных методов, созданных в форме. Если вы создали новый метод в форме, то ваш метод не известен (его нельзя использовать) при объявлении объекта `FormRun` вне формы, так как только члены объекта `FormRun` известны (можно использовать при объявлении объекта `FormRun`).

Иногда бывает необходимо вызвать ваш собственный метод формы из другого объекта. Этого можно достигнуть использованием класса родителя системного класса `FormRun`.

#### Элементы проекта MORPHXIT\_Forms

- `Form, MyForm_CallingUserMethod`
- `Menu item display, MyForm_CallingUserMethod`
- `Job, Forms_CallingUserMethod`

Продублируйте `MyForm`, переименуйте созданную форму в “`MyForm_CallingUserMethod`” и создайте новый пункт меню для новой формы.

```
void setReadOnly()
{
    salesTable_ds.allowEdit(false);
    salesTable_ds.allowCreate(false);
    salesTable_ds.allowDelete(false);
    salesTable_ds.insertAtEnd(false);
    salesTable_ds.insertIfEmpty(false);

    element.design().caption(strfmt("READ ONLY - %1", element.design().caption()));
}
```

Добавьте приведенный выше метод в вашу форму MyForm\_CallingUserMethod. Вызов этого метода запретит изменения данных в форме и установит текст заголовка формы с сообщением только на чтение. Вызовем этот метод формы из другого объекта, например из задания.

```
static void Forms_CallingUserMethod(Args _args)
{
    Args        args;
    Object       formRun;
;

    args = new Args();

    formRun = new menuFunction(menuItemDisplayStr(MyForm_CallingUserMethod),
                               MenuItemType::Display).create(args);

    formRun.init();
    formRun.setReadOnly();
    formRun.run();
}
```

Создайте новое задание [job] с расположенным выше кодом. При выполнении задания будет вызвана форма MyForm\_CallingUserMethod. Вместо использования системного класса FormRun, используем системный класс Object для объявления объекта FormRun. Object не проверяется на доступные методы FormRun. Это представляет опцию для вызова любого другого метода, добавленного в форму. Вы должны быть уверены, что метод существует, так как не производится проверки во время выполнения и не появляется списка методов при установке точки после переменной класса Object.

Метод, добавленный в форму MyForm\_CallingUserMethod, не имеет функциональности ссылки на форму и методы источника данных. Но использование подобных методов может быть ценно, так как вам необходимо добавить только пару строк кода в задание, что бы вызвать ваши методы на форме. Так как форма может быть вызвана из различных мест, то вы можете сохранять код в одном месте.

## Переопределение методов

Если вы программно добавляете поля на форму, то у вас может быть ситуация, когда необходимо переопределить еще и методы таких полей. Переопределить методы динамического поля возможно, вам просто необходимо активировать эту возможность на форме и создать метод на форме для каждого переопределенного метода.

### Элементы проекта MORPHXIT Forms

- Form, MyForm\_OverloadMethod
- Menu item display, MyForm\_OverloadMethod

## ➤ Job, Forms\_OverloadMethod

Продублируйте MyForm, переименуйте созданную форму в “MyForm\_OverloadMethod” и создайте пункт меню для новой формы.

```
public void init()
{
    super();

    element.controlMethodOverload(true);
}
```

В методе init() новой формы вы должны активировать опцию переопределения методов для полей, которые добавляются программно. Код для переопределяемого метода должен быть добавлен в форму. Синтаксис такой: <controlname>\_<controlmethod to be overridden>. Это означает, что вы должны знать имя добавляемого поля наперед. В нашем случае метод нового поля называется showOnlyOpenOrders, переопределим его:

```
Boolean showOnlyOpenOrders_modified()
{
    FormCheckBoxControl checkBoxControl = this.controlCallingMethod();
    QueryBuildRange      rangeSalesStatus;
    Boolean              ret;
;

    ret = checkBoxControl.modified();

    rangeSalesStatus =
SalesTable_ds.query().dataSourceTable(tablenum(SalesTable)).findRange(fieldnum(SalesTable,
salesStatus));

    if (!rangeSalesStatus)
    {
        rangeSalesStatus =
SalesTable_ds.query().dataSourceTable(tablenum(SalesTable)).addRange(fieldnum(SalesTable,
salesStatus));
    }

    if (checkBoxControl.value() == NoYes::Yes)
    {
        rangeSalesStatus.enabled(true);
        rangeSalesStatus.value(queryValue(SalesStatus::Backorder));
    }
    else
    {
        rangeSalesStatus.enabled(false);
    }

    SalesTable_ds.executeQuery();

    return ret;
}
```

Добавим программно в форму новое поле типа галки [check box]. Новое поле формы будет использоваться для показа только открытых заказов. Код в этом методе будет выполняться при изменении значения в новом поле. Вызвав `this.controlCallingMethod()` вы сразу же получаете доступ к новому полю. Это важно, так как иначе бы вам пришлось делать вызов `super()` нового метода вручную. Также ваш метод должен возвращать то же самое значение, что и переопределенный стандартный метод поля.

---

**Примечание:** Вы можете разместить метод поля, добавляемого программно [runtime], в классе, используя метод `controlMethodOverloadObject()` объекта `FormRun`.

---

В методе добавляется новый критерий для фильтрации по статусам заказа. Так как этот метод может вызываться несколько раз объект `rangeSalesStatus` пытается инициироваться первым, определяя в запросе (источнике данных) критерий по полю `salesStatus`. Если не обнаружено критериев метод выполняется в первый раз, и новый критерий добавляется. Ставя галку в упомянутом поле, критерий становится недоступным и происходит фильтрация по открытым заказам. Если снять галку, то оба критерия (если существуют оба, или если существует один) становятся не активным.

```
static void Forms_OverloadMethod(Args _args)
{
    Args                args;
    FormRun              formRun;
    FormCheckBoxControl  ctrlShowOnlyOpenOrders;
;

    args = new Args();

    formRun = new menuFunction(menuItemDisplayStr(MyForm_OverloadMethod),
                                MenuItemType::Display).create(args);

    formRun.init();
    formRun.design().columns(1);

    ctrlShowOnlyOpenOrders = formRun.design().addControl(FormControlType::CheckBox,
                                                            "showOnlyOpenOrders");

    ctrlShowOnlyOpenOrders.label("Only open orders");
    formRun.design().moveControl(ctrlShowOnlyOpenOrders.id());

    formRun.run();
}
```

Форма вызывается из задания. После инициализации формы активируется переопределенный метод и добавляется поле типа галки. Запомните, что название нового поля [name control] должно состоять из префикса названия метода формы, в данном случае до `modified()`. Необходимо добавить метку для поля. Мы не используем меточную систему для упрощения нашего примера. Для позиционирования нового поля в верхней части формы число колонок для дизайна формы следует установить в 1. Так как новое поле всегда прикрепляется,

как последнее поле, метод дизайна `moveControl()` используется для позиционирования поля `check box`, как первого поля. `MoveControl()` имеет два параметра, второй параметр определяет какое поле вставляется следующим. Так как поле типа галки должно быть первым полем, то второй параметр пропускаем.

Переопределение методов динамических полей [dynamic control] не оптимально, так как вам каждый раз следует создавать метод на форме для нового добавляемого поля. Часто при добавлении динамического поля вы даже не предполагаете количество программно добавляемых полей, особенно при добавлении нескольких полей в несколько форм. Альтернатива этому – программное добавление пункта меню. Добавленный пункт меню вызывает исполнимый класс, и вы можете добавить всю логику в вызываемый класс. Это вам не предоставит в точности тех же самых возможностей, что и при переопределении методов полей формы, но в большинстве случаев это реализует ваши требования.

## Общие изменения формы

Каждый раз при запуске формы вызывается класс `SysSetupFormRun` через класс `ClassFactory`. Класс `SysSetupFormRun` является наследником системного класса `FormRun`. Это означает, что у вас есть возможность обращения к `FormRun` при выполнении формы. Модифицируя `SysSetupFormRun`, вы сможете делать изменения во время выполнения [runtime changes] на любой формы без дополнительного изменения самой формы. Эта продвинутая возможность может решить проблему излишнего кодирования.

Помните, что такие модификации загружаются в самом начале работы, так как ваш код будет вызываться каждый раз при выполнении любой формы. Вам следует быть осторожным при модификации `SysSetupFormRun`. Всегда пробуйте ваши изменения с локальной установкой Ахарта. Если вы сделали ошибки в коде, то это можете сделать нерабочим целое приложение, а это не понравится другим пользователям этого приложения. Это воздействие будет не только на стандартные формы приложения. Все инструменты MorphX, существующие в АОТ, такие как компилятор форм, поиск по форме не будут работать.

### Элементы проекта MORPHXIT\_Forms

#### ➤ Class, `SysSetupFormRun`

Добавьте новый метод с названием `showDatasourceInfo` в класс `SysSetupFormRun`:

```
void showDatasourceInfo()
{
    FormDatasource          formDatasource;
    FormGroupControl        formGroupControl;
    FormStaticTextControl    formStaticTextControl;
```

```

Counter                counter;
Counter                noOfDatasources;
;

noOfDatasources = this.form().dataSourceCount();

if (noOfDatasources)
{
    formGroupControl = this.form().design().addControl(FormControlType::Group,
                                                        "formGroupControl");

    formGroupControl.caption("Tables used by form");
    formGroupControl.frameType(3);
}

for (counter=1; counter <= noOfDatasources; counter++)
{
    formStaticTextControl = formGroupControl.addControl(FormControlType::StaticText,
                                                         "formStaticTextControl" + int2str(counter));
    formStaticTextControl.text(tableid2name(this.form().dataSource(counter).table()));
}
}

```

Этот метод выводит имена таблиц, используемых в источнике данных формы. Группа полей будет добавляться на каждую форму, и для каждого источника данных будет выводиться поле [static text control], показывающее его название. Группа полей будет выводиться только при условии, что форма имеет источник данных.

```

void new(Args args)
{
    KMActionMenuButtonAuto          KMActionMenuButtonAuto;
    CCDatalinkMenuButtonAuto        CCDatalinkMenuButtonAuto;
    KMKnowledgeCollectorMenuButtonAuto KMKnowledgeCollectorMenuButtonAuto;
    ;

    super(Args);
    // CC Start
    if (this.name() != formStr(SysLicenseCode))
    {
        KMActionMenuButtonAuto = new KMActionMenuButtonAuto(this);
        if (KMActionMenuButtonAuto.check())
            KMActionMenuButtonAuto.create();

        CCDatalinkMenuButtonAuto= new CCDatalinkMenuButtonAuto(this);
        if (CCDatalinkMenuButtonAuto.check())
            CCDatalinkMenuButtonAuto.create();

        KMKnowledgeCollectorMenuButtonAuto= new KMKnowledgeCollectorMenuButtonAuto(this);
        if (KMKnowledgeCollectorMenuButtonAuto.check())
            KMKnowledgeCollectorMenuButtonAuto.create();
    }
    // CC End

    this.showDatasourceInfo();
}

```

}

Наш метод вызывается из метода `new()` класса `SysSetupFormRun`. При запуске любой формы, автоматически добавиться группа полей в рамочке, показывая таблицы, используемые в форме.

Это достаточно простой пример, как производить общие изменения в формах. Посмотрите вызовы других методов в методе `new()`. Код используется модулем HRM для динамического добавления кнопок на форму. Класс `CCDataLinkMenuButtonAuto` используется для программного добавления кнопок на форму. Эта возможность HRM называется связь по данным [Data links] и используется для связи записей Ахарты с внешними базами. Если форма использует источник данных настроенный на связь по данным [data link], то кнопка, вызывающая форму со связанными записями из внешней базы, добавляется во время выполнения формы.

## Цвета

Любая форма Ахарты использует стандартный набор цветов. Но только в нескольких местах в стандартной поставке поля формы раскрашиваются другими цветами. Изменяя цвета на форме, вы простыми средствами делаете ваш интерфейс более наглядным. Раскраска полей – это альтернатива сортировке данных, так как это делает возможным упростить поиск поля с определенным значением.

### Элементы проекта MORPHXIT Forms

- `Form, MyForm_ColorRow`
- `Menu item display, MyForm_ColorRow`
- `Menu item display, MyForm_ColorColumn`

Продублируйте форму `MyForm` и переименуйте созданную форму в “`MyForm_ColorRow`”.

Перейдите к форме `MyForm_ColorRow` и переопределите метод `displayOption()` источника данных `SalesTable` следующим кодом:

```
public void displayOption(Common _record, FormRowDisplayOption _options)
{
    SalesTable salesTableLocal = _record;
    ;

    if (salesTableLocal.currencyCode == CompanyInfo::find().currencyCode)
    {
        _options.backColor(WinApi::RGB2Int(255, 255, 255)); // White
    }
    else
    {
        _options.backColor(WinApi::RGB2Int(255, 0, 0)); // Red
    }
}
```



```

    }

    super(_record, _options);
}

```

Метод `displayOption` выполняется для каждой записи в форме при загрузке формы. Вы используете этот метод для изменения цвета поля по умолчанию. В этом примере записи, у которых валюта заказа отличается от валюты компании, будут подкрашены красным. Отметьте, что цвет должен устанавливаться на все записи, а не те только те, которые выделяются красным.

Рассмотренный пример раскрашивает все поля записи. Измените метод `displayOption()` на форме `MyForm_ColorRow` следующим кодом:

```

public void displayOption(Common _record, FormRowDisplayOption _options)
{
    SalesTable salesTableLocal = _record;
    ;

    if (salesTableLocal.currencyCode == CompanyInfo::find().currencyCode)
    {
        _options.backColor(WinApi::RGB2Int(255, 255, 255)); // White
    }
    else
    {
        _options.backColor(WinApi::RGB2Int(255, 0, 0)); // Red
        _options.affectedElementsByControl(SalesTable_SalesId.id(), SalesTable_CurrencyCode.id());
    }

    super(_record, _options);
}

```

Единственное отличие от предыдущего примера – это вызов метода `affectedElementsByControl()`. Поэтому только поля, передаваемые в список параметров для функции `affectedElementsByControl()` поменяют свой цвет. Вы можете добавить любое количество полей для раскраски. Только помните, что поля, добавляемые в функцию, должны быть явно объявлены [auto declared].

Этот пример показывает принудительное кодирование для критериев при использовании цветов. В реальной ситуации вам следует предоставить настройки критериев пользователю. При использовании модуля HRM форма `EmplTable` использует подкраску рядов. Цвета, используемые в `EmplTable`, определены пользователем в параметрах формы `HRMParamters`.

## 6.6 Резюме

В этой главе вы узнали, как использовать некоторые части формы, такие как источник данных и поля формы в дизайне формы. Вы также познакомились с тем,

как получить данные из базы и как ими можно манипулировать в форме. Так же вы приобрели знания, как создавать формы с стандартным интерфейсом. Так как формы – это главная часть пользовательского интерфейса, то стандартный вид форм – это существенно, они должны быть понятными пользователю.

## 7 Отчеты

Отчеты в Ахapta строятся на запросах и располагаются в Application Object Tree (АОТ). Отчет – это лишь вывод данных из таблиц. Отчеты не содержат данных. При выполнении он получает данные из базы данных. Когда отчет выполнен, вы можете распечатать его сразу же или определить пакетное задание [batch job] для исполнения отчета позже на пакетном сервере [batch server]. Пакетная обработка обычно используется для «тяжелых» отчетов, таких как распечатка месячного баланса по клиенту.

Отчеты в Ахapta очень гибкие. Технология Morphx предоставляет инструменты для создания отчета без программирования. Используя Morphx, вы можете фильтровать данные или изменять вывод данных на печать.

Существует два способа создания отчетов в Ахapta. Это мастер отчетов и создание вручную в АОТ в узле *Reports*.

Эта глава посвящена техническим аспектам создания отчетов, а не объяснение различных опций, содержащихся в диалоге отчета. Все же очень важно понимать интерфейс Ахapta. Вам нет необходимости, знакомясь с этой главой, понимать пользовательский интерфейс. Если вы не знакомы с конечным интерфейсом пользователя, вы можете получить детальную информацию из руководства пользователя, входящего в стандартную поставку.

### 7.1 Мастер отчета

Мастер отчетов – это средство разработки для не посвященного в технические подробности пользователя. Мастер доступен в меню **Сервис \ Средства разработки \ Мастера \ Мастер отчетов**. Это подходящее средство для начала изучения отчетов Ахapta. Мастер отчетов - это средство конечного пользователя, проводящее через все ступени создания отчета. У вас есть возможность сортировки отчета, созданного в АОТ. Это увлекательно для начинающего программиста просматривать объекты созданные мастером для ознакомления со стандартными элементами отчета. Мастер так же будет полезен опытным программистам, которые создают базовую структуру отчета, а затем используют её для финальных модификаций.

Для пошагового руководства, показывающего использование мастера отчетов смотри: **Мастер Отчета**.

### 7.2 Создание отчета

Просмотрев отчет, созданный мастером вы увидите, что будет требоваться для создания отчета вручную. Когда вы в первый раз создаете новый отчет полезно просмотреть некоторые стандартные отчеты в АОТ, что бы выяснить, не существует ли уже, подходящего. Скопируйте этот отчет и начните модификации в вашей копии. За помощью в рассмотрении вызова отчета из меню, смотри главу **Меню и Пункты Меню**.

Когда начинаете разрабатывать отчет, не забудьте про tutorial reports. Это также очень полезно. Смотрите отчеты с префиксом **tutorial\_**.

Отчеты в Ахарта разделены на две составные части: источник данных (*Data Sources*), который определяет данные, из которых формируется отчет и дизайн (*Designs*), который используется для вывода представления отчета. **Рисунок 1:Обзор Отчета** показывает типичный отчет.

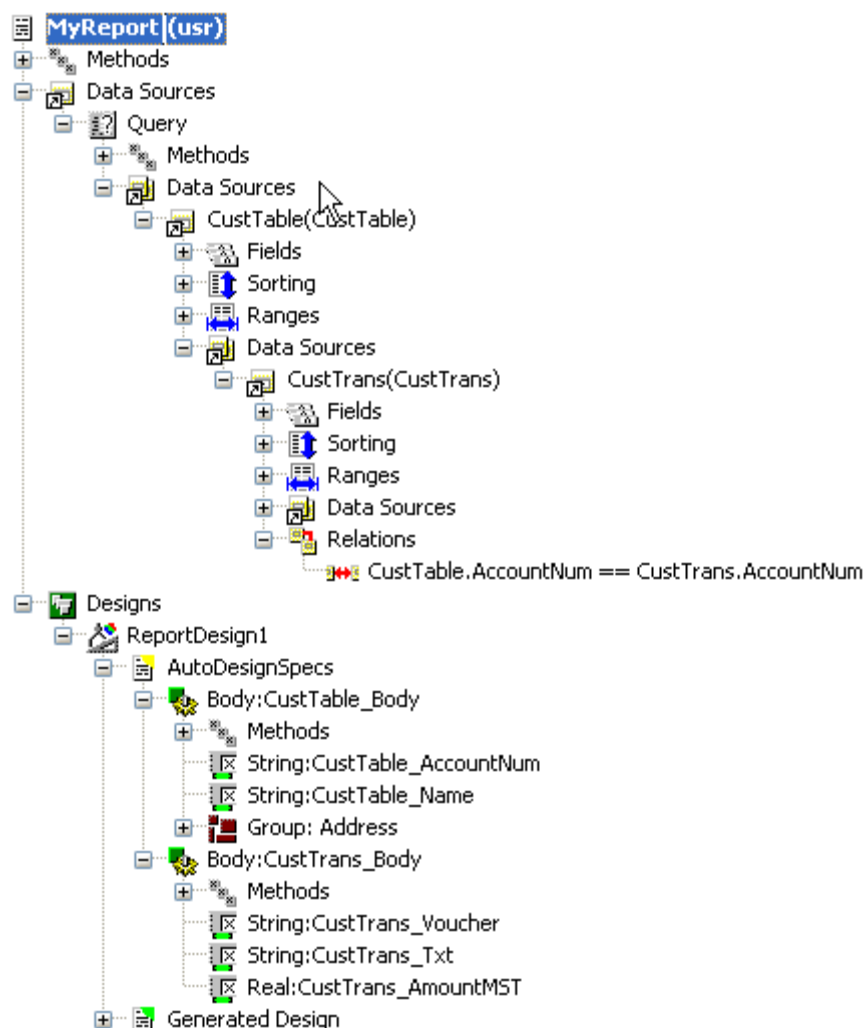


Рисунок 32: Обзор отчета

---

### Пример 1: Мой первый отчет

#### Элементы проекта MORPHXIT Reports

- Report, MyReport
- Menu item output, MyReport

Как начальное упражнение, создайте отчет, как показано на **Рисунке 1 Обзор Отчета**. Отчет распечатывает проводки покупателя сгруппированные по покупателю. Рассматриваемый пример достаточно прост, но наша задача

разобрать базовые шаги при создании отчета. Позже в этой главе некоторые детали мы усложним и добавим их к этому примеру.

1. Создайте новый отчет через контекстное меню узла *Reports* AOT и выберите *Создать Report*. Новый отчет называется "Report1". Откройте свойства созданного отчета (ALT+Enter) или через контекстное меню и переименуйте отчет "MyReport".
2. Для этого отчета вы получите информацию из главной таблицы покупателей, связанной с каждой проводкой покупателя. Это осуществим определением двух уровней в источнике данных запроса. Например, таблица клиентов на первом уровне, а таблица проводок на втором уровне. Разработка в MorphxX часто предполагает использование технологии «drag and drop». И в нашем случае перетащите таблицы из AOT в источник данных отчета, который вы создали. Использование технологии «drag and drop» настолько просто, насколько возможно, также Ахарта позволяет открывать многократно AOT. В нашем случае откройте ещё одно окно AOT и раскройте узел *Data Dictionary/Tables*. Выберите таблицу *CustTable*. Раскройте узел *Data Sources/Query* в вашем отчете и перетащите *CustTable* в узел *Data Sources/Query/Data Sources*. Теперь первый уровень запроса создан. Раскройте узел источника данных *CustTable* в вашем отчете. Перетащите таблицу *CustTrans* в узел *CustTable/Data Sources*.
3. Ваши два источника данных запроса должны быть связаны иначе все проводки, содержащиеся в таблице проводок, будут выведены в отчет для всех клиентов. Перейдите в узел *CustTrans* запроса [*Query*] и установите свойство **Relations** в "Yes." Узел *CustTrans/Relations* теперь содержит связь двух таблиц. Теперь в отчет попадут только проводки, принадлежащие клиентам, которым эти проводки принадлежат.
4. Теперь можно выводить данные из запроса в отчет. Перейдите в узел *Designs* и в контекстном меню выберите *Создать Report Design*. Новый дизайн называется "ReportDesign1". Откройте свойства узла *ReportDesign1* и введите текст "Customer transactions list" в поле **Caption**.
5. Перейдите к узлу *ReportDesign1/AutoDesignSpecs*. Через контекстное меню узла выберите свойство *Генерация дизайна по Query*. Ваш дизайн теперь содержит две секции *body* по одной на каждую таблицу в запросе.
6. Последний шаг - это выбор полей, которые будут напечатаны. В контекстном меню узла *Query* вашего отчета выбираем *Open New Window*. Менее трудоемко перетащить поля, которые выводятся на печать, в дизайн. Выберите поля *AccountNum* и *Name* из источника данных *CustTable* и перетащите в секцию узла *CustTable\_Body*. Перейдите к источнику данных *CustTrans* и перетащите поля *Voucher*, *TransDate* и *AmountMST* в секцию узла *CustTrans\_Body*.
7. Теперь у вас есть отчет как показано на **Рисунке 1**. Запустите отчет через контекстное меню *Открыть*. Ахарта выведет диалог для фильтрации,

сортировки и других настроек. Кликните *OK*. Следующий диалог это настройки принтера. Проверьте, чтобы свойство канала вывода было установлено “Экран” и кликните *OK*. Ваш отчет будет выведен на экран.

8. Как вы можете заметить, сформированный отчет имеет недоработки. Задачи по форматированию можно возложить на MorphxX, определив использование специальных шаблонов. Шаблоны в MorphxX формируют стандартные опции, такие как заголовок. Как добавить шаблон к вашему отчету, просто перейти к узлу *Designs/reportDesign1* и определить свойство **ReportTemplate**. Кликните на стрелочке и выберите необходимый шаблон из списка.
  9. Запустите отчет еще раз с шага 7. Теперь отчет имеет заголовок с информацией о названии, количестве страниц, дате и времени. Поздравляем, вы создали ваш первый отчет!
- 

В рассмотренном примере MyReport вы не написали ни строчки кода. Когда вы создаёте отчеты в Ахарт, то можете не беспокоиться о написании кода для связи таблиц и размещения полей в отчете. MorphxX сделает это за вас. Просто создайте запрос и перетащите необходимые поля в дизайн отчета. Вам следует использовать X++ когда создаются более продвинутые отчеты, где формирование данных затруднительно в запросе или когда вам необходим специальный отчет.

Когда выполняется отчет, то вам показывается два диалоговых окна. Создайте пункт меню для MyReport, просто перетащив MyReport в узел *Menu Items/Output*. Теперь вы можете вызвать отчет из контекстного меню, нажав *Открыть* или верхнего меню toolbar или CTRL+O. Информация, содержащаяся в двух отдельных окнах, теперь представлена в одном. Запуская отчет через пункт меню, автоматически активируется более сложная структура запуска отчета. Это диалог, который видит пользователь. Дополнительные детали будут рассмотрены далее в этой главе.

## 7.3 Запрос отчета

Когда запрос уже не может формировать необходимым вам образом данные, следует использовать X++, хотя в большинстве случаев вывод данных можно обеспечить использованием запроса, который определяется в источнике данных отчета. Генератор отчета использует стандартный запрос Ахарт. За подробной информацией обращайтесь к главе **Запросы**.

Перед построением вашего запроса, вам необходимо определить, какие таблицы вы будете использовать. Часто, как и в нашем примере MyReport, вам необходимо вывести данные из одной таблицы или связанной с ней таблицы. В этом случае вам необходимо исследовать формы на предмет имен таблиц, содержащихся в них.

За помощью в расположении таблиц и полей смотри главу **Меню и Пункты Меню**.

После того как вы определили таблицы, вам необходимо определить, как сортировать данные, и как их фильтровать. Решение, принятое вами играет существенную роль. Небольшое планирование на этой стадии существенно снизит трудозатраты на переделку в будущем. Например, критерий выборки запроса более эффективен, когда помещается на более высокий уровень в источнике данных. Как основное правило, вам следует ставить критерий для таблиц в первых двух уровнях вашего запроса. Если ваш отчет фильтрует данные на третьем уровне запроса, то вам необходимо вернуться к дизайну и попытаться найти более эффективный путь. Можно ли добиться этого использованием двух отдельных отчетов? Если нет, может использование временной таблицы поможет достичь результата? Вы решаете сами.

Для большинства отчетов вы добавляете требуемую таблицу в узел запроса [Query]. Если к тому же используются данные и из связанной таблицы, тогда необходимо объединить таблицы в запросе, как вы делали с CustTable и CustTrans в примере MyReport. Иногда необходимо выводить данные из двух таблиц, у которых нет прямой связи. В этом случае возможно использование еще одной таблицы, которая обеспечивала бы связь с каждой таблицей, используемой в отчете. Например, если вы хотите вывести отчет по счетам, сгруппированный по покупателю, то нет прямой связи между таблицей Customer и таблицей Sales Invoice Lines. Поэтому таблица Customer Invoice Journal должна использоваться для установления связи и создания отчета. Смотри **Рисунок 2: связь между CustTable и CustInvoiceTrans**. Вам необходимо добавить все три таблицы в запрос или добавить на первый уровень таблицу **CustTable**, как источник данных, и использовать X++ для вывода данных других двух таблиц. В общем предпочтительнее использовать запрос вместо X++ - это добавляет возможность пользователю заполнения диалога запроса.

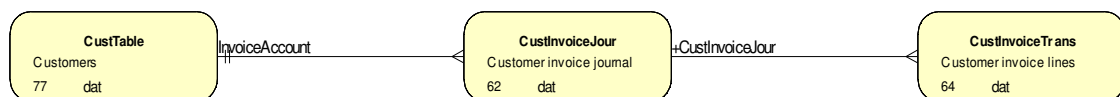


Рисунок 33: Связь между CustTable и CustInvoiceTrans

Объединение источников данных может быть сделано двумя способами. Если у источников данных уже есть связь, тогда свойство **Relation** источника данных на нижнем уровне должно быть *True*, как в нашем примере MyReport. Связь будет видна под узлом *Relations* для объединенных источников данных. Если связь не показывается, тогда необходимо вручную создать связь под узлом *Relations*, и свойство **Relation** должно быть установлено в *False*. Правила хорошего тона программирования [Best practice] гласят, чтобы использовать существующую связь чаще, чем созданную вручную. По умолчанию источники данных объединяются как inner join, но режим объединения можно изменить в свойстве для объединяемого [зависимого] источника данных. Inner joins часто используется в бизнес отчетах, где у вас есть данные в главной таблице и вы хотите выводить данные из подчиненной таблицы. Однако если вы хотите вывести все данные из главной таблицы, даже если нет данных в подчиненной таблице, то вам следует изменить режим на OuterJoin в подчиненной таблице.

Кликните правой кнопкой на узле *Fields*, вы выбираете поле или агрегатную функцию. По умолчанию все поля из текущей таблицы уже добавлены списком и нет смысла еще раз добавлять поле из таблицы. Если вы добавляете агрегатную функцию, то все поля удаляются, и вы не можете использовать и то и другое. Удалите агрегатную функцию и восстановите список полей, измените в узле *Fields* свойство **Dynamic** в *Yes*. Агрегатную функцию можно использовать, если вы хотите посчитать число покупателей или сгруппировать их. Тогда вам необходимо выбрать таблицу определить агрегатную функцию и поля, которые следует использовать.

---

### Пример 2: Агрегатная функция

#### Элементы проекта MORPHXIT Reports

- Report, MyReport\_aggregate
- Menu item output, MyReport\_aggregate

Следующий пример выводит количество покупателей, сгруппированный по покупателям. Рассмотрим агрегатные функции.

1. Добавим таблицу Customer в запрос отчета. Затем в контекстном меню узла *Fields* выберем агрегатную функцию **Count**. Считываемое поле должно быть *AccountNum*.
2. В источнике данных CustTable установим свойство **OrderMode** в *Group by*. Последний шаг – это определим, как информация должна сортироваться. Переходим к узлу *Sorting* и добавляем поле *CustGroup*. Теперь у вас есть запрос, как показано на **Рисунке 3: Агрегатная функция**.

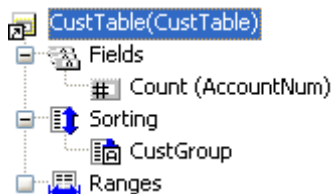


Рисунок 34: Агрегатная функция

3. Следующий шаг – создадим дизайн для вывода отчета. Создадим auto design и выберем *Генерация дизайна по Query*, как делали в примере MyReport. Теперь у вас есть body section для CustTable с одним полем, выводящим поле CustGroup. Добавим поле AccountNum.
  4. Запустим отчет. В ряду выводятся данные для каждой группы покупателей. Поле AccountNum показывает число покупателей в каждой группе покупателей.
- 

Когда мы используем агрегатную функцию, данные должны выбираться с параметром группы, установленном в **OrderMode**. Компилятор выдаст ошибку, если использовать выборку order by. Когда OrderMode установлен в group by MORPHXIT выводит одну запись для каждой группы, основанной на сортируемых полях. Это значит, что можно добавить только сортируемые поля.



Вы можете добавить сколько угодно агрегатных функций. Это пригодится в случае, когда необходимо напечатать список проводок и найти минимум, максимум или среднее.

Узел *Sorting* под узлом *Data Sources* используется для определения, как сортировать выходные данные отчета. Этого можно добиться использованием индексов или выбором полей. Как минимум один индекс или сортируемое поле должны быть определены. Можно предоставить пользователю выбор поля сортировки. Помните, что использование поля в сортировке более чем индекс может «подтормаживать» отчет.

Поле сортировки добавлено со свойством **AutoSum**, это используется для того, чтобы MorphX печатал подсуммы при изменении значения поля. Формирование авто суммы подробно объясняется в **Auto design**.

Критерии [Ranges] используются для фильтрации записей в отчете. Критерии по умолчанию определяются использованием узла *Range* под узлом *data source*. При выполнении можно разрешить пользователю добавлять или удалять критерии по умолчанию в зависимости от установленного свойства на *range*. Вы можете определить значение по умолчанию для *range*. Свойство может быть установлено в критерии как заблокировано [locked] или скрыто [hidden]. Если не определено ни одного критерия первый элемент каждого индекса на таблице будет использован, как критерий по умолчанию. Попробуйте выполнить отчет MyReport. Вы увидите, что критерий по умолчанию установлен. Теперь вернитесь и добавьте поля AccountNum в узел *Data sources/CustTable/Ranges*. Когда выполняете MyReport, то только критерий AccountNum будет активен.

## 7.4 Шаблоны

В Ахapta существует два различных вида шаблонов [template]: шаблоны отчета [report templates] и шаблоны секции [section templates]. Шаблоны расположены в двух первых узлах в *Report* в АОТ.

### Шаблон Отчета

Шаблоны отчета используются для определения базовых форматов таких как заголовок. Вы можете создать свой шаблон для дальнейшего использования, например, вывода данных из специфической таблицы. Но шаблоны отчета обычно не связывают с таблицей при определении заголовка, нумератора страниц и т.д. Шаблон **InternalList** используется в примере MyReport, как обычный шаблон, формирующий заголовок, название компании, номера страниц, дату и время. Для просмотра шаблона, выберите его в АОТ, открыв визуальным редактором в контекстном меню узла *template*, и выберите *Правка*. При создании шаблона отчета с полями из специальной таблицы, чтобы любой отчет может использовать этот шаблон, возможно при условии явного объявления таблицы и вывода требуемых записей.

Элементы проекта MORPHXIT Reports

- Report template, MyInternalList
- Report, MyReport\_MyInternalList

В этом примере вы создадите новый шаблон, основанный на шаблоне InternalList. Шаблон InternalList содержит базовый формат заголовка. Далее добавим секции Prolog и Epilog в новый шаблон. Новый шаблон добавим в новый отчет, созданный на основе MyReport.

1. Начала создайте копию MyReport и переименуйте новый отчет в “MyReport\_MyInternalList”.
2. Перейдите к шаблону **InternalList** в AOT. В контекстном меню выберете *Дублировать*. Переименуйте новый шаблон отчета в “MyInternalList”.
3. В контекстное меню выберете *Создать* и далее выберете *Prolog*. Prolog будет содержать текст, заполняющий новую страницу. Во-первых, необходимо определить текст, который будет выводиться на печать. Переходим к *Prolog/Methods* и в контекстном меню выбираем *Создать Method*. Открываем новый метод и вводим следующее:

```
display description prologDescription()
{
    return sprintf("Start of report: %1", element.design().lookupCaption());
}
```

Этот метод возвращает строку “Start of report”, содержащую значение свойства заголовка дизайна отчета. Этот метод необходимо вывести в prolog’s design. Закройте редактор и перетащите метод в узел *Prolog*. Ахарта создаст строковое поле, которое выведет возвращаемое значение нашего дисплей метода.

4. В контекстном меню выберете *Создать* и далее выберете *Epilog*. Теперь создадим следующий метод, и перетащим метод в *Epilog*.

```
display epilogDescription()
{
    return sprintf("End of report: %1", element.design().lookupCaption());
}
```

5. Для того чтобы секции prolog и epilog печатались на новой странице, необходимо таковую добавить. Переходим к *Prolog/Methods* в контекстном меню выбираем *Перекрыть метод* и выбираем executeSection(). Отметим, что вызов newPage расположен после super().

```
public void executeSection()
{
    super();

    element.newPage();
}
```

6. Добавим новую страницу к секции epilog. Вызов новой страницы должен выполняться перед super() в секции epilog, так как epilog должен печататься на новой странице. Теперь у вас есть шаблон отчета как показано на **Рисунке 4, шаблон отчета**.

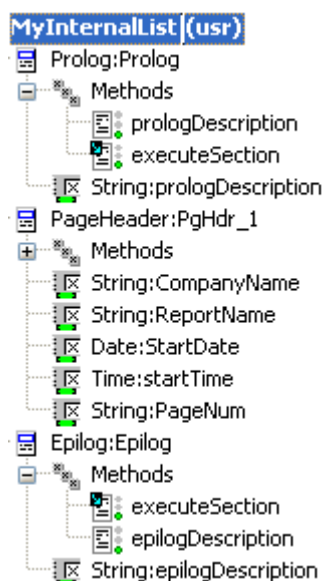


Рисунок 35 Шаблон отчета

7. Следующий шаг – это использование нового шаблона отчета в MyReport\_MyInternalList. Переходим к узлу *Designs/ReportDesign1*, откроем лист свойств и выберем MyInternalTemplate, как шаблон отчета.
8. Создадим новый пункт меню для отчета. При выполнении отчета prolog страница напечатается перед отчетом, а epilog страница после отчета.

Шаблон MyInternalList использует дисплей методы для вывода значений в поля отчета. Так же как и в формах, вы часто будете использовать display методы при создании отчетов. Это один из путей для вывода данных, которые не возможно получить напрямую из запроса отчета. В этом случае вы используете display метод, который возвращает строку текста, в других случаях это может быть результат некоторого расчета. Просто создавайте display метод и перетаскивайте его в дизайн. Вы можете не беспокоиться о типе поля, назначенного в display методе, это сделает за вас MorphX.

Вам в большинстве случаев придется создавать два или более шаблонов для использования в отчетах. Преимущество использования шаблона – легкость по стандартному форматированию отчета. Если позже вы решите изменить шаблон отчета, то все отчеты, использующие этот шаблон автоматически изменятся

**Примечание:** Часто бывает необходимо печатать как номер страницы, так и общее количество страниц. Шаблон InternalList использует element.page() для печати текущего номера страницы. Метод element.pagesTotal() возвращает общее количество печатаемых страниц. Element.pagesTotal() может только возвращать значение для display метода с целочисленным типом. Общее количество подсчитывается во время выполнения отчета. Для печати что-то вроде этого <page>of <page total> вам необходимо использовать 3 поля.

## Шаблон секции

Шаблоны секции [section template] представлены в версии 3.0. Это может быть основой их редкого использования, и вы вряд ли найдете пример использования в стандартной базе. Шаблон секции должен быть основан на *table map*. Что такое Table maps объясняется в главе **Словарь Данных**. Поля map могут добавляться в отчет как поля. В случае, где ваш отчет использует похожие section blocks, выбор шаблона уменьшит написание одного и того же кода, это лучше, чем переделывать одинаковые block section во многих отчетах. Однако на практике, может быть проще создать отчет, используя X++ для модификации выхода, чем иметь два отчета, использующих section template. Отчет SalesInvoice – превосходный пример этого.

## 7.5 Дизайн

Расположение и тип полей в вашем дизайне обрабатывает MorphX. Все поля по умолчанию имеют авто расположение. Это означает, что расположение шрифтов и их размер по умолчанию берется из настроек пользователя **Сервис\Параметры** на закладке **Шрифты**.

Когда вы выбираете поле [controls] из списка полей, MorphX позиционирует поле, на основании информации из расширенного типа данных. Это очень полезно, если вам необходимо добавить поле в середину списка или если вы прячете поле, другие поля автоматически перераспределяются. Вам следует позволить MorphX авто располагать поля, однако когда поля должны быть фиксировано расположены, то вам необходимо поменять установку по умолчанию. Неудобство заключается в том, что при установке одного поля в ряду в фиксированное положение, необходимо менять фиксированное положение всех остальных полей. В общем, это не рекомендуется если только ваши поля должны быть закреплены в форме предварительной печати или жестко привязаны в каком-нибудь налоговом отчете или справке.

Если необходимо печатать поля, которые располагаются один ниже другого, в одной колонке имеет смысл использовать свойство **Model Fieldname**, (все поля отчета имеют это свойство). Позиция текущего поля прикрепляется к позиции поля со свойством ModelFieldName, если расположение текущих полей установлены в auto.

## Создание дизайна

Отчет может иметь более чем один дизайн. Под узлом Design, вы можете создать столько дизайнов, сколько необходимо. Это может быть нужно, если у вас есть форма и вы хотите распечатать отчет с различным выводом данных для каждого языка или групп покупателей. Отчеты с множественными дизайнами не часто встречаются в стандартной поставке. Вместо нескольких дизайнов для различного вывода данных используется X++, смотри отчет SalesInvoice. В отчете SalesInvoice, метод element.changeDesign() определяется, печатать поле или нет. Часто больше времени расходуется на поддержку различий во множественном

дизайне, чем управлением одним дизайном, с использованием X++. Управление заголовком секций с несколькими дизайнами утомительное занятие, так как это отнимает время на определение и проверку идентичности изменений во всех дизайнах

---

**Примечание:** Если вы создаете отчет подобный форме, который содержит предварительный вывод, необходимо настроить конечные настройки, использующие драйвер принтера для конечного вывода данных. Изменения в настройках принтера могут быть причиной изменения полей на странице. Часто вывод данных настраивается на конкретный принтер.

---

Дизайны могут создаваться, как auto дизайн или как generated дизайн (создаваемый вручную). Дизайн может так же состоять как из auto дизайна, так и из generated дизайна. В нашем случае используется только generated дизайн. Главное отличие между auto дизайном и generated дизайном в том, что auto дизайн использует все преимущества MorphX, допуская динамические шаблоны, авто заголовки и auto sums, устанавливаемые в критерии запроса. Generated дизайн статичен, и не может автоматически подстраивать изменения, сделанные в запросе или в шаблоне отчета. Рекомендуется использовать generated дизайн в случаях, где необходим фиксированный вывод. Generated дизайн в общем требуется, где вывод оговаривается контрактом или при использовании предопределенной формы (чеки, ордера на закупку и т.д.).

Generated дизайн имеет некоторые отличные секции для добавления заголовков к секциям body. За исключением этого auto дизайн и generated дизайн имеют одни и те же секции. Смотри **Рисунок 5, Секции дизайна отчета** для обзора секций в отчете дизайна.

Тип	Описание
Prolog	Это первая печатаемая секция. Prolog обычно используется для печати логотипа или заголовка на первом листе.
Page Header	Заголовок листа, печатается на верху каждого листа. Отчет может иметь более чем один заголовок.
Body	Секция body печатается после заголовка. Это секция с данными. Отчет обычно содержит секцию body для каждого источника данных.
Page Footer	Секция печатается внизу каждой страницы. Отчет может иметь более чем footer.
Epilog	Это последняя страница.
Programmable Section	Секция Programmable выполняется из кода. Этот тип секции используется при невозможности получения данных из запроса.
Section Template	Секция Templates используется для определения простых, часто повторяемых данных. Обычно используется в секции body. Секция основана на Map.

Header	Header используется в Generated дизайне как заголовок .
Section Group	В Generated дизайне, секция Body добавляется к Section Group.
Footer	Footer используется в Generated дизайне как footer для секции body.

Рисунок 36, секции дизайна отчета

Вы так же можете посмотреть приблизительный вид отчета, встав на узле под узлом *ReportDesign* , в контекстном меню по пункту *Просмотр*. Если отчет имеет комплексный дизайн как у *SalesInvoice*, тогда может быть затруднительно определить, как будет выглядеть результат.

У вас есть две опции для добавления полей в ваш дизайн, или использовать узлы дерева отчета, как в предыдущем примере, или использовать визуальный редактор. Визуальный редактор предоставляет возможность, как просмотра, так и правки дизайна. Для правки отчета с использованием дерева отчета – это двойной щелчок на узле дизайна; если вы хотите использовать визуальный редактор, тогда это контекстное меню, пункт *Правка*. При создании дизайна визуальным редактором предоставляются те же самые возможности, что и в АОТ. Используя визуальный редактор, вы можете менять свойства элемента отчета: добавлять или удалять поля. Для изменения отчета через визуальный редактор просто поставьте курсор и через контекстное меню выберите *Правка*, *Удаление* или *Новый*. Для изменения разметки встаньте на нее и через контекстное меню выберите сантиметры, дюймы или знаки.

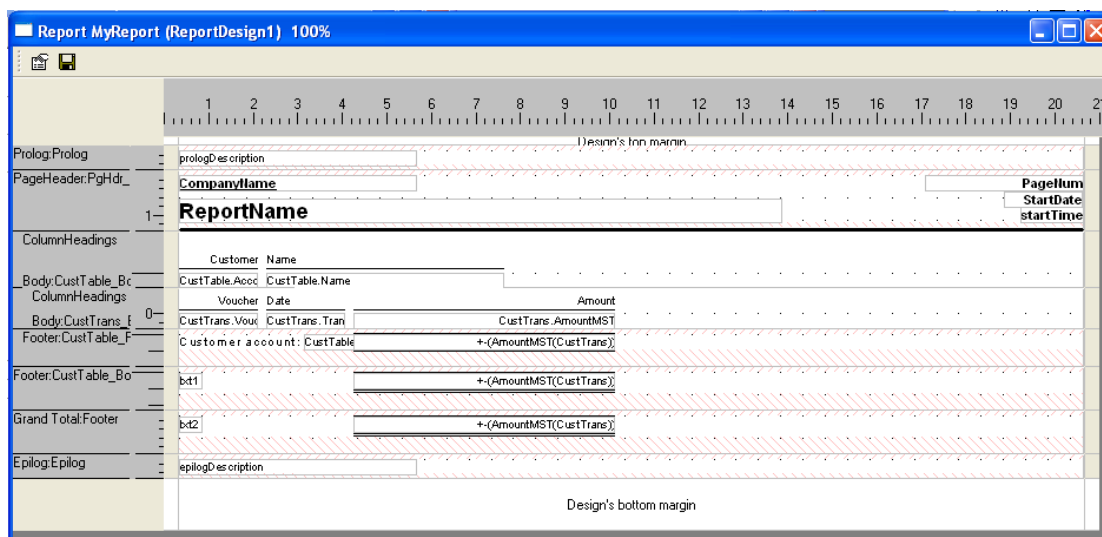


Рисунок 37: Визуальный редактор

На практике визуальный редактор лучше использовать для просмотра дизайна или для расположения или изменения свойства специфических полей. Визуальный редактор медленен и многие вещи можно сделать быстрее, используя дерево отчета.

## Авто дизайн

Самый обычный путь создания вывода отчета – это использовать авто дизайн [auto design]. При использовании авто дизайна, вам предоставляется возможность выбора шаблона отчета и полей, которые попадут на печать из запроса. MorphX отформатирует вывод. Если в отчете есть поля integer или real, пользователь имеет возможность выбрать суммирование.

Для быстрого создания вашего авто отчета в контекстном меню узла auto design выберете *Генерация Дизайна По Query*, секция body создается для каждого источника данных в запросе, и поля сортировки будут добавлены к дизайну.

Визуальный просмотр вашего отчета доступен через контекстное меню на узле auto design и выборе *Просмотр*. Попробуйте открыть MyReport в визуальном редакторе. Отметим, что визуальный редактор покажет секции из шаблона отчета, даже при условии, что секции шаблона не являются частью узлов отчета. Это очень полезная функция. Для правки отчета выберите в контекстном меню *Правка*. В правке участвуют только узлы, которые являются частью отчета.

Для использования авто суммы используйте возможность авто дизайна, это дает возможность пользователю определить подсуммы при выполнении отчета. Из диалога отчета вы можете установить подсуммы для любого поля или секции body. Это делает ваш отчет более гибким и сводит к минимуму программирование.

---

### Пример 4: Авто сумма

#### Элементы проекта MORPHXIT\_Reports

- Report, MyReport\_Sums
- Menu item output, MyReport\_Sums

Добавим в MyReport суммы по проводкам. Добавим также промежуточные суммы для каждого покупателя.

1. Продублируем отчет MyReport и переименуем в “MyReport\_Sums”.
2. Так как у индекса нет опции установки общего количества, то уберем из узла сортировки источника данных индекс AccountIdx CustTable, а вместо этого установите поле AccountNum. Установите свойство **AutoSum** в Yes для поля AccountNum. Каждый раз при изменении значения customer account изменяется и промежуточная сумма.
3. Поля для суммирования должны быть предопределены. В этом примере используется поле, с расширенным типом AmountMST из таблицы CustTrans. Найдите поле, печатающие AmountMST в body section CustTrans, Откройте лист свойств и установите **SumAll** в Yes.
4. Создайте новый пункт меню для отчета и запустите отчет. Для каждого покупателя будет выводиться промежуточная сумма.

5. В этом случае будут напечатаны только промежуточные суммы. Для добавления итоговой суммы отчета, закройте отчет и перейдите к АОТ еще раз. Перейдите к секции `body CustTable` и установите свойство **GrandTotal** в **Yes**. Секция `body` в `CustTrans` не имеет этого свойства, а только в `CustTable`, потому что это главный источник данных.
- 

Шаг 2 – это было определение промежуточных суммы в отчете, а у каких полей это делать можно? Дизайн определяет, какие поля суммируются, в этом примере использован `AmountMST`. Это рекомендуемая установка. Пользователь может добавить остальные опции самостоятельно. Сортируемые поля изменяют значение промежуточных сумм, а установка общей суммы определяется настройками по умолчанию и может быть изменена.

Узел *AutoDesignSpecs* имеет свойство **GrandTotal**. Это свойство позволяет выводить общую сумму для отчета с названием 'Итог', если установлено в **Yes**. Итоговая и общая сумма устанавливаются из диалога отчета или секции `body`, что дает тоже самый результат. Обе печатают общую сумму для всего отчета. Итак, если пользователь устанавливает общую сумму из диалога, вам не следует использовать итог.

Значение авто суммы может быть установлено использованием метода `element.sumControl()`. Для подсчета авто суммы поля `CustTrans.AmountMST` в примере выше вам следует рассмотреть следующий код:

```
element.sumControl(identifierstr(CustTrans_AmountMST), element.indent());
```

`Element.sumControl` возвращает суммируемое значение. Первый параметр в названии поля – имя суммируемого поля. Системная функция `identifierstr()` используется для проверки и вывода предупреждения. `element.indent()` всегда используется, как второй параметр, для установки корректного уровня.

Как вы заметили, есть свойство называемое авто заголовок. Оно используется также образом, как и авто сумма. Вместо печати общих итогов, заголовок будет выводиться каждый раз при разрыве по сортируемым полям. Пользователь может установить авто заголовок во время выполнения, но если необходимо, вы можете установить видимость авто заголовка. И авто сумма и авто заголовок – возможности, реализуемые только в авто дизайне.

Все упоминаемые секции вызываются или структурой отчета или запросом отчета. У вас могут быть ситуации, где необходимо вызвать секции отчета вручную. В этом случае лучше использовать `programmable sections`, которая выполняется из `X++`.

---

#### Пример 5: Секция Programmable

##### Элементы проекта MORPHXIT Reports

- Report, `MyReport_ProgSec`
- Menu item output, `MyReport_ProgSec`



Добавим секцию programmable в MyReport. Для простоты будем выводить только текст.

1. Продублируйте MyReport и переименуйте в новый отчет «MyReport\_ProgSec». Перейдите к узлу *AutoDesignSpecs* и в контекстном меню выберите создать *ProgrammableSection*.
2. Откройте лист свойств новой секции programmable и выберите свойство **ControlNumber**. Оно используется для ссылки на секцию из X++. Установите свойство в 10.
3. Теперь добавьте поле в programmable section. В контекстном меню секции programmable выберите создать *Control* для добавления text control. Перейдите к text control и определите свойство **Text** в 'Header for customers'.
4. Теперь определите выполнение секции programmable. Перейдите к *MyReport\_ProgSec/Methods*, в контекстном меню выбираете *Перекрыть Метод* и выбираете *init()*. Метод *init()* должен выглядеть так:

```
Public void init()
{
    super();

    element.execute(10);
}
```

5. Создайте пункт меню для отчета MyReport\_ProgSec. При выполнении отчета MyReport\_ProgSec текст 'Header for customers' будет напечатан перед данными из запроса.

В примере вы изменили номер programmable section в 10. Рекомендуется, чтобы вы оставляли промежутки в последовательности номеров у programmable sections. Этим в дальнейшем вы сможете добавлять новые programmable section и сохраните логическую последовательность.

Выполнение секции programmable можете вызывать из X++ где угодно. Однако использование в комбинации с авто суммой, необходимо учитывать следующее: предположим, что вы хотите напечатать programmable section перед секцией body, тогда логичнее написать ваш код в методе *executeSection()* как раз перед *super()* в секции body. Это сделает возможным печать вашей секции programmable перед секцией body, но секция programmable так же будет выполнена перед каждой авто суммой. Во время выполнения MorphX обращается с авто суммой как с footer, и это является причиной того, что секция body section будет выполнена снова. Решение этой проблемы – использование метода отчета *header()* или *footer()*. Теперь вы сможете сказать, какая секция body будет выполняться. При выполнении секции body, в параметрах *\_tableId* и *\_fieldId* содержатся значения текущего поля.

```
public void header(ReportSection _headerSection, tableId _tableId, fieldId _fieldId)
{
```

```
if (_tableId == tableNum(custTable) && _fieldId)
    element.execute(10);

super(_headerSection, _tableId, _fieldId);
}
```

Это пример показывает, как использовать метода `header()`. Сделана проверка, что бы убедиться, что секция `body` из выводимой таблицы клиентов. Если так, тогда секция `programmable` выполняется. Другой путь использования методов `header()` и `footer()` может быть в добавлении новой страницы после суммы.

---

**Примечание:** Секция `programmable` часто используется для вставки разделителя подобно чистым рядам или линиям. Сделать это можно установкой свойства секции `programmable`. Вам необходимо добавить "dummy" (пустое) поле в вашу секцию `programmable`, если секция `programmable` не имеет ни одного поля, ничего не напечатается.

---

## Генерируемый дизайн

Использование генерируемого дизайна [generated design] на первый взгляд проще, чем авто дизайна, так как вы имеет много секций, которые определяют построение вашего отчета и все секции видимы. Однако генерируемый дизайн более статичен, дизайн основан на настройках из запроса и свойств, определенных в узлах дизайна. Недостаток в том что, когда вы выбираете шаблон отчета [report template] для вашего отчета и затем решаете изменить его или просто выбрать другой шаблон, то не произойдет автоматического обновления дизайна. И пользователь не сможет выбрать суммирование при исполнении отчета.

Познакомимся поподробнее с генерируемым дизайном и создадим его в MyReport. Выберем в контекстном меню узла *Designs/AutoDesign1* MyReport *Generate Design*. Новый узел называется *Generated Design*, и располагается под *Designs/AutoDesign1*. Если вы развернете генерируемый дизайн, то заметите, что дизайн похож на авто дизайн. Отличие в том, что секции из отчета и секции рассчитываемых общих итогов уже добавлены.

Если вам необходимо посмотреть отчета с авто дизайном, тогда опция создания генерируемого дизайна на основании авто дизайна сразу приходит в голову, так как все секции, основанные на вашем шаблоне и все авто суммы сохраняться.

## Поля в дизайне

Самый простой способ добавить поля в дизайн – это перетащить их или `display` методы из источника данных или из таблицы. Когда перетаскиваете поля или `display` методы такие, как `string`, `enum`, `integer`, `real` date and time, MorphX автоматически создает поле того же типа в дизайне. Такие поля как `prompt`, `shape`, `sum` или группа полей используются для более специфических целей и должны добавляться вручную. Вы, конечно, можете добавить все типы полей вручную, но это займет много времени, лучше перетаскивайте поля так, как MorphX авто

позиционирует поле и заполнит необходимые свойства ссылками полей или `display` методов.

Авто дизайн и генерируемый дизайн имеют то же самый набор полей. Для просмотра доступных полей смотри **Рисунок 7, поля в Отчете**.

Имя	Описание
String	Используется для строковых величин. Если печатать поля методом, свойство <code>DynamicHeight</code> авто регулирует высоту, согласно номеру печатаемых рядов.
Enum	Используется для печати значений типа <code>enums</code> .
Integer	Используются для <code>integer</code> значений.
Real	Используются для <code>real</code> значений.
Date	Используется для печати дат. Даты форматируются согласно региональной настройке Windows
Time	Используется для печати времени. Время форматируется согласно региональной настройке Windows
Text	Используется для печати фиксированного текста. Если текст должен содержать динамическое изменяемое значения, тогда оптимально использовать <code>display</code> метод.
Prompt	Prompt добавляет текст со следующими точками и двоеточиями.
Shape	Рисует <code>box</code> . Размер и расположение определяется свойствами. Может использоваться для вывода формулы.
Bitmap	Используется для печати графики. Введите путь к рисунку, ссылку на контейнер и используемый ресурс. Пример использования <code>bitmap</code> смотрите в отчете <i>HRMAplicantStatus</i> .
Sum	Используется для печати итоговых сумм. С суммами в авто дизайне может использоваться для вывода в секции <code>programmable</code> . Пример использования в авто дизайне смотрите в отчете <i>SalesLinesExtended</i> .
Field group	Используется для добавления групп полей. Группа полей в отчете автоматически обновляет изменения, сделанные в АОТ

**Рисунок 38, поля в отчете**

Поля `Bitmap` имеет некоторую особенность в использовании. Есть несколько путей для настройки `bitmaps`. Вы можете использовать иконки из Ахарт. Кроме того, можно включить `id` ресурса в лист свойств или использовать `display` метод, возвращающий `id` ресурса. Получить просмотр доступных ресурсов можно, используя отчет `tutorial_Resources`, который печатает `id` ресурса и соответствующие иконы. Использование других опций это указание пути, где

находится bitmap или ссылка на контейнер с bitmap. Когда указываете путь, помните об использовании двойной наклонной черты.

Поля суммы в генерируемом дизайне используйте в секции footer. При использовании авто дизайна sum control, помещайте в секцию programmable.

---

**Примечание:** При печати отчета в infolog может появиться сообщение, что отчет подгоняется к параметрам страницы. Это причина использования большого количества колонок в вашем дизайне. Установите свойство FitToPage в No в узле *design* для деактивирования этой функции.

---

При добавлении полей в дизайн, следует проверять, чтобы поле имело ссылку на источник данных, тогда секция печатается. В нашем примере MyReport, вы не сможете печатать поля из секции body CustTable и CustTrans в заголовке секции, потому что при обработке заголовка страницы MorphX не выводит информацию, не связанную с секцией body. То же самое происходит при добавлении поля в секцию body CustTable со ссылкой на поле из CustTrans. Это так же дает ошибку, когда CustTable выводится пред CustTrans.

## 7.6 Методы Отчета

Вы можете создать простой отчет такой, как список номенклатуры без использования X++ просто используя возможности генератора отчета. Для более продвинутых отчетов, которые фильтруют данные или требуют специфической сортировки, вам необходимо перекрывать методы отчета, используя X++. Для просмотра методов отчета смотри **Рисунок 8: Методы отчета**. Методы запроса описаны в главе запросы, смотри **Запросы**.

Часто используются системные классы при изменении отчета во время выполнения. Это основные компоненты отчета, они позволяют программисту переопределить все аспекты во время выполнения. Действительно, вы можете создать отчет на пустом месте, только используя системные классы. Для дальнейшей информации о системных классах смотри главу **Классы**.

Имя	Параметры	Описание
CallMenuFunction	MenuFunction _menuFunction	Web метод.
Caption	str _reportSpelling, str _reportName, str _designCaption, str _designName	Устанавливает заголовок отчета. Параметры _reportSpelling и _designCaption устанавливают заголовок в Выводе на Экран.
CreateProgressForm		Перекрывается стандартная форма прогресса при создании страниц отчета. Метод дает опции для создания собственной формы прогресса.
Dialog	Object _dialog	Dialog() используется для добавления полей в диалог отчета. Структура runbase отчета вызывает диалог при выполнении отчета.

		Смотрите отчет <i>KMAction</i> .
Fetch		Этот метод – это двигатель отчета. Fetch() открывает пользовательский диалог, выбирает записи из базы данных выполнением запроса и посылает записи на печать. Этот метод, как правило, перегружен, когда данные не могут быть получены из запроса. В качестве примера может служить отчет <i>HRMCourseSkills</i> .
Footer	ReportSection _footerSection, tableId _tableId, fieldId _fieldId	Метод вызывается каждый раз, при выполнении секции в дизайне. В случае если авто сумма не часть дизайна, тогда предоставляется возможность выполнения кода перед или после печати авто суммы.
GetTarget		Возвращает выбранную среду печати.
Header	ReportSection _headerSection, tableId _tableId, fieldId _fieldId	Метод вызывается каждый раз, при выполнении секции в дизайне. В случае если авто сумма не часть дизайна, тогда предоставляется возможность выполнения кода перед или после печати авто суммы.
Init		Это первый вызываемый метод. Метод инициализирует отчет. Переменные объектов, объявленных в отчете, обычно инициализируются здесь. Смотрите отчет <i>salesFreightSlip</i> .
New	anytype _argsOrReportOrContainer, str _designName, boolean _isWebReport=FALSE	Используется для инициализации объекта reportRun. Это обычно не делается из генератора отчета. Типичный случай использования, если отчет инициализируется из X++.
Pack		Этот метод используется для хранения последних значений. Используется в связке с unpack(), загружает последние хранимые значения. Однако unpack() не базовый метод. При добавлении нового поля диалога, pack() загружает хранимые значения из формы диалога. Смотрите отчет <i>KMAction</i> .
PageFormatting		Этот метод не используется более. В версии 3.0 метод используется PrintJobSettings.PageFormatting.
Print		Print() возвращает число страниц на печать. Метод может использоваться для проверки печатать или нет какие-то

		страницы. Смотри отчет <i>projTimeSheetEmpl</i> .
PrinterSettings	int _showWhat=-1	Используется для активации различных частей диалога печати. Метод вызывается, если структура runbase отчета не используется, а вызывается из метода prompt().
ProgressInfo	int _pageNo, int _lineNo	ProgressInfo выполняется для каждой строчки на форме.
Prompt	boolean _enableCopy=TRUE, boolean _enablePages=TRUE, boolean _enableDevice=TRUE, boolean _enableProperties=TRUE, boolean _enablePrintTo=TRUE	До версии 3.0 prompt() обрабатывал диалог отчета. Теперь используется метод dialog(). Метод не может использоваться в комбинации со структурой runbase отчета, так как структура отвергнет установки. Вместо этого необходимо использовать класс PrintJobSettings.
Run		Run() вызывается при нажатии ОК в диалоге. Run() выполняет следующие шаги: <ul style="list-style-type: none"> <li>• Если не существует генерируемого дизайна, а дизайн создается на лету по авто дизайну.</li> <li>• Вызывает fetch()</li> <li>• Вызывает print()</li> </ul> Метод может использоваться для добавления критерия в отчет после Based On установки в диалоге. Смотрите отчет <i>ReqPO</i> .
Send		Send() связан с fetch(). Fetch() перебирает записи из запроса, а send() посылает их в дизайн. Метод может быть перекрыт, если необходимо решить выводить или нет записи на печать. Смотрите отчет <i>CustTransList</i> .
SetTarget	PrintMedium _target	Устанавливает media для отчета
ShowMenuFunction	MenuFunction _menuFunction	Web метод
ShowMenuReference	WebMenu _menuReference	Web метод
Title	str _title=""	Не используется более. Может переопределить заголовок, при выводе, если выполняется из fetch().
ExecuteSection		Каждая секция в дизайне имеет метод executeSection, который используется для печати секции. Метод может использоваться для принятия решения выводить или

		нет секцию на печать. Смотрите отчет <i>CustCollectionJour</i> .
--	--	--

Рисунок 39: Методы Отчета

Предыдущие части этой главы были сфокусированы на индивидуальных элементах, составляющих отчет. Теперь самое время погрузиться в использование X++ для модификации отчета.

## Структура RunBase отчета

Вы, наверное, задавались вопросом, почему иногда получаете различные диалоги при выполнении отчета в Ахapta. Если отчет выполняется из AOT, вы сначала получаете диалог Запроса, а затем диалог Печати. Когда отчет выполняется через пункт меню, то вы получаете только один диалог. В версии 3.0 Ахapta был введен новый класс RunbaseReportStd. Если отчет вызывается через пункт меню, то RunbaseReportStd вызывается из класса SysReportRun.

Структура Runbase отчета является источником смущения новичков в программировании. Это необходимо понимать, что отчет может вызываться одним из четырех путей:

Напрямую из узла отчетов в AOT.

Через пункт меню так же, как из пользовательского меню или напрямую из AOT.

Через класс наследник RunBaseReport.

Вызываться напрямую через X++.

Класс RunbaseReportStd вызывается структурой Runbase отчета только если ваш отчет не вызывается из класса наследника RunBaseReport. Однако когда отчет выполняется напрямую из узла *Reports*, структура Runbase отчета не исполняется, и показываются два диалога.

---

**Примечание:** Отчеты часто используются для проверки целостности данных в системе. В отчете можно обновлять данные, но правила хорошего тона программирования [best practice] не рекомендуют производить запись в базу данных. Если ваш отчет должен обновлять или вставлять записи, то вам следует рассмотреть создание класса для таких манипуляций и вызывать его из класса наследника RunBaseReport.

---

Правильно всегда создавать свой пункт меню, вызывающий отчет, так вы будете иметь тот же диалог, что и пользователь при вызове отчета из пользовательского меню. Помните, что отчет, выполняемый из класса должен всегда быть наследником класса RunBaseReport. Класс RunbaseReportStd только использует внутреннюю структуру RunBase отчета. Для дальнейшей информации о классе RunBase, смотрите главу **Классы**.



Рисунок 40: Классы Runbase отчета

При создании отчета в предыдущей версии Ахapta, это было правилом, что отчет должен вызываться из наследника класса RunBaseReport. Это было сделано для объединения двух упомянутых диалогов отчета и возможности запуска пакета. Это так же обеспечивает лучшую производительность при выполнении класса на сервере. С введением класса RunBaseReportStd, только отчеты с тяжелыми данными должны быть наследниками класса RunBaseReport.

---

#### Пример 6: Отчет runbase

##### Элементы проекта MORPHXIT\_Reports

- Класс, SalesReport\_DailyEntries
- Отчет, SalesDailyEntries
- Menu item output, SalesReport\_DailyEntries

Для исследования структуры Report Runbase, перейдем к узлу *Classes* и выделим класс SalesReport\_DailyEntries. Этот класс наследуется от класса RunBaseReport и поэтому имеет возможность вызова отчета. Этот стандартный класс отчета названный с префиксом модуля. Вы можете увидеть много подобных классов.

SalesReport\_DailyEntries вызывает отчет SalesDailyEntries. Этот класс не так уж нужен с версии Ахapta v3.0 так, как логика может быть обработана и классом RunBaseReportStd. Однако класс SalesReport\_DailyEntries – хороший пример построения класса. Этот класс имеет следующие методы:

```
public identifiername lastValueElementName()
{
    return reportstr(SalesDailyEntries);
}
```

Это определение вызываемого отчета, метод должен быть перегружен. Функция reportstr() проверяет, что введено правильное имя отчета.

```
client server public static ClassDescription description()
{
    return "@SYS77491";
}
```

Этот метод определяет заголовок диалога.

```
static void main(Args args)
{
    SalesReport_DailyEntries salesReport_DailyEntries;
    ;
    salesReport_DailyEntries = new salesReport_DailyEntries();

    if (salesReport_DailyEntries.prompt())
    {
        salesReport_DailyEntries.run();
    }
}
```

Метод main() – статический метод, который инициализирует класс. Это позволяет запускать его через пункт меню. В классе сделана проверка вызываемого диалога. Если кнопка ОК нажата в диалоге, выполняется отчет.



Для использования класса в отчете, определяется переменная класса salesReport\_DailyEntries в class declaration отчета SalesDailyEntries:

```
public class ReportRun extends ObjectRun
{
    SalesReport_DailyEntries salesReport_DailyEntries;
}
```

```
public void init()
{
    super();

    salesReport_DailyEntries = element.args().caller();

    if (!salesReport_DailyEntries)
    {
        throw error(Error::missingRecord(funcName()));
    }
}
```

Сущность класса SalesReport\_DailyEntries определяется через args().caller() в отчете. Делается проверка для определения класса, вызывающего отчет. Это сделано для предотвращения вызова отчета напрямую через X++ или AOT. В этом примере не имеет значения, какой источник у отчета, но в некоторых случаях класс может фильтровать данные на печать.

Как было упомянуто выше класс SalesReport\_DailyEntries, не так уж нужен с появлением класса RunBaseReportStd, который использует логику диалога запрос/печать и автоматически может выполнять пакетную обработку. Для модификации отчета, что бы не использовать класс SalesReport\_DailyEntries, измените метод init() отчета. В нашем случае просто удалите init(). Пункт меню все еще ссылается на класс, поэтому необходимо перейти к output menu item SalesReport\_DailyEntries и изменить свойство пункта меню, который вызывает отчет. При выполнении отчета напрямую из AOT, вы получите тот же самый результат, как если бы использовали класс SalesReport\_DailyEntries. Если выполнить отчет из menu, то вы увидите консолидированный диалог на экране.

---

#### Пример 7: Диалог отчета

##### Элементы проекта MORPHXIT\_Reports

- Report, SalesDailyEntries
- Menu item output, SalesDailyEntries\_Without\_Class

Теперь самое время добавить некоторые возможности к отчету SalesDailyEntries. Добавим поле диалога для определения должен ли печататься отчет. Значение нового поля будет храниться как последнее значение при выполнении отчета. Добавим следующее в Class Declaration отчета:

```
public class ReportRun extends ObjectRun
{
    DialogField dialogPrintDetails;
```

```

Boolean    printDetails;

#define.CurrentVersion(1)
#define.LOCALMACRO.CurrentList
    printDetails
#define.ENDMACRO
}

```

DialogPrintDetails – переменная класса DialogField и используется для создания нового поля диалога, показываемого пользователю при запуске отчета. Переменная printDetails хранит значение поля диалога. Макрос CurrentList – список переменных, необходимых для хранения. Список обычно содержит переменные каждого поля. В этом примере CurrentList содержит только одну переменную. Для добавления прочих переменных, просто отделите их запятой. CurrentVersion содержит версию CurrentList. Ахapta сохраняет параметры пакетной обработки при переходе к следующей обработке. Система затем загружает параметры, которые пользователь использовал в этот раз. Если сделаны изменения в CurrentList, то CurrentVersion следует увеличить на единичку. Можно переустановить сохраненные данные, использованные в предыдущей обработке, но это также приведет к изменению других значений таких, как критерий выборки, установленный пользователем.

```

public Object dialog(DialogRunbase _dialog = null)
{
    DialogRunBase dialog;
;

    dialog                = super(_dialog);
    dialogPrintDetails    = dialog.addFieldValue(typeId(NoYesId), printDetails, "Print details",
                                                "Print additional information for the transactions.");

    return dialog;
}

```

Метод диалога, (не удивляйтесь) определяет диалог. Диалог иницируется из super() и содержит диалог отчета по умолчанию. Единственную вещь, которую добавим – это новое поле возможности печати. Новое поле автоматически добавляется в группу полей по умолчанию называемое Парметры.

```

public boolean getFromDialog()
{
    boolean ret;

    printDetails    = dialogPrintDetails.value();
    ret              = true;

    return ret;
}

```

При нажатии ОК в экранном диалоге, система автоматически вызывает метод getFromDialog. Значение нового поля диалога хранится в переменной printDetails.

```

public container pack()
{
    return [#CurrentVersion, #CurrentList];
}

```

В этом методе сохраняется последнее значение поля из нового диалога. Pack() сохраняет текущее значение CurrentVersion и CurrentList определенные в ClassDeclaration.

```
public boolean unpack(container packedClass)
{
    boolean    ret;
    Integer    version = conpeek(packedClass,1);

    switch (version)
    {
        case #CurrentVersion:
            [version, #CurrentList] = packedClass;
            ret = true;
            break;
        default:
            ret = false;
    }
    return ret;
}
```

Этот метод загружает последнее значение, хранимое в CurrentVersion и CurrentList.

```
public void run()
{
    if (printDetails)
    {
        SalesLine_Name.visible(true);
    }
    else
    {
        SalesLine_Name.visible(false);
    }

    super();
}
```

Последний шаг – проверить выводить детали или нет. В этом примере – это SalesLine.name. В этом отчете поля должны быть определены пред тем, как на них ссылаются из X++. Вам следует изменить свойства. Видимость (обращение к полю из X++) поля отчета определяются установкой свойства **AutoDeclaration** в Yes. В рассмотренном примере вам необходимо установить свойство AutoDeclaration для поля SalesLine\_Name в Yes.

---

В нашем примере диалога также были использованы методы отчета pack() и unpack(). Этими методами сохраняются последние значения полей диалога и переводятся в пользовательские параметры, определенные в форме диалога с клиента на сервер, где будет осуществляться пакетная обработка. Если вам необходима эта функциональность в диалоге, вы можете просто копировать эти два метода из существующего класса и CurrentVersion и CurrentList в ClassDeclaration.

## Динамические отчеты

Возможность программного изменения отчета во время выполнения очень полезна, так это сопровождается опцией изменения свойств или добавлением новых элементов. Итак, вы можете создать один отчет, вместо нескольких с одинаковым дизайном. Отчет SalesInvoice – как раз такой пример. Отчет SalesInvoice печатается с различной видимостью полей, в зависимости от установленных параметров.

Так как Ахарта – это мульти языковая система, становится возможным печатать отчеты на различных языках. В большинстве случаев, Ахарта применяет это корректно без дополнительного программирования. По умолчанию, отчет печатается на языке пользователя Ахарта. Однако исключения случаются при некоторых обстоятельствах. Например, отчет должен быть распечатан на языке покупателя. Это делается установкой свойства **Language** узла отчета *ReportDesign*. Вы можете установить свойство в фиксированное значение, но лучший путь установить язык через X++. Здесь myTable.languageId используется для установки языка в дизайне:

```
public void init()
{
    super();

    element.design().languageId(myTable.languageId);
}
```

Ключевое слово element используется для ссылки на все элементы отчета. Здесь element применяется для ссылки на метод languageId() дизайна. Таким образом, вы можете получить доступ к любому элементу отчета. Используя element для ссылки на элемент отчета, путь ссылки может быть длинным, как показано здесь:

```
element.design().sectionGroup(tablenum(CustInterestTrans)).section(ReportBlockType::BODY).control
Name('custInterestTrans_custInterestJourInterestAmount');
```

Верхняя строка взята из метода init() отчета *CustInterestNote*. Лучший путь в такой ситуации использовать свойство **AutoDeclaration**. При использовании свойства AutoDeclaration в сущности вы определяете элемент системного класса. Поле SalesLine\_Name, рассмотренное нами в примере диалога, - сущность системного класса ReportStringControl.

---

**Примечание:** просматривая стандартные отчеты в АОТ, вы увидите определения системных классов для секций и полей отчета в коде чаще, чем использование свойства AutoDeclaration. Это потому, что свойство AutoDeclaration отчета было добавлено только в версии v3.0 Ахарта.

---

Системные классы используются при программном добавлении элементов таких, как секции или поля в отчет во время выполнения. Например, при определении объекта, вызвавшего ваш отчет, вы решаете печатать дополнительную информацию или даже печатать дополнительную секцию из другой таблицы. Вместо применения системного класса для создания полей, можно стандартно добавить секции или поля, а затем использовать свойство видимости для определения показывать поле или нет. В некоторых случаях использование

системных классов предпочтительнее. Используя системные классы для создания полей, вы можете не торопиться с определением типа необходимого поля. Это особенно ценно, когда необходимые поля для добавления, зависят от настроек пользователя или параметров.

---

#### Пример 8: Системные классы отчета

##### Элементы проекта MORPHXIT Reports

- Report, MyReport\_SystemClasses
- Menu item output, MyReport\_SystemClasses

Используем системные классы и создадим новый отчет, как показано на рисунке **Рисунок 10: Тест системных классов отчета**. Отчет создаст секцию body для CustTable. Секция body будет содержать 10 полей, распечатывающих первые 10 записей из CustTable. Добавьте таблицу CustTable в источник данных и создайте авто дизайн.

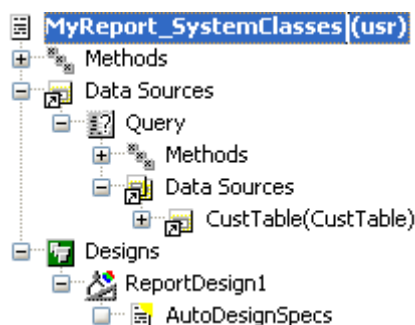


Рисунок 41: Тест системных классов отчета

Перепишем метод init() отчета как показано здесь. Никакого другого кода не требуется:

```
public void init()
{
    ReportSection    reportSection;
    DictTable        dictTable;
    DictField        dictField;
    Counter          fieldCounter;

    super();

    reportSection = element.design().autoDesignSpecs().addSection(ReportBlockType::Body);
    reportSection.table(tableNum(custTable));

    dictTable = new DictTable(tableNum(custTable));

    while (fieldCounter < 10)
    {
        fieldCounter++;
        dictField = new DictField(dictTable.id(), dictTable.fieldCnt2Id(fieldCounter));
        reportSection.addControl(dictTable.id(), dictTable.fieldCnt2Id(fieldCounter));
    }
}
```

Сущность системного класса `reportSection` и `reportControl` используются для создания секции `body` и связанных `controls`. Новая секция отчета типа `Body Section` добавляется в узел *AutoDesignSpecs* и определяется связь с таблицей `CustTable`. `DictTable` – это тоже сущность системного класса. Сущность `DictTable` часто используется для обращения к таблице или свойствам полей таблицы. Таким образом, `DictField` – системный класс, позволяющий обращаться к определенным полям определенной таблицы. Здесь `dictField` и `DictTable` используются для обращения к первым 10 записям из `CustTable`. При каждом обращении добавляется поле к секции `body`.

---

MorphX добавляет соответственный тип поля и автоматически располагает новые поля, так при выполнении отчета вы получите первые десять записей из таблицы `CustTable`, напечатанные подряд. Если вы создаете модуль, где пользователь имеет опцию определения своего собственного вывода на печать в отчете, тогда системные классы будут достойным ответом.

### Последовательность методов отчета

При производстве модификаций в отчете, вам приходится переписывать существующие методы или добавлять новые методы, которые вызываются из перекрытых методов. Выполнение следующих методов происходит по порядку при загрузке отчета:

`init()` ► `dialog()` ► `run()` ► `fetch()` ► `send()` ► `print()`

1. `Init()` и `dialog()` запускаются при загрузке отчета.
2. `Run()` запускаются при нажатии клавиши ОК в диалоге.
3. `Fetch()` организует цикл по запросу и для каждой найденной записи запускает `send()`.
4. Наконец, запускается `print()`.

Эти методы очень важны в отчете и некоторые вы будете часто перегружать. Типичные настройки включают в себя добавление полей в диалоге, управление выводом из запроса перед печатью или отладка вывода секций `programmable`, которые должны печататься в секциях `body`.

Верхний порядок выполнения используется при работе структуры `RunBase` отчета. Если вы вызываете ваш отчет прямо из AOT без использования пункта меню, порядок выполнения методов немного меняется, как показано ниже:

`init()` ► `run()` ► `prompt()` ► `fetch()` ► `send()` ► `print()`

Отметим, что `dialog()` не вызывается. `RunBaseReportStd` поля диалога отчета и когда структура `runbase` “не активна” используется метод `prompt()`.

---

#### Пример 9: Отчет с перекрытым методом `fetch()`

#### Элементы проекта MORPHXIT Reports

- Report, MyReport\_Fetch
- Menu item output, MyReport\_Fetch

Перекроем метод fetch() в новом отчете для вывода проводок каждого покупателя, отфильтрованных по дате, от n-дней до текущей системной даты. Диалог отчета будет иметь номер n-дней, определяемый пользователем. Сделайте копию отчета MyReport. В итоге отчет должен выглядеть, как показано на **Рисунке 11: Отчет с перекрытым методом fetch()**. Пример сфокусирован на перекрытии методов.

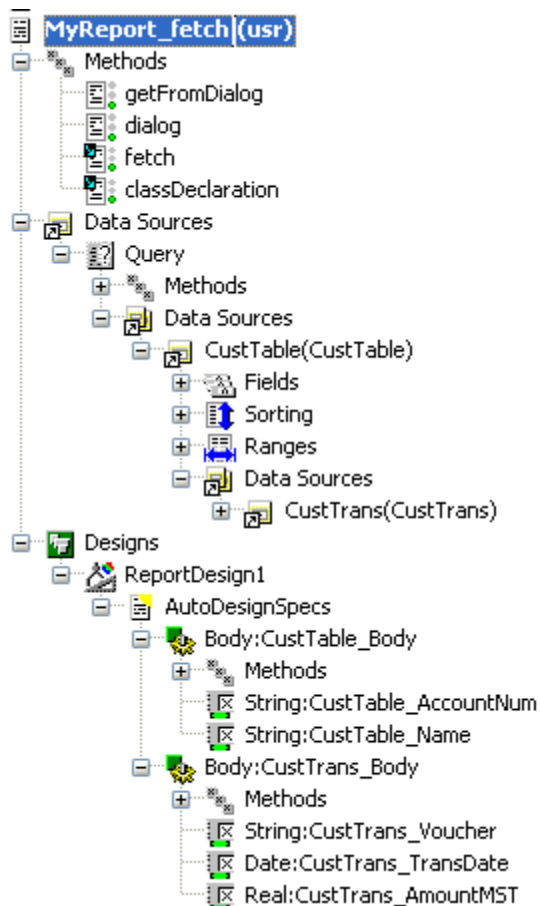


Рисунок 42: Отчет с перекрытым методом fetch()

Собственно вы создали запрос и дизайн отчета теперь создадим методы отчета.

```
public class ReportRun extends ObjectRun
{
    DialogField    dialogDaysBack;
    NumberOf      daysBack;
}
```

Переменная dialogDaysBack необходима для диалога отчета. Значение будет храниться в переменной daysBack.

```
public Object dialog(Object _dialog)
{
    DialogRunBase dialog;
    ;

    dialog                = super(_dialog);
```

```

        dialogDaysBack      = dialog.addFieldValue(typeId(NumberOf), daysBack, "Number of days",
                                                    "Number of days back to be printed.");

        return dialog;
    }

```

Добавляемое поле в диалоге служит для ввода n-1 дней.

```

public boolean getFromDialog()
{
    boolean ret;

    daysBack      = dialogDaysBack.value();
    ret           = true;

    return ret;
}

```

Переменная daysBack установлена для хранения значения из диалога.

```

public boolean fetch()
{
    QueryRun      qr;
    QueryBuildRange rangeTransDate;
    Boolean       ret;

    qr = new QueryRun(element);

    rangeTransDate =
        element.query().dataSourceTable(tablenum(CustTrans)).addRange(fieldnum(CustTrans, transDate));
    rangeTransDate.value(queryRange(systemdateGet()-daysBack, systemDateGet()));
    rangeTransDate.status(RangeStatus::LOCKED);

    element.design().caption(strfmt("%1, %2", element.design().caption(), rangeTransDate.value()));

    if (qr.prompt() && element.prompt())
    {
        while (qr.next())
        {
            custTable = qr.get(tableNum(CustTable));
            custTrans = qr.get(tableNum(CustTrans));

            if (!custTable)
            {
                ret = false;
                break;
            }

            if (qr.changed(tableNum(custTable)))
            {
                element.send(custTable, 1);
            }

            if (qr.changed(tableNum(custTrans)))
            {
                element.send(custTrans, 2);
            }
        }
        ret = true;
    }
}

```



```

    }
    else
        ret = false;

    return ret;
}

```

Переменная `daysBack` содержит значение, которое вводит пользователь. Необходимо ввести критерий в запрос для фильтрации транзакции с `n-1` дня по системную дату. Объект `QueryRun`, определенный как `qr`, инициализируется с активацией запроса, и затем добавляется критерий для даты проводки покупателя. Критерий блокируется так, что пользователь не может его изменить. Критерий даты проводки добавляется к заголовку отчета.

С этой точки запускается цикл по запросу. Стандартный цикл выборки и вывода на печать записей – это то, что вызывает `super()` в `fetch()`. Перед выборкой из запроса есть проверка вызова диалога запроса и отчета. Эти два диалога обернуты в `RunBaseReportStd`. В каждом цикле инициализируются таблицы `CustTable` и `CustTrans`. Если записей не обнаружено, цикл прерывается, и отчет не запускается. Если изменился источник данных и новая запись обнаруживается, тогда она выводится на печать, используя метод `send()`. Отметим второй параметр в методе `send()`. Второй параметр определяет уровень записи. Запись из `CustTable` будет выводиться на первом уровне, а `CustTrans` на втором уровне. Важно, если не установить это корректно, авто суммы не напечатаются.

---

В примере `fetch`, запрос отчета выбирается в цикле. В выборке можно использовать инструкцию `WHILE`, `SELECT`, или комбинацию обоих. Для каждой записи выбираемой в запросе, вы можете выбрать запись из таблицы не входящей в запрос, или даже построить временную таблицу и вывести её на печать. Для каждой выводимой записи всё, что вам необходимо сделать – это вызвать метод `send()`.

Если вам необходимо проверить, какие записи выводить, перегрузите метод `send()` вместо метода `fetch()`:

```

public boolean send(Common _cursor, int _level=1, boolean _triggerOffBody=TRUE, boolean
    _newPageBeforeBody=FALSE)
{
    boolean    ret;
    CustTrans  custTrans;

    if (_cursor.tableId == custTrans.tableId)
    {
        custTrans = _cursor;
    }

    if (custTrans.transDate == systemDateGet())
    {
        ret = super(_cursor, _level, _triggerOffBody, _newPageBeforeBody);
    }

    return ret;
}

```

Метод `send()` имеет в качестве параметра запись, которая должны выводиться. Всё, что необходимо сделать – это инициализировать соответствующую таблицу. В нашем случае инициализируется таблица `CustTrans`, если переданный параметр – запись `CustTrans`. Будут выводиться только проводки с датой равной системной.

Добавление значений критериев в запросе часто может просто определить, перекрыв метод `init()`, который значительно проще так, как требует добавление только нескольких строчек кода. Критерий добавляется к запросу в методе `fetch` и зависит от значения, введенного пользователем. Модификация происходит после закрытия диалога, поэтому код был помещен в `fetch()`. Перед добавлением критерия в запрос, вам необходимо всегда проверять, содержит ли уже запрос критерий для поля, используя метод `QueryBuildRange.findRange()`. Если создано два критерия для одного и тоже поля, критерии будут соединены по условию “AND-ed”, что выдаст неожиданные результаты. Пользователь имеет опцию ввода критерия в диалоге отчета, при посылке на печать. Критерии добавляются к источнику запроса так же, как добавляются из X++; однако, если метод `fetch()` перекрыт, то эта опция не доступна.

## 7.7 Специальные Отчеты

Этот раздел фокусирован на базовых шагах создания отчета. Вы получите представление об опциях MorphX, рассмотрим примеры обращения со специальными отчетами и возможности, что бы ваш отчет имел более наглядный интерфейс.

### Выполнение отчета из X++

Отчеты обычно активируются через пункт меню, а также из главного меню или формы. Иногда бывает необходимо вызвать отчет прямо из X++, ведь пользователь может вызывать отчет и не через пункт меню.

#### Элементы проекта MORPHXIT Reports

- `Job, Reports_ExecuteReport`
- `Job, Reports_ExecuteReportSilent`

```
static void Reports_ExecuteReport(Args _args)
{
    Args                args;
    SysReportRun        reportRun;
;

    args = new Args();

    reportRun = new menuFunction(menuItemOutputStr(MyReport),
                                MenuItemType::Output).create(args);
    reportRun.run();
}
```

Задание показывает, как вызвать отчет `MyReport` из X++. Отметим, что используется класс приложения `SysReportRun`. Класс `SysReportRun` наследуется от системного класса `ReportRun`. Преимущество использования класса

приложения в том, что вы имеете возможность переписать код класса SysReportRun. Для этого создайте собственный класс, наследуемый от класса SysReportRun. Это даст вам шанс иметь серию отчетов с проверкой при печати. Это лучше чем исправлять каждый отчет.

MyReport вызывается через пункт меню, который вызывается структурой отчета RunBase и загружает корректный диалог. Если вы не хотите использовать структуру runbase или вы хотите печатать отчет без вмешательства пользователя, то вам следует запустить отчет без использования структуры RunBase отчета.

```
static void Reports_ExecuteReportSilent(Args _args)
{
    Args          args;
    SysReportRun  reportRun;
;
    args = new Args();
    args.name(reportstr(MyReport));

    reportRun = classFactory.reportRunClass(args);
    reportRun.query().interactive(false);
    reportRun.report().interactive(false);
    reportRun.setTarget(PrintMedium::Printer);
    reportRun.run();
}
```

В этом задании отчет выполняется без использования пункта меню и структуры runbase отчета. Пример печатает отчет прямо на принтер по умолчанию, без вывода диалогов и вмешательства пользователя отметьте, что и запрос, и диалог отчета установлены в Inactive. Если метод dialog() переписан в вашем отчете, то вам необходимо удостовериться в том, что не создаются проблемы при невыполнении dialog().

Если ваш отчет состоит более чем из одного дизайна, вы можете определить, какой дизайн использовать. Если не определено, или если не корректно имя дизайна, будет использован первый дизайн.

```
reportRun.design("MyDesign");
reportRun.run();
```

## Использование временных таблиц

Если нужна специальная сортировка или вы выбираете данные из нескольких таблиц, которые не могут быть объединены, использование временной таблицы может быть достойным выбором. Использование временной таблицы достаточно просто. Временную таблицу следует заполнить и передать в отчет. Могут быть проблемы выполнения при использовании временных таблиц, если отчет запускается два раза. Сначала строится временная таблица. Затем она перебирается в цикле в методе отчета. Использование временных таблиц должно быть вашим взвешенным выбором. Вам следует тщательно исследовать ваш дизайн. Для дальнейшей информации по временным таблицам, смотрите главу: **Словарь Данных**.

### Элементы проекта MORPHXIT Reports

- Class, Reports\_TempTable
- Report, Reports\_TempTable

При использовании временной таблицы, отчет лучше вызывать из класса. Это даст вам возможность строить временную таблицу на сервере. Следующий пример класса показывает, как создать отчет, используя временную таблицу. Для простоты, пример добавляет 10 записей во временную таблицу и печатает результат.

```
class Reports_TempTable extends runBaseReport
{
}
```

Класс наследуется от runBaseReport.

```
public identifiername lastValueElementName()
{
    return reportstr(Reports_TempTable);
}
```

Имя для отчета должно быть предопределено.

```
tmpAccountSum tempTable()
{
    CustTrans          custTrans;
    TmpAccountSum      tmpAccountSum;
    Counter            counter;
;

    while select custTrans
    {
        counter++;

        if (counter == 10)
        {
            break;
        }

        tmpAccountSum.accountNum      = custTrans.accountNum;
        tmpAccountSum.currencyCode    = custTrans.currencyCode;
        tmpAccountSum.balance01       = custTrans.amountMST;
        tmpAccountSum.insert();
    }

    return tmpAccountSum;
}
```

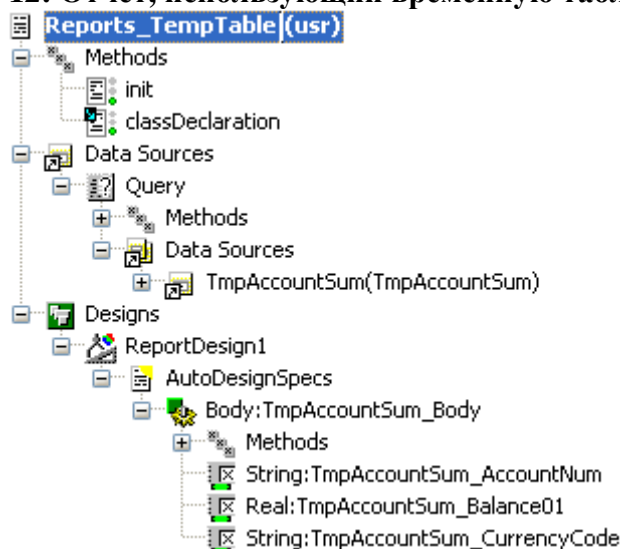
Используется временная таблица tmpAccountSum. Первые десять записей из таблицы CustTrans вставляем в tmpAccountSum. Этот метод используется для передачи параметром заполненной временной таблицы в отчет.

```
static void main(Args args)
{
    Reports_TempTable reports_TempTable = new reports_TempTable();

    if (reports_TempTable.prompt())
    {
        reports_TempTable.run();
    }
}
```

Класс инициализируется и выполняется отчет.

Последний шаг – это создание отчета. Необходимо создать временную таблицу TmpAccountSum как источник данных. Три поля заполняются значениями из CustTrans и выводятся. Ваш отчет должен выглядеть, как показано на **Рисунке 12: Отчет, использующий временную таблицу.**



**Рисунок 43: Отчет, использующий временную таблицу**

Необходимо переписать Init(). Класс RunBase инициализируется и запрос заполняется временной таблицей. Отметим, что вам необходимо установить ссылку на буфер. Запрос заполнится временной таблицей, и выведет все записи из временной таблицы.

```
public void init()
{
    Reports_TempTable    reports_tempTable;
;
    super();

    reports_TempTable = element.args().caller();

    if (!reports_TempTable)
    {
        throw error(Error::missingRecord(funcName()));
    }
    reports_TempTable.queryRun().setRecord(reports_TempTable.tempTable());
}
```

## Подкраска рядов

Цвета редко используются в стандартных отчетах. Вам необходимо немного доделать отчет, что бы получить желаемый результат. Однако использование цвета может предоставить вашему отчету последний штрих, что сделает его чтение более легким.

### Элементы проекта MORPHXIT Reports

- Report, MyReport\_Color
- Menu item output, MyReport\_Color

Этот пример показывает, как подкрасить одну колонку от определенного условия. Отчет устанавливает цвет фона поля, показывающего CustTrans.amountMST. Для простоты кода, проверим условие, что отчет выводится из X++. В реальной жизни условие должно устанавливаться в диалоге или основываться на данных из формы. Вы должны довести дизайн отчета, как показано на **Рисунке 13: отчет с подкрашенными рядами**.

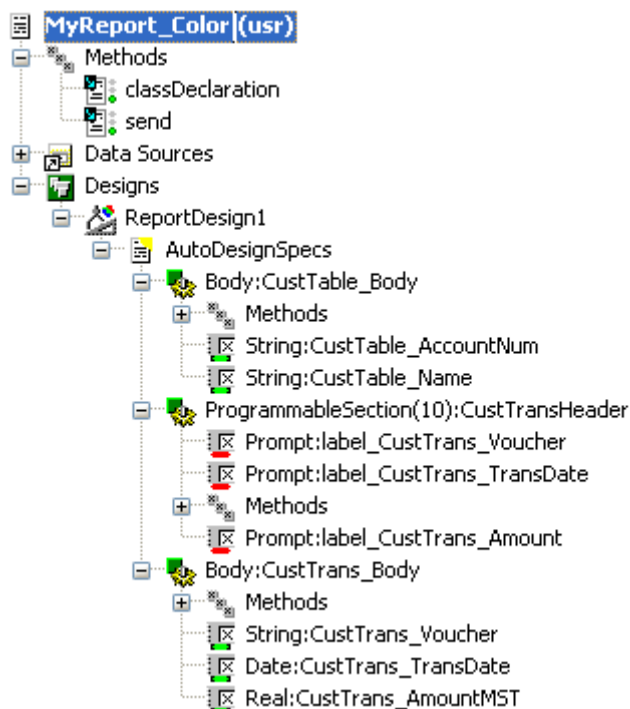


Рисунок 44: Отчет с подкрашенными рядами

1. Сделайте копию отчета “MyReport,” и переименуйте в “MyReport\_Color”.
2. Цель – определение цвета поля CustTrans\_AmountMST. Используя пример MyReport, как есть, header label будет менять цвет. Вместо стандартного заголовка для секции body CustTrans\_Body, необходимо создать новый. Чтобы уйти от стандартного заголовка, установите свойство на header **NoOfHeadingLines** в “0” в секции body CustTrans\_Body.
3. Создайте новый заголовок добавлением секции programmable и полей prompt для каждого из трех полей в секции body. Свойство ModelFieldName на каждом поле prompt должно быть установлено в секции body в соответствии с name поля. Добавьте метку для каждого поля prompt.
4. Определите переменную, содержащую условие, когда секция programmable используемая для header должна печататься.

```
public class ReportRun extends ObjectRun
{
    Boolean    printCustTransHeader;
}
```

5. Теперь перепишите метод `send()`. Если текущая запись – это запись из `CustTrans`, заголовок напечатается для первой записи `CustTrans` в ряд. Условие установлено для `CustTrans.AmountMST`. Если сумма более 500, цвет фона устанавливается в желтый. Иначе цвет фона будет стандартным.

```
public boolean send(Common _cursor, int _level=1, boolean _triggerOffBody=TRUE, boolean
    _newPageBeforeBody=FALSE)
{
    boolean ret;
    ;

    if (_cursor.tableId == tableNum(custTable))
        printCustTransHeader = true;

    if (_cursor.tableId == tableNum(custTrans))
    {
        if (printCustTransHeader)
        {
            element.execute(10);
            printCustTransHeader = false;
        }

        if (custTrans.amountMST > 500)
        {
            CustTrans_AmountMST.backgroundColor(Winapi::RGB2int(255, 255, 0));
        }
        else
        {
            CustTrans_AmountMST.backgroundColor(Winapi::RGB2int(255, 255, 255));
        }
    }

    ret = super(_cursor, _level, _triggerOffBody, _newPageBeforeBody);

    return ret;
}
```

## Печать отчета, используя Microsoft Word

Создание отчета с комплексным дизайном такого, как таблицы с графикой, расчет формул в Ахарт, может быть реализовано. Используя Com интерфейс, вы можете соединиться с внешним приложением таким, как Microsoft Word. Что бы использовать Microsoft Word для печати данных, вам следует создать шаблон Microsoft Word с закладками. Закладки используются для размещения данных из Ахарт. Отметим, что вы должны иметь лицензионный код для, как минимум, одного Com клиента, использующего Com интерфейс.

---

**Примечание:** Различные документы в стандартной поставке обрабатываются, используя Com интерфейс. Классы, использующие взаимодействие с классом Document имеют префикс

DocuActionCOM. В российской локализации можно использовать класс ComWordDocument\_RU, который является наследником общего класса ComOfficeDocument\_RU, а он в свою очередь использует Com.

---

### Элементы проекта MORPHXIT Reports

- Class, Report\_PrintUsingWord
- Job, Reports\_PrintUsingWord

### Дополнительно

- Шаблон Microsoft Word, Reports\_WordTemplate.dot

Этот пример покажет, как соединиться с Microsoft Word и создать новый документ, который будет печатать данные из таблицы InventTable. Будут напечатаны первые 10 записей из InventTable. Будет печататься метка для отчета и для заголовков колонок.

Вам необходимо создать шаблон Microsoft Word, со следующими закладками: label\_header, label\_itemid, label\_itemname, и label\_itemdesc. Label\_header будет печатать текст для колонок. Создайте таблицу и добавьте упомянутые 3 закладки, как заголовки для таблицы. Следующий шаг - создание следующего класса:

```
void run()
{
    COM          COMAppl, COMDocuments, COMDocument;
    ;

    COMAppl      = new COM('Word.Application');
    COMDocuments = COMAppl.documents();

    // enter path to the template Reports_wordtemplate.dot
    COMDocument = COMdocuments.add('d:\Reports_WordTemplate.dot');

    if (COMDocument)
    {
        this.setLabels(COMDocument);
        this.sendInventTable(COMDocument);
        this.showDocument(COMAppl);
    }
}
```

Run() инициирует COM соединение и открывает новый Microsoft Word документ, основанный на шаблоне Reports\_WordTemplate.dot. Запомните о необходимости проверки пути к шаблону. Если документ создан, название и дата добавляются. Окончательно документ открывается. Отметьте, что документ открывается не сохраненным. Если вы хотите сохранить документ, то должны добавить: COMdocument.saveAs(<filename>,0,false,"",false);.

```
void setLabels(COM _COMDocument)
{
    COM          COMBookmarks, COMBookmark, COMrange;
    DictField    dictField;
    Label        label;
    ;

    COMBookmarks = _COMDocument.bookmarks();
```



```

if (COMbookmarks.exists('label_header'))
{
    COMbookmark      = COMbookmarks.item('label_header');
    COMrange          = COMbookmark.range();
    COMRange.InsertAfter("Inventory list");
}

if (COMbookmarks.exists('label_itemId'))
{
    COMbookmark      = COMbookmarks.item('label_itemId');
    COMrange          = COMbookmark.range();
    DictField        = new dictField(tableNum(inventTable), fieldNum(inventTable, itemId));
    COMRange.InsertAfter(dictField.label());
}

if (COMbookmarks.exists('label_itemName'))
{
    COMbookmark      = COMbookmarks.item('label_itemName');
    COMrange          = COMbookmark.range();
    DictField         = new dictField(tableNum(inventTable),
                                      fieldNum(inventTable, itemName));

    COMRange.insertAfter(dictField.label());
}

if (COMbookmarks.exists('label_itemDesc'))
{
    COMbookmark      = COMbookmarks.item('label_itemDesc');
    COMrange          = COMbookmark.range();
    label             = new Label(CompanyInfo::languageId());
    COMRange.InsertAfter(label.extractString(literalstr("@SYS58702")));
}
}

```

Проверка делается для просмотра: существуют закладки или нет. Если закладки найдены, устанавливается заголовок. Название заголовка устанавливается в статический текст "Inventory list." Названия для закладок label\_itemId и label\_itemName устанавливаются из соответствующих полей таблицы. Названия для закладок label\_itemDesc устанавливаются в методе label.extractString() для вывода названия языка компании по умолчанию.

```

void sendInventTable(COM _COMDocument)
{
    COM      COMTable, COMRows, COMRow;
    COM      COMCells, COMCell, COMRange;
    InventTable inventTable;
    Counter    counter;
;

    //init tabel
    COMTable      = COMDocument.Tables();
    COMTable      = COMTable.Item(1);
    COMRows       = COMTable.Rows();

    while select inventTable
    {
        counter++;

        if (counter == 10)

```

```

    {
        break;
    }

    // add new row
    COMRow      = COMRows.Add();
    COMCells    = COMRow.Cells();

    // item id
    COMCell      = COMCells.Item(1);
    COMRange    = COMCell.Range();
    COMRange.InsertAfter(inventTable.itemId);

    // item name
    COMCell      = COMCells.Item(2);
    COMRange    = COMCell.Range();
    COMRange.InsertAfter(inventTable.itemName);

    // item description
    COMCell      = COMCells.Item(3);
    COMRange    = COMCell.Range();
    COMRange.InsertAfter(inventTable.itemDescription());
}
}

```

Этот код выбирает в цикле записи из `inventTable`. Таблица в Microsoft Word иницируется первой. С каждым циклом, новый ряд добавляется в таблицу Word. Поля `itemId`, `itemName` и дисплей метод `itemDescription()` из `InvenTable` добавляется в третий ряд в таблице Microsoft Word. Отметьте, что нет необходимости в закладках.

```

void showDocument(COM _COMAppl)
{
    _COMAppl.visible(TRUE);
}

```

Открывается созданный документ Microsoft Word.

```

static void main(Args _args)
{
    Reports_PrintUsingWord printUsingWord = new Reports_PrintUsingWord();
    ;

    printUsingWord.run();
}

```

Этот код иницирует и исполняет класс.

## 7.8 Резюме

Эта глава ввела вас в отчеты в Ахapta. Глава содержит основные базовые знания по созданию отчетов. Теперь вам должны быть знакомы различные элементы отчета, как источник данных, дизайн, секции, поля дизайна и основные методы отчета. Теперь у вас есть знания по созданию отчетов, используя генератор отчетов, и вы с уверенностью вникните в мощь предлагаемую отчетами Ахapta и средой MorphX.

## 8 Запросы

Использование инструкции `select` и запросов AOT [Queries] - это две основные возможности выборки данных в технологии MorphX из базы данных. Если инструкция `select` это статическое выражение, написанное на X++, то запрос AOT может быть написан как на языке X++, так и создан в узле *Queries* AOT. Критерии запроса можно менять во время его выполнения в диалоге запроса. Из X++ запрос так же можно модифицировать специальными системными классами. Это делает запрос крайне удобным при работе с такими объектами, как формы и отчеты, так как данные могут фильтроваться и сортироваться с изменением условий в форме диалога запроса или в языке X++.

Настройки, сделанные пользователем в диалоге запроса сохраняются для пользователя в разрезе компании, как последнее значение. Хранение последних значений облегчает работу пользователя, так как настройки сохраняются с первого запуска запроса.

Использовать запрос или инструкцию `select` - зависит от задачи представления критериев пользователю. Инструкция `selects` и запросы - это часть MorphX и оба выполняются ядром системы. Использование `select` или запроса зависит от ваших потребностей. Запросы следует использовать, если вы предполагаете предоставление интерфейса фильтра данных или если вам необходимо делать изменения критериев во время выполнения запроса, таких как фильтрация данных в форме или отчете. Инструкция `selects` обычно используется при выводе данных без использования диалогового интерфейса.

Эта глава будет сосредоточена на создании и использовании запросов в MorphX. Вы можете найти информацию по использованию интерфейса пользовательских запросов в руководстве пользователя в стандартной поставке.

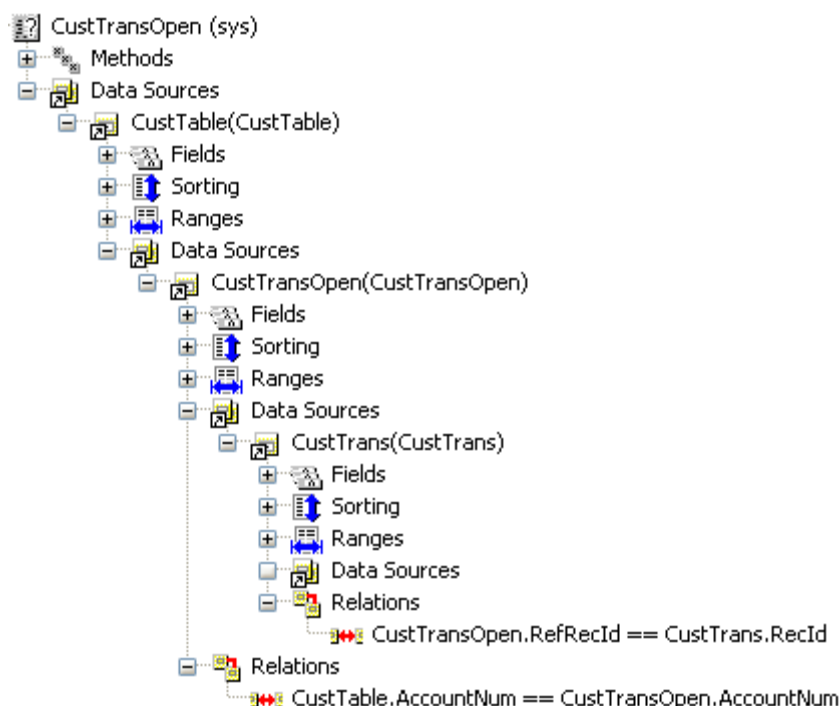


Рисунок 45: Запросы хранятся в АОТ

## 8.1 Создание запросов

Сложные запросы и запросы, у которых методы по умолчанию переопределены, следует строить с использованием узла *Queries*. Запросы, расположенные в специальном узле АОТ и представлены в виде древовидной структуры для лучшего обзора компонент запроса. Особенно новичкам в запросах это будет очень удобно, так как вам не стоит беспокоиться об особенностях использования системных классов.

### Запрос АОТ

Запросы, расположенные в АОТ вы можете использовать в любом месте вашего кода. Запросы АОТ не могут быть объявлены в коде как тип или таблица. Вам следует использовать вместо этого системные классы для выполнения запроса. Это делает запрос более гибким, так как вы используете всего несколько строчек кода для интеграции вашего запроса в любое место кода. Если вы позднее решите изменить ваш запрос, например, изменить фильтрацию данных, то ваши изменения отразятся во всех местах, где используется ваш запрос. Недостаток может быть в том, что не совсем понятно какой запрос использовать. Если вы обнаружите запрос в АОТ подходящий вашим требованиям, но вы не знаете, где он используется, если только не воспользоваться перекрестными ссылками. Делая изменения в таком запросе, может иметь фатальные последствия. Так что вы, возможно, придете к тому, чтобы создать свой запрос в АОТ.

**Базовые компоненты**

Первый шаг в создании запроса – это определение таблиц, которые будут использоваться в запросе. Таблицы, используемые в запросе, располагаются в источнике данных. Вы можете сравнить источник данных с табличной переменной. Обычно источник данных имеет тоже название, что и связанная таблица. Если только вы будете использовать таблицу в запросе более чем один раз в одном и том же запросе, то вам следует изменить название источника данных. На название источника данных можно ссылаться из X++ так же, как и на табличную переменную. Таблицы, карты соответствия [table maps], обзоры [views] могут использоваться в качестве источника данных. Использование этих объектов в X++ тоже самое. Только при использовании временных таблиц, вам следует добавлять некоторый код.

---

**Примечание:** Если вы не уверены в выходе вашего запроса, то вам следует попытаться создать отчет с использованием вашего запроса. Отчет выведет результат вашего запроса. Смотри главу **Отчеты** по теме использования вашего запроса.

---

Если запрос выводит данные из более, чем одной таблицы, то вам также как в инструкции select следует определить условие объединения таблиц. Каждый уровень дерева запроса определяет условие объединения двух таблиц.

---

**Пример 1: Создание запроса АОТ**Элементы MORPHXIT проекта Queries

➤ Query, MyQuery

Этот пример покажет вам, как создать запрос в АОТ с объединением двух таблиц и установкой критерия по умолчанию для фильтрации. Вам следует создать запрос, как показано на **рисунке 46: MyQuery**.

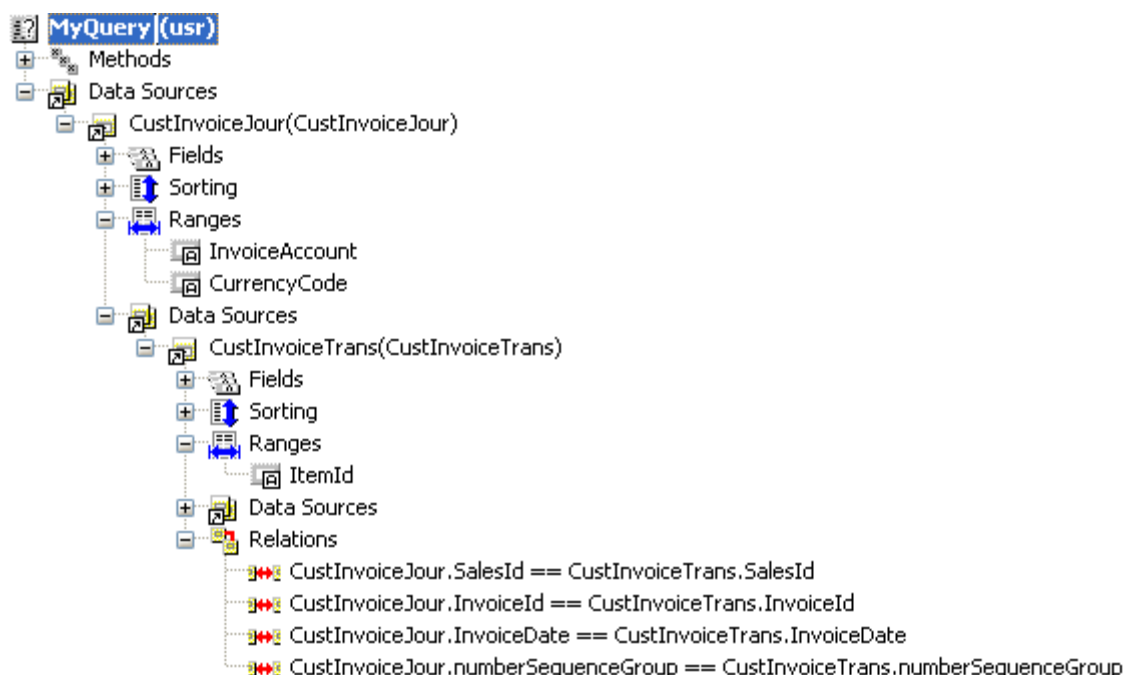


Рисунок 46: MyQuery

1. Перейдите к узлу *Queries* в AOT, и через контекстное меню выберите *Создать Query*. Переименуйте запрос в "MyQuery" и разверните новый источник данных.
2. Откройте еще одно окно AOT и найдите таблицу *CustInvoiceJour*, перетащите таблицу в узел запроса *Data Sources*. Источник данных будет иметь суффикс "\_1". Это делается для защиты уникальных имен. Уберите суффикс.
3. Разверните узел нового источника данных, и перетащите таблицу *CustInvoiceTrans* в узел *Data Sources/CustInvoiceJour/Data Sources*, удалите суффикс.
4. Перейдите к листу свойств источника данных *CustInvoiceTrans* и установите свойство **Relations** в "Yes". Разверните новый источник данных и проверьте, что поля, по которым связаны таблицы, представлены списком в *CustInvoiceTrans/Relations*.
5. Перейдите к узлу *Ranges* под источником данных *CustInvoiceJour*, и выберите *Создать Range*. Новый критерий будет добавлен первым полем из источника данных. Исправьте поле, открыв лист свойств, для нового критерия и выберите поле *InvoiceAccount*, используя свойство **Field**.
6. Повторите шаг 5 добавлением критерия для поля *CustGroup*.
7. Перейдите к источнику *CustInvoiceTrans* и добавьте критерий для поля *ItemId*.

Запрос, созданный в этом примере показывает связь таблиц *CustInvoiceTable* и *CustInvoiceTrans*, которые составляют журнал накладных клиента. Таблицы по умолчанию объединены по inner join. Вы можете изменить режим объединения

двух источников данных установкой свойства **JoinMode** нижнего по уровню источника данных. Свойство **Relations**, расположенное у нижнего по уровню источника данных, используется для определения полей связи объединения. Эти поля связи определены в словаре данных [data dictionary]. Вы можете определить поля связи вручную, но предпочтительно использовать настройки словаря данных, так как это делает настройку проще. Если связи изменятся в словаре данных, то все запросы, использующие источник данных автоматически обновятся.

Теперь попробуйте создать задание [job] для проверки запроса:

```
static void Queries_TestMyQuery(Args _args)
{
    SysQueryRun      queryRun = new SysQueryRun(querystr(MyQuery));
    CustInvoiceJour   custInvoiceJour;
    CustInvoiceTrans  custInvoiceTrans;
;

    if (queryRun.prompt())
    {
        while (queryRun.next())
        {
            custInvoiceJour = queryRun.get(tableNum(CustInvoiceJour));
            custInvoiceTrans = queryRun.get(tableNum(CustInvoiceTrans));

            if (queryRun.changed(tableNum(CustInvoiceJour)))
            {
                info(strfmt("Account: %1", custInvoiceJour.invoiceAccount));
            }

            if (queryRun.changed(tableNum(CustInvoiceTrans)))
            {
                info(strfmt("Item: %1, Qty: %2, ", custInvoiceTrans.itemId, custInvoiceTrans.qty));
            }
        }
    }
}
```

SysQueryRun – наследник системного класса QueryRun используется для выполнения запроса. Название запроса определяется функцией querystr(). Название запроса можно вводить просто как текст, но функция querystr() проверит, чтобы запрос существовал в АОТ. QueryRun.prompt() вызывает диалог запроса и если кнопка ОК нажата в диалоге запроса, то выводятся записи соответственно определенным в запросе критериям. Необходимо явно определить табличную переменную каждого источника данных для получения значений записей из запроса. При выборке записей запроса в цикле, вам необходимо определить, из какого источника выводить данные. Это делается использованием queryRun.changed().

---

**Примечание:** Перетаскиванием запроса АОТ в редактор, вы можете создать базовую структуру цикла обработки запроса.

---

При выполнении MyQuery, появляется диалог запроса. Три поля, определенные под узлом *Ranges* в запросе, по умолчанию проставляются как поля для фильтра.

Пользователь может добавлять или удалять в любом поле критерия по умолчанию. Однако вы можете установить запрет редактирования для критерия, используя свойство **Status**. По умолчанию критерии создаются со статусом **Open**. Вы можете ввести значение по умолчанию для критерия, используя свойство **Value**. Если свойство **Status** установлено в **Lock**, то пользователь не сможет редактировать критерий. Запрет редактирования критерия может быть полезен, если у вас есть фильтрация данных в форме, и вы хотите информировать пользователя об обязательном фильтре. Вы можете установить свойство критерия **Status** в **Hidden**, если вы не хотите показывать поля фильтра по умолчанию пользователю.

Узел *Sorting* используется для определения индекса или полей для сортировки данных. Определяя поле для сортировки, вы определяете порядок вывода данных (order by) или (group by). В примере MyQuery не было определено сортировки. Подобно и в инструкции selects, если вы уверены в необходимости изменения сортировки выводимых данных, то это следует отдельно определить выражением (order by). Сортировка обычно используется в запросах отчетов. Запрос отчета имеет дополнительные свойства для сортировки, делая возможным суммирование, основанное на полях сортировки. За дальнейшей информацией по использованию запросов в отчетах, смотри главу **Отчеты**.

```
static void Queries_MyQueryAsSelect(Args _args)
{
    CustInvoiceJour      custInvoiceJour;
    CustInvoiceTrans     custInvoiceTrans;
;

    while select custInvoiceJour
        join custInvoiceTrans
            where custInvoiceJour.salesId == custInvoiceTrans.salesId
                && custInvoiceJour.invoiceId == custInvoiceTrans.invoiceId
                && custInvoiceJour.invoiceDate == custInvoiceTrans.invoiceDate
                && custInvoiceJour.numberSequenceGroup == custInvoiceTrans.numberSequenceGroup
        {
            // print fetch data
        }
    }
```

Если бы MyQuery был определен с помощью инструкции select, то код соответствовал бы приведенному выше примеру. Отметьте, что все поля в условии where соединены как and'ed (&&). Запросы всегда обрабатывают данные с использованием and'ing в полях критериев. Только выражение select может использовать оба варианта "and" и "or". Однако попробуем поискать использование той же самой опции в запросах. Это будет объяснено в следующих разделах.

### Агрегатные функции

По умолчанию запрос выводит все поля таблицы, так же как и в инструкции select. Если изменить свойство **Dynamic** на узле *Fields* в No, то запрос будет выводиться только те поля, которые особым образом определены, например,



агрегатной функцией. Рекомендуется использовать эту возможность, если вы понимаете данную настройку при выводе данных запроса в X++.

Свойство Dynamic так же служит и другой цели. Через контекстное меню на узле *Fields* вы можете выбрать одну из агрегатных функций AVG, SUM, COUNT, MIN или MAX. Выбором агрегатной функции вы устанавливаете свойство Dynamic в No. Если агрегируемые поля таблицы не содержат значений, то запрос ничего не выведет. Агрегатная функция используется для производства вычислений над выводимыми данными. Самое распространенное использование агрегатных функций – это суммирование в отчете.

---

### Пример 2: Агрегатная функция

#### Элементы MorphxIt проекта Queries

##### ➤ Query, MyQuery\_Aggregate

Этот пример показывает, как использовать агрегатную функцию SUM, для суммирования проданного количества за единицу номенклатуры по журналу накладных.

1. Создайте новый запрос и переименуйте его в "MyQuery\_Aggregate".
  2. Перетащите таблицу CustInvoiceTrans в источник данных и удалите суффикс.
  3. Разверните источник данных CustInvoiceTrans, выберите *Создать/SUM* в контекстном меню на узле *Fields*. Откройте лист свойств суммируемого поля и выберите поле Qty.
  4. Перейдите к узлу *Sorting*, выберите в контекстном меню *Создать/Field*. Используйте лист свойств для выбора сортируемого поля ItemId.
  5. Последний шаг – установка порядка сортировки. Перейдите к источнику данных CustInvoiceTrans и установите свойство **OrderMode** в "Group by".
- 

Вы можете добавлять любое количество агрегатных функций в запрос. Для упрощения мы использовали только одну агрегатную функцию. При использовании агрегатной функции, сортируемые поля вы можете использовать в функции группировки. При выборе агрегатной функции, только сортируемые поля будут выводиться со значениями. Режим order mode следует выставлять в group by при использовании агрегатной функции. Не имеет смысла использовать order by так как это будет следствием вывода всех записей, и в этом примере нечего будет суммировать.

```
static void Queries_TestMyQuery_Aggregate(Args _args)
{
    SysQueryRun      queryRun = new SysQueryRun(querystr(MyQuery_Aggregate));
    CustInvoiceTrans custInvoiceTrans;
;


    if (queryRun.prompt())
    {
```

```
while (queryRun.next())
{
    custInvoiceTrans = queryRun.get(tableNum(CustInvoiceTrans));

    if (queryRun.changed(tableNum(CustInvoiceTrans)))
    {
        info(strfmt("Item: %1, Qty: %2, ",custInvoiceTrans.itemId, custInvoiceTrans.qty));
    }
}
}
```

Это задание [job] можно использовать для тестирования запроса. При выполнении вы увидите, что сортируемые поля добавлены в диалог запроса на закладку сортировки. Так как данные выводятся с функцией group by, то вы не можете изменить сортируемые поля.

### Продвинутые критерии

Ограничение в использовании запроса заключается в том, что критерии and'ed (прикрепляются с условием И). Скажем, вы хотите найти проводки по журналу накладных, принадлежащих группе покупателей "40" или проводки, которые имеют код валюты "USD". Если использовать запрос, то вы получите вывод записей, у которых оба условия истина. Однако существует обработка и для таких ситуаций. Откройте форму CustInvoiceJournal и нажмите иконку . Теперь введите следующее выражение в критерий: ((custgroup == "20") || (currency == "USD")). Не имеет значения, в какой критерий запроса вы вводите это, так как введенный код не переводится как значения выбранных полей запроса. При нажатии ОК в диалоге запросе, записи проводок из формы журнала накладных выводятся с использованием выражения OR.

---

**Примечание:** Групповые символы, используемые в диалоге запроса такие как '\*' и '.' могут также использоваться в X++ при построении критерия запроса. Смотри online help построения диалога запроса с использованием списка групповых символов [wildcard].

---

Круглые скобки позволяют писать даже булево выражение в критерии запроса с использованием полей источника данных. Но не существует проверок при вводе такого рода выражений. В действительности могут возникнуть даже проблемы. По этой причине вам всегда следует добавлять такие выражения в запрос, используя X++. Для тестирования таких выражений быстрее попытаться выполнить выражение, используя диалог запроса, а затем скопировать текст критерия запроса в X++.

Вы, может быть, хотите знать, что мешает использовать это в инструкции select. Может так случиться, что Вам придется модифицировать существующий объект, как класс или форма, использующий такой запрос. Если вы захотите заменить существующий запрос инструкцией select, то вам придется менять много строчек кода. Помимо прочего всегда предпочтительно применять запрос АОТ для объектов, используемых в пользовательском интерфейсе.

## Методы Запроса

Методы запроса АОТ почти никогда не перекрываются. Изменения всегда производятся в методах источника данных. Только формы имеют узлы в АОТ для переопределения методов источника данных. Смори главу **Формы** по использованию методов источника данных.

## X++ Запрос

Как мы уже видели, запрос можно выполнять из X++, но запрос также можно строить и выполнять из X++ с использованием системных классов с префиксом Query\*. Обычно простой запрос или запрос, который используется для специфических целей, строиться с использованием X++.

---

**Примечание:** некоторые системные классы наследуют классы приложения. Вам следует использовать наследников классов приложения по причине содержания дополнительной логики. Наследник системного класса обычно имеет префикс Sys\*, например, SysQuery.

---

Когда вы напишите небольшой запрос, то найдете достаточно легким строить запросы с использованием X++.

```
static void Queries_SystemClasses(Args _args)
{
    SysQuery          query;
    SysQueryRun        queryRun;
    QueryBuildDataSource custInvoiceJourDS, custInvoiceTransDS;
;

    query = new Query();
    custInvoiceJourDS = query.addDataSource(tablename(CustInvoiceJour));
    custInvoiceJourDS.addRange(fieldnum(CustInvoiceJour, InvoiceAccount));
    custInvoiceJourDS.addRange(fieldnum(CustInvoiceJour, CurrencyCode));
    custInvoiceTransDS = custInvoiceJourDS.addDataSource(tablename(CustInvoiceTrans));
    custInvoiceTransDS.addRange(fieldnum(CustInvoiceTrans, ItemId));
    custInvoiceTransDS.relations(true);

    queryRun = new SysQueryRun(query);
    queryRun.prompt();
}
```

Это запрос, используемый в примере MyQuery, только построенный с использованием системного класса. Здесь логика такая же, как и при строительстве запроса в АОТ. Во-первых, новый запрос так же объявляется, и добавляется источник данных к узлу запроса. Класс QueryBuildDataSource осуществляет присоединение источника данных. Таблица CustInvoiceJour добавляется источником данных. Так же как и в запросе АОТ связи (relations) устанавливаются в true, вследствие этого используются связи словаря данных (data dictionary relations). Поля критериев по умолчанию добавляются с применением метода addRange(). Для обзора диалога запроса, вызывается метод prompt(), при выполнении задания (job). Делая видимым диалог запроса, вам легче разобраться в построенном запросе.

Если в обработке запроса используется выбор записей из временной таблицы, то вам следует добавить следующую строчку после инициализации объекта QueryRun. Это распространенная ошибка – написание кода без этой строчки. И вы будете задаваться вопросом, почему не получили вывод данных из вашей временной таблицы, как при использовании инструкции select.

```
queryRun.setRecord(MyTable);
```

Добавление критериев в запрос часто производится с помощью X++. Типичный случай – это фильтрация выводимых данных из запроса с использованием значений переменных.

```
static void Queries_SystemClassesRanges(Args _args)
{
    SysQuery          query;
    SysQueryRun        queryRun;
    QueryBuildDataSource custInvoiceJourDS;
    QueryBuildRange     rangeInvoiceAccount, rangeInvoiceDate, rangeDimensionDepartment;
;

    query = new Query();
    custInvoiceJourDS = query.addDataSource(tablename(CustInvoiceJour));
    rangeInvoiceAccount = custInvoiceJourDS.addRange(fieldnum(CustInvoiceJour, InvoiceAccount));
    rangeInvoiceAccount.value(queryValue("4000"));

    rangeInvoiceDate = custInvoiceJourDS.addRange(fieldnum(CustInvoiceJour, InvoiceDate));
    rangeInvoiceDate.value(queryRange(datenull(), systemdateget()));

    rangeDimensionDepartment = custInvoiceJourDS.addRange(fieldId2Ext(
                                                                    fieldnum(CustInvoiceJour, Dimension), 1));
    rangeDimensionDepartment.value(queryValue("Sales"));

    queryRun = new SysQueryRun(query);
    queryRun.prompt();
}
```

В представленном выше примере, добавлено 3 критерия в запрос, содержащий один источник данных. Метод addRange() возвращает сущность класса QueryBuildRange, который используется для установки значения для каждого критерия. Режим группировки или сортировки определяется методом системного класса QueryBuildDataSource (orderMode(orderMode::GroupBy)). You can define order mode in your query in method of system class QueryBuildDataSource (orderMode(orderMode::GroupBy)).

Отметьте глобальные функции queryValue() и queryRange(). Вам следует запомнить использование этих функций при добавлении значения критерия в запрос, так как они принимают значения любого типа в качестве параметра и преобразуют значение в определенный критерием тип. Функция queryRange() используется для ввода и вывода значений, только queryValue() используется для одного значения. Если вам необходимо оперировать множеством значений для критерия, то вместо этого используйте функцию strfmt() для форматирования вашего выражения. Только помните, что значения в критерии следует разделять запятыми.

Использование ввода массива полей в качестве критерия запроса потребует дополнительного кодирования. Аналитика [dimension] – самый распространенный массив полей. При добавлении аналитики в критерий запроса АОТ, добавляется каждый член массива. Вы можете также выбрать каждый член массива из списка полей. Это немного отличается в X++, так как вы не можете определить номер в массиве, просто введя Dimension[1], для выбора первого члена массива. Вместо этого используется глобальная функция fieldId2Ext(). Первый параметр это id поля, а второй параметр - номер в массиве. Здесь использована первая аналитика из CustInvoiceJour и добавлена в качестве критерия.

Критерий запроса имеет ограниченный размер. Так как максимальный размер высок, то вам необходимо связать несколько сотен критериев запроса для достижения предела. Вы думаете, что это много. Если значения критериев для вывода данных не могут быть заполнены из запроса, то частенько прибегают к использованию инструкции selects для построения строки критерия запроса. Это может быть список номенклатур, выводимых из таблицы номенклатур. Однако это не рассматривается хорошим решением, так как вы не знаете, когда придет предел. Добавление дополнительного критерия приводит к тому, что в результате два критерия соединяются с условием И (AND'ed). Достигнув предела, будет результатом ошибки базы данных. Используя расширенный тип данных Range, вы вновь можете продвинуться дальше. Расширенный тип данных range имеет тип строку, ограниченную длиной в 250 символов. Вы можете изменить длину строки до 1000 символов, который является пределом для типа строки. Перед тем как зайти так далеко, вам следует рассмотреть использование временной таблицы или отсортированного листа записей [records sorted list].

Вы часто будете использовать критерии запроса, при конструировании форм и отчетов, так как критерии обычно используются в X++ для фильтрации вывода данных, например, для фильтрации данных формы или отчета от курсора вызывающей формой.

## 8.2 Запросы в формах и отчетах

И формы и отчеты используют запросы для вывода данных. Так как формы и отчеты имеют только свои предопределенные узлы, то для построения запросов они не используют запросы АОТ. По сравнению с запросом АОТ формы и отчеты имеют дополнительные свойства и возможности. Однако, используя X++, вы можете указать другой запрос или событие, строящее запрос (источник данных), вместо использования методов источника данных формы или запроса. Однако обычно используются методы форм и отчетов.

Инструкция Select может производить ту же самую работу, что и запрос в форме или отчете, но использование select будет менее наглядно.

В формах опция фильтрации и сортировки данных, также используемая для печати авто отчета (auto reports), основана на определенных полях в группе полей AutoReport таблицы. Формы, которые не содержат запрос (источник данных) так же не имеют и этой опции.

Отчеты, не содержащие запрос (источник данных), не имеют фильтра или опции автоматической печати итогов. Если использовать select в отчете, то фильтр можно создать вручную в коде.

### 8.3 Резюме

Запросы играют основную роль в MorphX. Важно понимать концепцию запросов, так как вы будете часто их использовать. Понимание, как конструировать запрос и как использовать системные классы, сделает для вас легкой модификацию объектов, использующих запрос, а так же поможет вам делать более наглядные модификации.

Теперь у вас есть базовые знания по построению и использованию запросов.

Прочитав о запросах в главах **Формы** и **Отчеты**, вы получите финальную картину по использованию запросов в MorphX.

## 9 Задания

Вы, наверное, заметили, что многие примеры в этой книге написаны с использованием заданий (jobs). Это в действительности главное назначение заданий, написание тестовых скриптов для приложения.

Задания используются для тестирования ваших модификаций, так как иногда вам необходимо протестировать целый блок кода, вычислить, что возвращает специфическая функция, или произвести единичный запуск задания для обновления данных. Задания не могут наследоваться или вызываться из кода как методы.

### 9.1 Создание заданий

Задание создается с модификатором `static` для того, что бы они были запускаемыми (runable), например, горячей клавишей. Без модификатора `static` компилятор будет расценивать задание как метод. При создании нового задания автоматически добавляется `Args` в профиль параметров. Профиль в точности такой же, что и запускающий метод класса (например, `main`), который производит запуск класса. Задание не может возвращать значения, как метод. Вот почему задание создается с ключевым словом `void`.

```
static void Jobs_MyJob(Args _args)
{
;
    info("Test of job.");
}
```

Задание может вызываться через пункт меню, делая возможным запуск задания из главного меню. Это бывает необходимо, например, при создании или изменении данных, в конфигурации Ахарта, где нет лицензионного кода на разработку MorphX. Задание нельзя перетащить в узел `menu item` как формы, отчеты или запускаемые классы. Вместо этого вам нужно создать пункт меню вручную. Причина этого в том, что задания не предназначены для размещения в меню, а так же для того, чтобы вы лишний раз подумали перед добавлением задания в меню. Вы можете перетащить задание в узел *Classes*. Этим действием Вы создадите запускаемый класс (runable class) с кодом из вашего задания. Это достаточно просто. И все что останется, так это создать пункт меню для класса. Если поместить задание в меню, то вам следует быть уверенным, что данные не повредятся запуском задания второй раз.

```
static void Jobs_ExecutingJob(Args _args)
{
    Args          args;
;

    args = new Args();
    args.name(identifierStr(Jobs_MyJob));

    new menuFunction(menuItemActionStr(Jobs_MyJob), MenuItemType::Action).run(args);
}
```

Создав пункт меню для задания, вы можете обращаться к нему уже в вашем коде. Пример показывает выполнение задания `Jobs_MyJob`. Отметьте, что название задания в `args.name()` определяется использованием встроенной функции `indentifierStr()`, так как не существует специфической функции для заданий. `Args` может использоваться для передачи курсора, или другого параметра в вызываемое задание. Это может быть необходимо, если ваше задание вызывает форму с параметрами, а форме и заданию необходима текущая запись для определения выполнения кода в задании. Передача курсора записи в вызывающий пункт меню разъясняется далее в главе **Формы**.

## 9.2 Резюме

Теперь у вас есть представление об использовании заданий в коде и необходимости размещения кода в методах.



## 10 Меню и Пункты меню

Для того чтобы объекты, такие как формы, отчеты и исполнимые классы, были доступны пользователям приложения, их помещают в меню. Часто пользователи не имеют доступа к репозитарию, то есть меню позволяют контролировать ту функциональность, которая доступна пользователям, в том смысле, что в меню добавляются только те объекты, которые полностью готовы к использованию.

Для того, чтобы иметь возможность вызова объекта из меню, необходимо выполнить 2 шага. Сначала необходимо создать пункт меню, а потом добавить его в меню, из которого он будет вызываться. Причиной наличия двух шагов является возможность создания более одного пункта меню для каждого объекта. Пункты меню могут иметь разные свойства, тем самым вызывая различное поведение вызываемого объекта.

### 10.1 Пункты меню

Пункты меню в АОТ сгруппированы в трех различных узлах. Группировка всего лишь логическая, также она отображает соответствующую группе иконку объекта в меню. Однако следует стараться располагать пункты меню в узле *display* для открытия форм, в узле *output* для запуска отчетов, а в узле *action* – для исполнимых классов. При использовании класса для запуска формы или отчета, следует все равно использовать пункты меню из узлов *display* и *output* соответственно. Пункт меню, а соответственно и иконка, отображаемая в меню, должны соответствовать тому объекту, который пользователь увидит на экране при активации пункта меню.

Самым простым способом создания пункта меню является перетаскивание выбранного объекта в узел соответствующего типа. Это действие приведет к созданию пункта меню с таким же именем, как и выбранный объект. Все, что остается сделать, это указать в его свойствах метку и настроить права пользователей. Если в свойствах пункта меню не будет указана метка, то в меню этот объект будет отображаться с таким же именем, которое он носит в АОТ, только с префиксом \*.

---

**Примечание:** Следует всегда проверять работоспособность ваших модификаций, вызывая их с помощью пунктов меню, а не непосредственно из АОТ, так как при этом вы будете видеть то поведение ваших модификаций, с которым столкнутся пользователи системы, которые будут пользоваться вашей модификацией.

---

Пункты меню являются одними из наиболее используемых объектов для настройки прав доступа пользователей. Свойства **ConfigurationKey** и **SecurityKey** используются для определения списка пользователей и групп пользователей, которые могут использовать данный пункт меню. Если пользователь не имеет прав на активацию пункта меню, то этот пункт меню не будет отображаться в меню для этого пользователя. Эта мощная функциональность обеспечивает отображение только тех меню, которые доступны пользователю. Соответственно, пользователь будет видеть сокращенный вид меню, в котором будет отображаться только та

функциональность приложения, которая предназначена для данного пользователя. Помимо пунктов меню обычно для настройки прав доступа к объектам используются еще и таблицы. Более детальную информацию об установке свойств конфигурационных ключей и ключей контроля доступа можно получить в главе **Словарь данных [Data Dictionary]**.

---

**Примечание:** MorphX автоматически определяет клавишу быстрого запуска (hotkey) для создаваемых пунктов меню. Для изменения клавиши быстрого запуска по умолчанию необходимо вставить символ ‘&’ перед тем символом, которые необходимо использовать, к примеру: Transacti&ons. Это не сработает при изменении метки на пункте меню. Необходимо менять свойство Text непосредственно на элементе формы типа MenuItemButton.

---

Пункт меню может быть активирован из меню, формы для вызова другого объекта или из кода X++. Объекты, используемые в пользовательском интерфейсе, такие как формы или отчеты, всегда следует вызывать с использованием пунктов меню. Можно, конечно, вызвать ту же форму или отчет из кода без использования пункта меню, но при этом придется вручную проверять, имеет ли пользователь доступ к вызываемому объекту, чего не нужно делать для пунктов меню.

```
static void MenuItems_ExecuteObject(Args _args)
{
;
    new menuFunction(menuItemDisplayStr(CustTable), MenuItemType::Display).run();
}
```

В приведенном примере вызывается форма CustTable через одноименный пункт меню типа display. Форма будет открыта только в том случае, если пользователь обладает правами на форму Клиенты. Метод run(), который используется для активации пункта меню, может принимать объект класса Args в качестве параметра. К примеру, можно отфильтровать открываемую форму, передав через Args нужную запись в вызываемый объект. Подробно о классе Args рассказывается в главе **Классы**.

Еще одним способом для передачи значения в вызываемый объект является использование других свойств пункта меню. Свойства **EnumTypeParameter** и **EnumParameter** используются для определения типа перечислимого и конкретного значения этого типа. Передача перечислимого типа в качестве параметра в пункте меню позволяет повторно использовать объекты, такие как форма, для других целей, вместо создания новой формы с подобным строением. Эта техника используется в приложении при вызове журналов. Взгляните на пункты меню типа display, которые начинаются с LedgerJournalTable. Вы увидите, что все эти пункты меню вызывают одну и ту же форму LedgerJournalTable с передачей в нее различных значений перечислимого типа LedgerJournalType. В форме LedgerJournalTable, перечислимый тип используется для определения поведения формы. Это хороший способ организации кода, так как это облегчает создания нового типа журнала. Для этого необходимо будет всего-навсего добавить новое значение перечислимого типа LedgerJournalType, создать новый пункт меню, передав в него новое значение и добавить проверку на новое значение типа журнала в форму LedgerJournalTable.

Перечень всех свойств пунктов меню приведен в разделе **Свойства в Приложении**.

## 10.2 Меню

Для того чтобы создать меню, достаточно перетащить в него требуемые пункты меню. Разделители и подменю создаются нажатием правой кнопки мыши на узле меню и выбором пункта Создать. Вместо того чтобы создавать подменю и перетаскивать в него пункты меню, стоит подумать об использовании Ссылки меню. Ссылка меню – это указатель на другое меню. В списке объектов, которые можно создать в меню, присутствует и создание Ссылки меню. При выборе этого пункта всплывает окошко с существующими меню, которые можно перетащить в создаваемое меню, как ссылку. Обратите внимание, что если вместо этого вы просто перетащите меню непосредственно из АОТ, то выбранное меню будет добавлено не как Ссылка меню, а как подменю.

При добавлении пункта меню в меню скорее всего вы будете модифицировать уже существующее меню. Для того, чтобы максимально облегчить процесс обновления меню, рекомендуется создать отдельное меню для создаваемых пунктов меню, и добавить его как ссылку в существующее меню. Так следует поступать по следующим двум причинам. Во-первых, Ваши модификации будут отделены от стандартных меню, а во-вторых, таким образом, Вы сможете повторно использовать ваше меню в других стандартных меню.

Конфигурационные ключи и ключи контроля доступа могут быть указаны для всего меню целиком в его свойствах. Однако так делать не рекомендуется. Вместо этого, следует определять настройки доступа пользователей только на пунктах меню, так как при этом вне зависимости от того, откуда вызывается этот пункт меню, будут проверяться права пользователя на его использование. Во-вторых, устанавливая права пользователей только на одном типе объектов, вы значительно упростите задачу их поддержки.

---

**Примечание:** Согласно правилам хорошего тона программирования (Best Practice) любое меню должно содержать не более 5 пунктов меню. Остальные пункты меню должны быть сгруппированы в подменю со следующими названиями: Журналы, Запросы, Отчеты, Периодические операции и Настройки.

---

### Определение названия объекта в АОТ из меню

Одной из первых задач при изменении какого-либо объекта является поиск этого объекта в АОТ, для чего необходимо знать название этого объекта. Скорее всего, Вы знаете, откуда вызывается данный объект в Главном Меню. Для этого можно пойти двумя путями: Так как Вам известен полный путь к объекту в меню, то вы можете просто открыть соответствующее меню в АОТ и пройти по его узлам вглубь вплоть до искомого пункта меню. Далее просмотреть свойства этого пункта меню, где свойства **Class** и **Object** укажут на конкретный объект в АОТ. (первое свойство определяет тип объекта, а второе – его название)

Можно пойти и другим путем: сделать правый щелчок мыши по открытой форме или диалогу и выбрать в контекстном меню пункт *Настройка*. Это откроет форму

пользовательских настроек объекта, как показано на **рисунке 47**:

**Пользовательские настройки.** Перейдите на закладку *Информация*. Первые три поля отображают название объекта в АОТ и название пункта меню, который был использован для вызова этой формы или класса. Нажав на кнопку *Правка* справа от названия объекта Вы откроете его в АОТ. Обратите внимание, что эта опция доступна только для форм и исполнимых классов и не может быть использована с запросами и отчетами.

The screenshot shows a 'User setup' window with two tabs: 'Layout' and 'Information'. The 'Information' tab is active. It contains three input fields with 'Edit' buttons: 'Form name' (CustTable), 'Caller' (empty), and 'Menu item' (CustTable). Below these are two columns of metadata: 'Created' and 'Modified'. Each column has three fields: 'By' (3.0), 'Date' (25-06-2002 / 26-06-2002), and 'Time' (06:49:29 / 09:33:04). At the bottom is a 'Version' section with three fields: 'Basic version' (0), 'Version' (0), and 'Save count' (0).

Рисунок 47: Форма 'Пользовательские настройки'

Как видно из описанного выше, оба способа поиска названия объекта в АОТ являются довольно медленными. К сожалению, это единственный выход, если Вы знаете только путь к объекту в меню. Однако так как MorphX предоставляет доступ к своему коду, можно с использованием системных классов изменить поведение системы для увеличения удобства получения информации из АОТ.

## 10.3 Резюме

В этом разделе Вы познакомились с тем, как вызываются объекты с использованием пунктов меню, увидели важность использования для этих целей именно пунктов меню, так как с помощью них контролируются права пользователей. При добавлении пунктов меню в меню и создании новых меню Вы теперь должны понимать, как лучше организовать их для того, чтобы можно было их повторно использовать и упростить процесс обновления приложения.

## 11 Ресурсы

Ресурсы используются для хранения любого типа файлов. Вместо того чтобы хранить используемые в модификациях картинки или файлы любого другого типа в файловой системе, эти файлы могут быть помещены в узел AOT *Resources*. Ресурсы в этом узле не полностью интегрированы в систему, как, к примеру, формы или отчеты. Поэтому при использовании ресурсов, приходится обращаться к некоторым общим системным классам для перебора узлов AOT. В некоторых случаях приходится временно экспортировать ресурс перед использованием сохраненного файла. Узел ресурсов был впервые введен в систему в версии Ахарта 3.0. Это, скорее всего, является причиной такого незначительного количества методов для использования ресурсов. Несмотря на это, ресурсы полезны, так как, используя их, нет нужды обращаться к файлам по определенным путям файловой системы, и, по крайней мере, все необходимые для модификаций файлы будут находиться в одном месте.

Ресурсы, которые хранятся в узле *Resources* в AOT, не следует путать с теми, которые хранятся в файле ядра. Точечные рисунки, которые используются для иконок узлов AOT, хранятся в файле ядра и к ним привязан код ресурса, как ссылка на этот ресурс. Для полного обзора ресурсов системы, включая их идентификаторы, запустите форму *Tutorial\_Resources*.

### 11.1 Использование ресурсов

Для того чтобы добавить ресурс, необходимо нажать правой кнопкой мыши по узлу *Resources* в AOT и выбрать «Создать из файла». При этом откроется диалог выбора файла, в котором необходимо указать путь к файлу, который будет сохранен в узле AOT *Resources*. Информацию о добавленном ресурсе можно просмотреть выбрав в контекстном меню команду Открыть. Тип ресурса будет отображаться в окне предварительного просмотра. Если же добавленный ресурс является изображением, то оно также будет отображаться в окне просмотра.

При вставке изображения в форму или отчет, у Вас в распоряжении есть несколько источников, содержащих изображения: это один из системных ресурсов, изображение, сохраненное в таблице, файл, хранящийся в файловой системе или же изображение, хранимое в узле ресурсов. В зависимости от конкретного случая оптимальным может быть любой из вариантов. В случае, если необходимо перемещать модификации вместе с изображениями или вы хотите убедиться, что будет использована именно указанное изображение, то стоит подумать о его хранении в ресурсах в AOT.

---

#### Пример 1: Загрузка изображения

##### Элементы проекта MORPHXIT Resources

- Ресурс, MORPHXIT
- Форма, Resources\_LoadBitmap

В этом примере в узел ресурсов в АОТ будет загружено изображение, которое затем будет использовано для отображения на форме. Для запуска приведенного примера необходимо создать ресурс типа Изображение и изменить его название на MORPHXIT.

1. Создайте новую форму и переименуйте ее в “Resources\_LoadBitmap”.
2. Перейдите к узлу *Design* и установите его свойства Width и Height в значение «Column width» и «Column height».
3. Добавьте управляющий элемент window в дизайн формы. Этот управляющий элемент будет использоваться для отображения изображения из узла ресурсов MORPHXIT. Установите свойство AutoDeclaration этого элемента в значение Yes.
4. Создайте новый метод формы showResource(). Метод будет использоваться для загрузки ресурса. Код метода должен быть таким, какой приведен ниже:

```
public void showResource()
{
    ResourceNode    resourceNode;
    Container       imageContainer;
    Image           image;
;

    resourceNode = SysResource::getResourceNode(resourceStr(MORPHXIT));
    resourceNode.AOTload();

    imageContainer    = SysResource::getResourceNodeData(resourceNode);
    image             = new Image(imageContainer);

    imageCtrl.widthValue(image.width());
    imageCtrl.heightValue(image.height());
    imageCtrl.image(image);
}
```

5. Последним шагом является вызов созданного метода showResource() из перекрытого метода run() формы так, как показано ниже:

```
public void run()
{
    super();

    element.showResource();
}
```

---

Класс SysResource и системный класс ResourceNode используются для получения доступа к ресурсам в АОТ. Класс SysResource содержит несколько полезных статических методов, которые используются как для загрузки, так и для сохранения ресурсов. Обычно SysResource используется для получения ссылки на узел ресурса в АОТ и ее возврата в объект класса ResourceNode. Если более детально изучить методы класса SysResource, то Вы увидите, что в методах класса

используется системный класс `TreeNode`. Класс `TreeNode` является базовым классом, который используется для переходов по любым узлам в AOT.

Обратите внимание, что после инициализации `ResourceNode` в методе `showResource()` необходимо вызвать метод `AOTLoad()`. Если этот вызов пропустить, то изображение не будет отображено на форме. Элемент формы `window`, добавленный для отображения рисунка, инициализируется с помощью системного класса `Image`. Высота и ширина этого элемента управления устанавливается таким образом, что будет произведена подгонка размеров изображения под размер элемента формы.

В приведенном выше примере использовался файл изображения типа `JPG`. При использовании ресурсов поддерживаются не все форматы файлов - к примеру, нельзя использовать файлы с расширением `GIF`. При попытке использования изображения неверного формата, такого как `GIF`, произойдет ошибка. Если Вы создали модуль или какое-то решение, расширяющее функциональность существующего модуля, ресурсы могут пригодиться для добавления файлов, необходимых для вашей разработки, даже демонстрационных данных. Демонстрационные данные и данные по умолчанию часто распространяются совместно с приложением для того, чтобы облегчить ознакомление с системой. Необходимо будет просто экспортировать нужные данные, и создать узлы ресурсов для экспортированных файлов данных. Такое решение используется в стандартном приложении для настройки `Enterprise Portal` и импорта ролей для пользователей `Enterprise Portal`.

---

#### Пример 2: Данные по умолчанию

##### Элементы проекта MORPHXIT\_Resources

- Ресурс, `CustGroup_Data`
- Ресурс, `CustGroup_Def`
- `Job, Resources_DefaultData`

В этом примере будет показано, как добавить данные по умолчанию, экспортированные пользователем, в качестве ресурсов. Данные будут автоматически импортированы. Перед импортом будет отображен диалог с вопросом о существовании каких-либо данных в таблице, в которую производится импорт. Для простоты примера данные будут импортироваться только в таблицу `CustGroup`.

1. Запустите экспорт данных таблицы `CustGroup`, используя пункт меню экспорта, который расположен в Главном меню в Администрирование | Периодические операции | Экспорт/Импорт данных | Экспорт.
2. Создайте два узла ресурсов для хранения файлов данных и определения. Узлы назовем соответственно `CustGroup_Data` и `CustGroup_Def`.
3. Создадим задание (`job`), которое назовем `Resources_DefaultData`, и которое будет содержать следующий код:

```
static void Resources_DefaultData(Args _args)
{
```

```

    FilePath      tempPath;
    ResourceNode   resourceDefinitionFile;
    ResourceNode   resourceDataFile;
    SysDataImport  sysDataImport;
;
    tempPath = xinfo::directory(directoryType::Temp);

    resourceDefinitionFile = sysResource::getResourceNode(resourceStr(CustGroup_Definition));
    resourceDataFile       = sysResource::getResourceNode(resourceStr(CustGroup_Data));

    sysResource::exportResource(resourceDefinitionFile, tempPath);
    sysResource::exportResource(resourceDataFile, tempPath);

    sysDataImport = sysDataImport::newFilename(tempPath + resourceDefinitionFile.filename());
    sysDataImport.parmLoadAll(true);

    sysDataImport.run();
}

```

Утилита для импорта, которая может быть запущена из Главного меню, используется для импорта данных из ресурсов. Перед тем, как применять утилиту импорта данных к сохраненным в ресурсах файлам, файлы необходимо экспортировать на жесткий диск. Файлы данных временно экспортируются во временную директорию приложения Ахapta. Это достигается с помощью системного класса SysResource, с помощью которого получают ссылки на узлы ресурсов в АОТ, которые затем экспортируются в файлы. Экспортированные файлы можно после этого импортировать с помощью утилиты импорта. При вызове утилиты импорта из Главного меню открывается диалог выбора настроек и пути к файлу импорта. В приведенном примере диалог не нужен, так как вся необходимая информация уже в наличии, соответственно он и не используется. Вот и все. Это поистине очень удобный для пользователя интерфейс импорта данных по умолчанию, значительно превосходящий неумелые попытки пользователя отыскать нужную утилиту в Главном меню и выполнить импорт вручную.

## 11.2 Резюме

Ресурсы не очень распространены в MorphX. Возможно, именно по этой причине про их использование часто забывают и находят другие пути решения поставленной задачи. Этот раздел проливает свет на то, как могут использоваться ресурсы, и как, используя ресурсы, можно получить доступ к файлам, не привязываясь при этом к конкретному пути на жестком диске.



## 12 Приложение. Свойства объектов

В этом разделе Вы найдете обзор всех свойств объектов, которые доступны в АОТ. В каждом подразделе свойства отсортированы в возрастающем порядке. Объекты с похожими и идентичными свойствами сгруппированы вместе для сокращения размера обзора. Некоторые из подразделов содержат дополнительную колонку, которая называется *Tip* и используется для указания, какие именно объекты имеют указанное свойство. Если в этой колонке указано слово *все*, то это значит, что это свойство доступно всем объектам подраздела.

### 12.1 Свойства объектов Словаря данных

Свойства для метаданных представлений приведены в подразделе **Свойства запросов**.

#### Таблицы, Табличные карты соответствия и Представления

Свойство	Описание
CacheLookup	Используется для указания алгоритма кэширования, который будет применен при выборке конкретной записи в запросе с использованием выражения WHERE.
ChangedBy	Пользователь, который последний вносил изменения в таблицу.
ChangedDate	Дата последнего изменения таблицы.
ChangedTime	Время последнего изменения таблицы.
ClusterIndex	Индекс, который следует использовать как кластерный. В свойстве могут использоваться только уникальные (не допускающие повторяющихся записей) индексы.
ConfigurationKey	Используется для указания конфигурационного ключа на таблице.
CreatedBy	Пользователь, создавший запись. Если включено, то системное поле <code>createdBy</code> будет заполнено при добавлении записи.
CreatedDate	Дата создания записи. Если включено, то системное поле <code>createdDate</code> будет заполнено при добавлении записи.
CreatedTime	Время создания записи. Если включено, то системное поле <code>createdTime</code> будет заполнено при добавлении записи.
CreatedTransactionId	Если включено, то идентификатор транзакции, в которой была создана запись, будет сохраняться в таблице при создании записи.
CreateRecIdIndex	При включении будет создан индекс <code>RecId</code> , содержащий одно поле <code>recId</code> .
CreationDate	Дата создания таблицы. Заполняется автоматически при создании
FormRef	Название пункта меню типа <code>display</code> , который следует использовать при переходе к основной таблице на формах.

ID	Код таблицы.
Label	Метка таблицы. Это метка, по которой пользователи будут идентифицировать таблицу в приложении.
LockedBy	Пользователь, который в текущий момент заблокировал таблицу.
MaxAccessMode	Устанавливает уровень доступа к таблице. При использовании таблицы в качестве источника данных на форме уровень доступа на форме не может превышать уровень доступа на таблице.
ModifiedBy	Если включено, то пользователь, последний модифицировавший запись, будет сохраняться в одноименном системном поле.
ModifiedDate	Если включено, то дата последней модификации записи будет сохраняться в одноименном системном поле.
ModifiedTime	Если включено, то время последней модификации записи будет сохраняться в одноименном системном поле.
ModifiedTransactionId	Если включено, то идентификатор транзакции, в которой была последний раз изменена запись, будет сохраняться.
Name	Название таблицы, карты соответствия или табличного представления.
PrimaryIndex	Индекс, который следует использовать как первичный. В свойстве могут использоваться только уникальные (не допускающие повторяющихся записей) индексы.
SaveDataPerCompany	Данные будут сохраняться отдельно по компаниям или в одной компании.
SecurityKey	Используется для указания ключа контроля доступа на таблице.
Systemtable	Если включено, то система будет считаться системной.
TableContents	Определяет, какие данные будет содержать таблица.
TableGroup	Устанавливает группу, к которой принадлежит таблица. Это свойство используется ядром для определения оптимального плана выполнения запроса при использовании соединений (joins).
Temporary	Следует включить для временных таблиц.
TitleField1	Первое из полей, которые используются при отображении на заголовке формы.
TitleField2	Второе из полей, которые используются при отображении на заголовке формы.

### Поля таблицы, карты соответствия

Свойство	Тип	Описание
Adjustment	String	Указывает на горизонтальное

		выравнивание значения в случае, если не указан расширенный тип данных.
AliasFor	Все	Определение поля таблицы, для которого поле является псевдонимом (alias)
AllowEdit	Все	Если включено, то поле редактируемо
AllowEditOnCreate	Все	Если включено, то поле редактируемо при создании записи
ConfigurationKey	Все	Используется для указания конфигурационного ключа на поле.
EnumType	Перечислимый тип	Укажите перечислимый тип, который будет использоваться для поля таблицы.
ExtendedDataType	Все	Укажите расширенный тип данных, который будет использоваться для поля таблицы.
FieldUpdate	Integer Real	Значением по умолчанию является Absolute, при использовании которого текущее значение поля будет перезаписано при обновлении. Значение Relative позволит нескольким пользователям изменить одну и ту же запись, при этом введенные значения будут просуммированы и записаны в это поле таблицы.
GroupPrompt	Все	Метка поля, которая будет использоваться при отображении поля таблицы в группе полей.
HelpText	Все	Подсказка к полю, которая выводится в статусной строке.
ID	Все	Внутренний код поля.
Label	Все	Метка, которая будет использоваться в формах и отчетах для поля. Имеет приоритет по сравнению с меткой расширенного типа данных поля.
Mandatory	Все	Если включено, то пользователю необходимо будет обязательно ввести значение в это поле. Заполнение будет проверено в методе validateWrite таблицы.
Name	Все	Название поля таблицы.
SaveContents	Все	Если включено, то значение поля таблицы будет сохранено в базе

		данных, иначе будет виртуальным полем.
StringSize	String	Используется для указания длины поля таблицы в символах в случае, если не указан расширенный тип данных.
Type	Все	Отображает базовый тип поля.
Visible	Все	Если включено, то поле будет отображаться на формах и отчетах.

## Поля Табличных представлений

Свойство	Тип	Описание
Aggregation	Все	Служит для выбора агрегатной функции, которая будет применена к полю.
ConfigurationKey	Все	Используется для указания конфигурационного ключа на поле.
DataField	Все	Служит для связи с полем выбранного источника данных.
DataSource	Все	Источник данных, используемый для поля. Данные в поле будут выбираться из указанного источника.
EnumType	Перечислимый тип	Отображается перечислимый тип, использованный для поля.
ExtendedDataType	Все	Отображается расширенный тип данных, использованный для поля.
GroupPrompt	Все	Метка, которая будет использоваться при отображении поля в группе полей.
HelpText	Все	Подсказка к полю, которая будет отображаться в статусной строке.
ID	Все	Внутренний код поля представления.
Label	Все	Используется для указания метки, которая будет использоваться для поля. Имеет приоритет над меткой, указанной на расширенном типе.
Name	Все	Название поля представления.
StringSize	String	Отображается размер поля в символах
Type	Все	Отображается базовый тип поля.

## Группа полей таблиц, карт соответствия, представлений

Свойство	Описание
Name	Название группы полей.
Label	Метка группы полей, которая отображается на формах и отчетах.

## Индекс таблицы

Свойство	Описание
Name	Название индекса таблицы.
AllowDuplicates	При значении «Yes» в таблице могут присутствовать две записи с одинаковыми полями, которые принадлежат индексу, то есть индекс будет не уникальным. Если в таблице отсутствует уникальный индекс, то Ахарта создаст его автоматически из наиболее короткого индекса таблицы добавлением к нему поля RecId.
ConfigurationKey	Используется для указания конфигурационного ключа на индекс.
Enabled	При значении «Yes» индекс активен и используется.
ID	Внутренний код индекса.

## Отношение на таблице

Свойство	Описание
Name	Название табличного отношения.
Table	Таблица, с которой настраивается отношение.
Validate	Определяет, должна ли проверяться ссылка отношения при заполнении полей в формах.

## Поле отношения на таблице

Свойство	Тип	Описание
Field	Нормально Поле фиксировано	Устанавливает одно из полей, по которому будет настраиваться отношение между таблицами.
RelatedField	Нормально Поле ссылки фиксировано	Устанавливает одно из полей, по которому будет настраиваться отношение между таблицами.
Value	Поле фиксировано Поле ссылки фиксировано	Указывает конкретное значение одного из полей отношения (только целочисленных полей). Обычно используется для установки значения

		связи перечислимого типа.
--	--	---------------------------

### Действие при удалении таблицы (DeleteAction)

Свойство	Описание
DeleteAction	Действие, которое будет выполнено при удалении записи таблицы.
Table	Таблица, к которой будет применено указанное действие.

### Mapping табличной карты соответствия

Свойство	Описание
MappingTable	Устанавливает таблицу, к которой будет настраиваться соответствие.

### Поле Mapping карты соответствия

Свойство	Описание
MapField	Устанавливает поле в табличной карте соответствия.
MapFieldTo	Устанавливает поле в таблице, к которому настраиваться соответствие поля карты соответствия.

### Расширенный тип данных

Свойство	Тип	Описание
Adjustment	String	Устанавливает горизонтальное выравнивание текста в случае, если расширенный тип данных не наследуется от другого расширенного типа.
Alignment	Все	Устанавливает горизонтальное выравнивание информации.
AllowNegative	Integer Real	Устанавливает возможность ввода отрицательных значений.
ArrayLength	Все	Отображает информацию о количестве элементов в массиве расширенного типа данных.
AutoInsSeparator	Real	Используется для указания системе автоматической вставки десятичного разделителя.
ButtonImage	Все	Графическое изображение, которое отображается в правом углу элемента ввода данных в случае, если возможен выбор значения из справочника.
ChangeCase	String	Устанавливает регистр ввода текста

		при вводе значения.
ConfigurationKey	Все	Устанавливает конфигурационный ключ для расширенного типа данных.
DateDay	Date	Устанавливает способ отображения дня в дате. Региональные настройки Windows используются по умолчанию.
DateFormat	Date	Устанавливает формат вывода даты. Региональные настройки Windows используются по умолчанию.
DateMonth	Date	Устанавливает способ отображения месяца в дате. Региональные настройки Windows используются по умолчанию.
DateSeparator	Date	Устанавливает разделители при отображении даты. Региональные настройки Windows используются по умолчанию.
DateYear	Date	Устанавливает способ отображения года в дате. Региональные настройки Windows используются по умолчанию.
DecimalSeparator	Real	Устанавливает десятичный разделитель при выводе действительного числа. Региональные настройки Windows используются по умолчанию.
DisplaceNegative	Real	Устанавливает, нужно ли использовать смещение при выводе отрицательных значений в элементах ввода.
DisplayHeight	String	Устанавливает максимальное количество строк, которые могут отображаться в одно время в элементе ввода текста.
DisplayLength	Все	Устанавливает максимальное количество символов, которые могут отображаться в одно время в элементе ввода текста.
EnumType	Перечислимый тип	Устанавливает перечислимый тип расширенного типа.
Extends	Все	Устанавливает расширенный тип, от которого будут наследоваться определенные свойства, такие как Alignment и StringSize.
FormatMST	Real	Форматировать содержимое элемента ввода как сумму в основной валюте компании.

FormHelp	Bce	Форма, которая будет использоваться при выводе всплывающего списка для элемента ввода.
HelpText	Bce	Подсказка к полю, которая будет выведена в статусной строке
Label	Bce	Метка, которая будет использоваться.
Name	Bce	Название расширенного типа данных.
NoOfDecimals	Real	Устанавливает количество цифр после запятой, которые будут отображаться.
RotateSign	Integer Real	Используется для визуального инвертирования знака чисел.
ShowZero	Integer Real	Устанавливает, будут ли отображаться в элементах ввода нулевые значения.
SignDisplay	Integer Real	Устанавливает способ отображения знака минус в числе.
StringSize	String	Устанавливает длину текста в символах.
Style	Перечислимый тип	Устанавливает, как будет выглядеть расширенный тип на форме.
ThousandSeparator	Real	Устанавливает разделитель для разрядов тысяч при выводе действительного числа. Региональные настройки Windows используются по умолчанию.
TimeFormat	Time	Устанавливает формат вывода времени. Региональные настройки Windows используются по умолчанию.
TimeHours	Time	Устанавливает необходимость отображения часов при выводе времени. Региональные настройки Windows используются по умолчанию.
TimeMinute	Time	Устанавливает необходимость отображения минут при выводе времени. Региональные настройки Windows используются по умолчанию.
TimeSeconds	Time	Устанавливает необходимость отображения секунд при выводе времени. Региональные настройки Windows используются по умолчанию.
TimeSeparator	Time	Устанавливает разделитель при выводе времени. Региональные настройки Windows используются по умолчанию.



## Перечислимый тип

Свойство	Описание
ChangedBy	Пользователь, который последний модифицировал перечислимый тип.
ChangedDate	Дата последнего изменения перечислимого типа.
ChangedTime	Время последнего изменения перечислимого типа.
ConfigurationKey	Устанавливает конфигурационный ключ для перечислимого типа.
CreatedBy	Пользователь, создавший перечислимый тип.
CreatedTime	Время создания перечислимого типа.
CreationDate	Дата создания перечислимого типа.
DisplayLength	Задаёт максимальное количество символов, отображаемых в одно время в элементе ввода.
Help	Подсказка к элементу ввода, которая выводится в статусной строке.
ID	Внутренний код перечислимого типа.
Label	Метка перечислимого типа. Это тот текст, который будет отображаться пользователю при работе.
LockedBy	Пользователь, который заблокировал перечислимый тип.
Name	Название перечислимого типа.
Style	Устанавливает графическое представление перечислимого типа.
UsedEnumValue	Определяет, будут ли использоваться значения перечислимого типа. Если отключить это свойство, то Axapta автоматически сгенерирует значения элементов.

## Элемент перечислимого типа

Свойство	Описание
ConfigurationKey	Используется для назначения конфигурационного ключа элементу перечислимого типа.
EnumValue	Это целочисленное значение элемента, которое будет сохранено в БД.
Label	Метка, которая будет отображаться для этого элемента перечислимого типа в формах и отчетах.
Name	Название элемента перечислимого типа, которое может быть использовано в коде X++ вместо числовых значений.

## Лицензионные коды

Свойство	Описание
ChangedBy	Пользователь, который последний модифицировал лицензионный код.
ChangedDate	Дата последнего изменения лицензионного кода.
ChangedTime	Время последнего изменения лицензионного кода.
CreatedBy	Пользователь, создавший лицензионный код.
CreatedTime	Время создания лицензионного кода.
CreationDate	Дата создания лицензионного кода.
Group	Группа, к которой принадлежит лицензионный код (Система, Модули, Партнерские модули, Интернет, Языки).
ID	Внутренний код лицензионного кода.
Label	Метка лицензионного кода. Это тот текст, который будет отображаться пользователю при работе с лицензионным кодом.
LockedBy	Пользователь, который заблокировал лицензионный код.
Type	Большинство лицензионных кодов имеют логическое значение (true и false). Это не так только при подсчете количества лицензий, таких как количество пользователей или пользователей COM.

## Конфигурационные ключи, ключи контроля доступа

Свойство	Тип	Описание
ChangedBy	Оба	Пользователь, который последний модифицировал ключ.
ChangedDate	Оба	Дата последнего изменения ключа.
ChangedTime	Оба	Время последнего изменения ключа.
ConfigurationKey	Ключ контроля доступа	Используется для указания конфигурационного ключа.
CreatedBy	Оба	Пользователь, создавший ключ.
CreatedTime	Оба	Время создания ключа.
CreationDate	Оба	Дата создания ключа.
ID	Оба	Внутренний код ключа.
Label	Оба	Метка ключа. Это тот текст, который будет отображаться пользователю при работе с ключом.
LicenseCode	Конфигурационный ключ	Лицензионный код, который используется для активации ключа.

LockedBy	Оба	Пользователь, который заблокировал ключ.
Name	Оба	Название ключа.
ParentKey	Оба	Ключ – родитель текущего ключа. Если отключить родительский ключ, то это отразится и на текущем ключе.

## 12.2 Свойства форм

### Источник данных формы

Свойство	Описание
AllowCheck	Определяет, будут ли проверять конфигурационные ключи и ключи контроля доступа при запуске формы.
AllowCreate	Включает возможность добавления записей таблицы источника данных.
AllowDelete	Включает возможность удаления записей таблицы источника данных.
AllowEdit	Включает возможность редактирования записей таблицы источника данных.
AutoNotify	Следует установить значение No в случае, если не используется запрос формы. Иначе устанавливайте значение Yes, так как оно используется для вывода сообщений при модификации записей.
AutoQuery	При отключенном свойстве пользователь не будет иметь возможности использовать форму настройки фильтров, а также форму поиска.
AutoSearch	Определяет, будет ли производиться автоматическая выборка записей при запуске формы.
Company	Если в свойстве указано значение, то данные будут выбраны из указанной компании.
CounterField	Используется для определения поля-счетчика для записей, которые вставляются с помощью источника данных. Это свойство используется, если необходимо поддерживать сортировку записей в той последовательности, в которой они были добавлены пользователями. Для этого в таблицу, используемую источником данных, необходимо добавить поле базового типа real, и это поле должно быть выбрано в свойстве CounterField источника. MorphX автоматически установит значение в это поле при добавлении записи. Форма <i>SalesTable</i> использует свойство CounterField при вставке строк заказа.
DelayActive	При установке значения Yes в этом свойстве, исполнение кода и связывание будет приостанавливаться во время просмотра записей, за счет чего повышается производительность.

Index	Индекс, который используется для сортировки записей при выборке.
InsertAtEnd	При значении Yes в свойстве будет автоматически создаваться запись при выходе пользователя за последнюю запись источника.
InsertIfEmpty	Если в результате выборки не будет возвращено ни одной записи, то, при установке этого свойства, автоматически создастся новая запись.
JoinSource	Источник данных формы, с которым будет объединен источник, в котором устанавливается свойство.
LinkType	Свойство следует использовать совместно со свойством JoinSource. LinkType определяет способ объединения источника с присоединенным источником данных.
Name	Название источника данных формы.
OnlyFetchActive	Устанавливает набор полей, которые будут выбираться запросом. В зависимости от значения свойства будут выбраны или все поля, или только те, которые используются элементами формы.
StartPosition	Указывает, на какую запись (первую или последнюю) должен позиционироваться источник данных.
Table	Таблица, используемая как источник данных.

## Поля источника данных формы

Свойство	Описание
AllowAdd	Устанавливает, разрешается ли пользователю добавлять данный объект во время пользовательской настройки.
AllowEdit	Устанавливает, могут ли данные изменяться посредством данного элемента графического интерфейса.
Enabled	Устанавливает, разрешен ли элемент графического интерфейса.
Mandatory	Устанавливает обязательность ввода значения в данное поле.
Skip	Устанавливает необходимость пропуска элемента графического интерфейса при табуляции.
Visible	Устанавливает, будет ли элемент отображаться на форме. При автоматическом расположении соседних элементов формы они будут автоматически сдвигаться при изменении этого свойства.

## Групповые элементы формы (Контейнеры)

В этом разделе рассмотрены свойства элементов формы ButtonGroup, Group, Tab и TabPage.

Свойство	Тип	Описание
AlignChild	Все	Следует ли включать данный элемент управления в

		выравнивание элементов группового элемента формы, которому он принадлежит.
AlignChildren	Все	При установке этого свойства элементы управления, дочерние по отношению к данному, будут выровнены относительно друг друга.
AlignControl	Все	При установке этого свойства элементы формы будут выровнены по самой длинной метке.
AllowEdit	Все	Управляет доступом на редактирование данных посредством данного элемента управления.
AllowUserSetup	Все	Управляет возможностью пользовательской настройки данного элемента управления.
ArrangeMethod	Все	Устанавливает способ упорядочивания элементов графического интерфейса в данном контейнере.
ArrangeWhen	Все	Устанавливает момент упорядочивания элементов графического интерфейса в данном контейнере.
AutoDataGroup	Group	При установке этого свойства группа может содержать только те поля, которые входят в указанную в свойстве DataGroup группу полей таблицы.
AutoDeclaration	Все	При установке этого свойства будет автоматически объявлена одноименная переменная для доступа к данному элементу из кода X++.
BackgroundColor	Все	RGB значение или название цветовой схемы Windows, которое задает фоновый цвет элемента управления.
BackStyle	Все	Устанавливает стиль фона для элемента графического интерфейса. Позволяет выбрать прозрачный фон для элемента. Это свойство используется для скрытия фона изображений на форме или для установки цвета фона для элемента согласно свойству BackGroundColor.

Bold	ButtonGroup Group	Устанавливает степень «жирности» шрифта, которым выводится текст.
BottomMargin	Bce	Устанавливает размер нижней границы элемента.
Caption	ButtonGroup Group TabPage	Устанавливает метку группового элемента.
ColorScheme	Bce	Устанавливает способ выбора цвета для элементов формы (RGB или цветовая схема Windows).
Columns	Bce	Количество колонок в групповом элементе. Дочерние элементы будут выровнены согласно количеству колонок.
ColumnSpace	Bce	Устанавливает расстояние между столбцами.
ConfigurationKey	Bce	Используется для указания конфигурационного ключа элемента формы.
DataGroup	Group	Название группы полей из используемого источника данных.
DataSource	Bce	Источник данных, который используется для выборки данных в элемент графического интерфейса.
DragDrop	Bce	Устанавливает режим перетаскивания для элемента графического интерфейса.
Enabled	Bce	Устанавливает, разрешен ли элемент графического интерфейса.
Font	ButtonGroup Group	Шрифт отображения текста в элементе графического интерфейса. Если значение свойства оставить пустым, то будет использоваться шрифт по умолчанию.
FontSize	ButtonGroup Group	Размер шрифта, которым выводится текст в элементе графического интерфейса. Если оставить значение свойства пустым, то будет использоваться размер шрифта по умолчанию.
FrameOptionButton	Group	Устанавливает, должна ли рамка

		элемента содержать кнопку выбора вариантов.
FramePosition	ButtonGroup Group	Устанавливает местоположение рамки элемента графического интерфейса.
FrameType	ButtonGroup Group	Устанавливает стиль рамки элемента графического интерфейса.
Height	Bce	Устанавливает высоту элемента графического интерфейса.
HelpText	Bce	Текст, который отображается в строке состояния при выборе элемента интерфейса. Если оставить значение свойства пустым, то будет использоваться подсказка, указанная для соответствующего поля источника данных.
HideIfEmpty	Bce	Делает группу невидимой, если дочерние элементы группы отсутствуют или также невидимы.
Italic	ButtonGroup Group	Текст будет выводиться курсивом.
LabelBold	Group	Устанавливает степень «жирности» шрифта, которым выводится метка.
LabelFont	Group	Устанавливает шрифт отображения текста метки элемента графического интерфейса. Если значение свойства оставить пустым, то будет использоваться шрифт по умолчанию.
LabelFontSize	Group	Устанавливает размер шрифта, которым выводится метка элемента графического интерфейса. Если оставить значение свойства пустым, то будет использоваться размер шрифта по умолчанию.
LabelItalic	Group	Текст метки будет выводиться курсивом.
LabelUnderline	Group	Текст метки будет выводиться с подчеркиванием.
Left	Bce	Устанавливает горизонтальную позицию элемента графического интерфейса. В случае если

		элементы формы имеют горизонтальное выравнивание, и один из них имеет фиксированную позицию слева, то для всех остальных элементов также нужно указать фиксированную позицию.
LeftMargin	Bce	Устанавливает размер левой границы элемента интерфейса.
Name	Bce	Устанавливает название элемента графического интерфейса.
NeededAccessLevel	Bce	Устанавливает уровень доступа, необходимый для активации данного элемента графического интерфейса.
OptionValue	Group	Устанавливает значение опции, если включено свойство FrameOptionButton. Используется для установки начального значения, если в FrameOptionButton выбрано значение Check или Radio.
RightMargin	Bce	Устанавливает размер правой границы элемента интерфейса.
SecurityKey	Bce	Используется для указания ключа контроля доступа для элемента графического интерфейса.
SelectControl	Tab	Делает активным первый элемент новой вкладки при переходе между ними.
ShowTabs	Tab	При отключенном свойстве вкладки отображаться не будут.
SizeHeight	ButtonGroup	Устанавливает, должны ли все кнопки данной группы кнопок иметь одинаковую высоту.
SizeWidth	ButtonGroup	Устанавливает, должны ли все кнопки данной группы кнопок иметь одинаковую ширину.
Skip	Bce	При установке этого свойства элемент графического интерфейса будет пропущен при переходе по клавише табуляции.
Tab	Tab	Устанавливает активную вкладку при открытии формы.
TabAppearance	Tab TabPage	Устанавливает стиль отображения вкладок.



TabAutoChange	Tab TabPage	Не работает. Должно устанавливать возможность перехода к следующей закладке по клавише табуляции.
TabLayout	Tab	Устанавливает способ упорядочивания вкладок. Значение Tunnel не может использоваться на формах, используется только в Web формах.
TabPlacement	Tab	Устанавливает расположение вкладок.
Top	Все	Устанавливает вертикальную позицию элемента графического интерфейса. При установке фиксированного значения оно рассчитывается от элемента формы, который находится выше данного.
TopMargin	Все	Устанавливает размер верхней границы элемента графического интерфейса.
Underline	ButtonGroup Group	Текст, который выводится в данном элементе формы, будет подчеркнут.
VerticalSpacing	Все	Устанавливает расстояние сверху и снизу графического элемента.
Visible	Все	Устанавливает видимость элемента интерфейса. Если следующие графические элементы имеют автоматическое выравнивание, то они будут выровнены с учетом невидимого элемента.
Width	Все	Устанавливает ширину графического элемента. Если в свойстве указано значение по умолчанию, то ширина будет определяться по расширенному типу данных соответствующего поля источника данных.

## Дизайн формы

Свойство	Описание
AlignChild	Устанавливает, следует ли включать дизайн формы в общее выравнивание.

AlignChildren	При установке этого свойства элементы управления, дочерние по отношению к дизайну, будут выровнены относительно друг друга.
AllowDocking	Устанавливает возможность привязки формы к границам главного окна.
AllowUserSetup	Управляет возможностью пользовательской настройки дизайна.
AlwaysOnTop	Устанавливает, всегда ли форма должна находиться поверх других окон.
ArrangeMethod	Устанавливает способ упорядочивания элементов графического интерфейса формы.
ArrangeWhen	Устанавливает момент упорядочивания элементов графического интерфейса формы.
BackgroundColor	RGB значение или название цветовой схемы Windows, которое задает фоновый цвет формы.
BottomMargin	Устанавливает размер нижней границы формы.
Caption	Устанавливает текст, который будет отображаться в заголовке формы.
ColorScheme	Устанавливает способ выбора цвета для элементов формы (RGB или цветовая схема Windows).
Columns	Количество колонок в дизайне формы. Дочерние элементы будут выровнены согласно количеству колонок.
Columnspace	Устанавливает расстояние между столбцами.
DataSource	Источник данных, который используется для выборки данных формы.
Font	Шрифт отображения текста в дизайне формы. Если значение свойства оставить пустым, то будет использоваться шрифт по умолчанию.
Frame	Устанавливает стиль отображения рамки формы.
Height	Устанавливает высоту формы.
HideIfEmpty	Делает дизайн формы невидимым и отображает только заголовок, если дочерние элементы формы отсутствуют или также невидимы.
HideToolbar	Делает невидимыми определенные кнопки формы панели инструментов, предназначенные для навигации по строкам и просмотра документов, связанных с записью.
Imagemode	Управляет способом отображения графики. См. свойство ImageName.
ImageName	Устанавливает имя файла точечного рисунка, который будет использоваться в качестве заднего фона формы.
ImageResource	Устанавливает идентификатор ресурса, который будет использоваться в качестве заднего фона формы.
LabelFont	Шрифт отображения текста метки формы. Если значение свойства оставить пустым, то будет использоваться шрифт по умолчанию.

Left	Устанавливает горизонтальную позицию формы.
LeftMargin	Устанавливает размер левой границы формы.
Mode	Похоже, что не дает никакого эффекта. Это свойство использовалось предшественником Ахарта для установки прав записи на формах.
NeededAccessLevel	Похоже, что не дает никакого эффекта. Необходимый уровень доступа обычно задается с помощью пунктов меню, исходя даже из того, что отображается для этого свойство в качестве подсказки.
RightMargin	Устанавливает размер правой границы формы.
SaveSize	Сохраняет размер дизайна формы при выходе и использует его при следующем открытии формы.
SetCompany	Устанавливает, будет ли изменяться компания при активации формы.
TitleDatasource	Устанавливает источник данных, данные которого будут отображаться в заголовке формы.
Top	Устанавливает вертикальную позицию элемента формы.
TopMargin	Устанавливает размер верхней границы формы.
Visible	Устанавливает видимость всего дизайна формы.
Width	Устанавливает ширину формы.
WindowResize	Устанавливает возможность изменения размеров окна формы.
WindowType	Устанавливает тип окна формы, к примеру с обычной формы во всплывающую форму. У всплывающей формы изменение размера запрещено.

## Управляющие элементы

В этом разделе приведен обзор элементов формы для базовых типов и других элементов графического интерфейса, используемых для отображения данных, такие как поле табличного документа [grid] или кнопка [button].

Свойство	Тип	Описание
ActiveBackColor	Grid	RGB значение или название цветовой схемы Windows, которое задает фоновый цвет при отображении текущей записи.
ActiveForeColor	Grid	RGB значение или название цветовой схемы Windows, которое задает цвет текста при отображении текущей записи.
AlignChild	Design	Устанавливает, следует ли включать элемент формы в общее выравнивание.

AlignChildren	Design	При установке этого свойства управляющие элементы, дочерние по отношению к дизайну, будут выровнены относительно друг друга.
AlignControl	Bce	При установке этого свойства элементы формы будут выровнены по самой длинной метке.
Alignment	DateEdit IntEdit RealEdit StaticText StringEdit TimeEdit	Устанавливает способ выравнивания информации, отображаемой в элементе графического интерфейса. Используется для левого выравнивания данных при вертикальном способе упорядочивания элементов формы.
AllowEdit	Bce	Управляет доступом на редактирование данных посредством данного элемента управления.
AllowNegative	IntEdit RealEdit	Устанавливает возможность ввода отрицательных значений посредством элемента интерфейса.
AnimateFile	Animate	Название файла с расширением .avi, который будет выводиться на элементе графического интерфейса.
AppendNew	ComboBox	Управляет возможность ручного добавления элементов выпадающего списка (только на независимых списках).
ArrayIndex	ComboBox DateEdit IntEdit Listbox RadioButton RealEdit StringEdit TimeEdit	Если выбранное поле является массивом, то будет отображаться только то поле массива, которое указывается в этом свойстве. Если оставить значение по умолчанию (ноль), то будут отображаться все элементы.
AutoArrange	ListView	Устанавливает, будут ли значки упорядочены автоматически.
AutoDataGroup	Grid	При установке этого свойства поле табличного документа [grid] может содержать только те поля, которые входят в указанную в свойстве DataGroup группу полей таблицы.
AutoDeclaration	Bce	При установке этого свойства будет автоматически объявлена одноименная переменная для доступа к данному элементу из кода X++.
AutoInsSeparator	RealEdit	Похоже, что не дает никакого эффекта.

		Должно позволять отключать автоматическую вставку десятичного разделителя системой.
AutoPlay	Animate	Устанавливает, автоматически ли проигрывать анимационный ролик .avi.
BackgroundColor	Button CheckBox ComboBox CommandButton DateEdit Grid IntEdit Listbox ListView MenuButton MenuItemButton RadioButton RealEdit StaticText StringEdit Table TimeEdit Tree Window	RGB значение или название цветовой схемы Windows, которое задает фоновый цвет элемента управления.
BackStyle	Button CheckBox ComboBox CommandButton DateEdit IntEdit Listbox ListView MenuButton MenuItemButton RadioButton RealEdit StaticText StringEdit TimeEdit Tree Window	Устанавливает стиль фона для элемента графического интерфейса. Позволяет выбрать прозрачный фон для элемента. Это свойство используется для скрытия фона изображений на форме или для установки цвета фона для элемента согласно свойству BackGroundColor.
Bold	Button ComboBox CommandButton DateEdit IntEdit Listbox ListView MenuButton MenuItemButton RadioButton RealEdit StaticText StringEdit TimeEdit	Устанавливает степень «жирности» шрифта, которым выводится текст в элементе графического интерфейса.

	Tree	
Border	Animate Button ComboBox CommandButton DateEdit IntEdit Listbox ListView MenuButton MenuItemButton RealEdit StringEdit TimeEdit Tree	Устанавливает стиль границы элемента графического интерфейса.
BottomMargin	Grid MenuButton RadioButton Table	Устанавливает размер нижней границы элемента.
ButtonDisplay	Button CommandButton MenuButton MenuItemButton	Устанавливает режим отображения кнопки. Могут выводиться текст, изображение, или текст и изображение одновременно, с указанием расположения изображения относительно текста.
CanScroll	ListView Tree	Управляет возможностью прокрутки (scrolling).
Caption	ActiveX HTML RadioButton	Устанавливает метку элемента.
Center	Animate	При установке значения Yes данного свойства анимационный ролик будет расположен по центру элемента графического интерфейса.
ChangeCase	StringEdit	Устанавливает верхний или нижний регистр при вводе значения в элемент графического интерфейса.
CheckBox	ListView Tree	При установке этого свойства в каждой строке данных элемента графического интерфейса будет выводиться флажок.
ClassName	ActiveX HTML	Название класса или идентификатор GUID объекта.
ColorScheme	Button CheckBox ComboBox CommandButton DateEdit Grid	Устанавливает способ выбора цвета для элементов формы (RGB или цветовая схема Windows).

	IntEdit Listbox ListView MenuButton MenuItemButton RadioButton RealEdit StaticText StringEdit Table TimeEdit Tree Window	
Column	Table	Устанавливает активный столбец.
ColumnHeader	ListView	При установке этого свойства столбцы будут отображаться с заголовками (только для случая ViewType = Report).
ColumnHeaderButton	ListView	При установке этого свойства заголовки столбцов будут выполнять функции кнопки (только для случая ViewType = Report).
ColumnImages	ListView	При установке этого свойства рядом с каждым элементом графического интерфейса будет помещено изображение.
Columns	RadioButton Table	Количество колонок в элементе графического интерфейса. Дочерние элементы будут выровнены согласно количеству колонок.
ComboType	ComboBox	Устанавливает тип комбинированного списка.
Command	CommandButton	Устанавливает команду, которая будет выполнена при нажатии на кнопку.
ConfigurationKey	Все	Используется для указания конфигурационного ключа элемента формы.
Custom	ActiveX Html	Используется для открытия редактора свойств ActiveX-компонента.
DataField	CheckBox ComboBox DateEdit IntEdit Listbox RadioButton RealEdit StaticText StringEdit TimeEdit Window	Используется для указания поля из выбранного источника данных, данные которого будут отображаться в элементе формы. Вместо выбора поля, можно указать название display или edit метода в свойстве DataMethod.

DataGroup	Grid	Название группы полей из используемого источника данных.
DataMethod	CheckBox ComboBox DateEdit IntEdit Listbox RadioButton RealEdit StaticText StringEdit TimeEdit Window	Используется для указания название метода, который будет возвращать значение, отображаемое в элементе формы. Если метод принадлежит таблице, источник данных также должен быть указан. При этом метод может принадлежать как таблице, так и непосредственно источнику данных.
DataSource	CheckBox ComboBox DateEdit Grid IntEdit Listbox MenuItemButton RadioButton RealEdit StaticText StringEdit TimeEdit Window	Источник данных, который используется для выборки данных в элемент графического интерфейса.
DateDay	DateEdit	Устанавливает способ отображения дня в дате. Региональные настройки Windows используются по умолчанию.
DateFormat	DateEdit	Устанавливает формат вывода даты. Региональные настройки Windows используются по умолчанию.
DateMonth	DateEdit	Устанавливает способ отображения месяца в дате. Региональные настройки Windows используются по умолчанию.
DateSeparator	DateEdit	Устанавливает разделители при отображении даты. Региональные настройки Windows используются по умолчанию.
DateValue	DateEdit	Используется для указания начального значения элемента графического интерфейса.
DateYear	DateEdit	Устанавливает способ отображения года в дате. Региональные настройки Windows используются по умолчанию.
DecimalSeparator	RealEdit	Устанавливает десятичный разделитель при выводе действительного числа. Региональные



		настройки Windows используются по умолчанию.
DefaultButton	Button CommandButton MenuButton MenuItemButton	Используется для установки кнопки, активной по умолчанию.
Direction	Progress	Устанавливает горизонтальную или вертикальную ориентацию хода процесса.
DisabledImage	Button CommandButton MenuButton MenuItemButton	Путь к файлу изображения, которое будет отображаться на кнопке в неактивном состоянии. Свойство ButtonText при этом должно быть установлено в значение с отображением изображений.
DisabledResource	Button CommandButton MenuButton MenuItemButton	Код ресурса изображения, которое будет отображаться на кнопке в неактивном состоянии. Свойство ButtonText при этом должно быть установлено в значение с отображением изображений.
DisplaceNegative	IntEdit RealEdit	Устанавливает, нужно ли использовать смещение при выводе отрицательных значений в элементах ввода.
DisplayHeight	DateEdit IntEdit RealEdit StaticText StringEdit TimeEdit	Устанавливает максимальное количество строк, которые могут отображаться в одно время в элементе ввода текста.
DisplayLength	ComboBox DateEdit IntEdit Listbox RadioButton RealEdit StaticText StringEdit TimeEdit	Устанавливает максимальное количество символов, которые могут отображаться в одно время в элементе ввода текста.
DragDrop	Bce	Устанавливает режим перетаскивания элемента графического интерфейса.
EditLabels	ListView Tree	Используется для установки права пользователя на редактирование наименования объектов в элементе графического интерфейса.
Enabled	Bce	Устанавливает, разрешен ли элемент графического интерфейса.
EnumType	ComboBox	Укажите перечислимый тип, который

	Listbox RadioButton	будет использоваться для элемента формы в случае, если не выбрано поле или метод источника данных.
ExtendedDataType	ComboBox DateEdit IntEdit Listbox RadioButton RealEdit StringEdit TimeEdit	Укажите расширенный тип данных, который будет использоваться для элемента формы в случае, если не выбрано поле или метод источника данных.
Font	Button ComboBox CommandButton DateEdit IntEdit Listbox ListView MenuButton MenuItemButton RadioButton RealEdit StaticText StringEdit TimeEdit Tree	Шрифт отображения текста в элементе графического интерфейса. Если значение свойства оставить пустым, то будет использоваться шрифт по умолчанию.
FontSize	Button ComboBox CommandButton DateEdit IntEdit Listbox ListView MenuButton MenuItemButton RadioButton RealEdit StaticText StringEdit TimeEdit Tree	Размер шрифта, которым выводится текст в элементе графического интерфейса. Если оставить значение свойства пустым, то будет использоваться размер шрифта по умолчанию.
ForegroundColor	Button CheckBox ComboBox CommandButton DateEdit IntEdit Listbox ListView MenuButton MenuItemButton RadioButton RealEdit StaticText StringEdit	RGB значение или название цветовой схемы Windows, которое задает цвет активного режима элемента графического интерфейса.

	TimeEdit Tree Window	
FormatMST	RealEdit	Форматировать содержимое элемента ввода как сумму в основной валюте компании.
FramePosition	RadioButton	Устанавливает местоположение рамки элемента графического интерфейса.
FrameType	RadioButton	Устанавливает стиль рамки элемента графического интерфейса.
GridLines	Grid ListView Table	Устанавливает, показывать ли линии сетки разметки. Для типа элемента формы ListView – только для случая ViewType = Report.
HasButtons	Tree	Устанавливает необходимость отображения + или – рядом с объектом, который может быть развернут.
HasLines	Tree	Устанавливает необходимость соединения узлов в дереве объектов линиями.
Headerdragdrop	ListView	Устанавливает возможность перетаскивания заголовков для изменения порядка отображения.
Height	Все	Устанавливает высоту элемента графического интерфейса.
HelpText	Все	Текст, который отображается в строке состояния при выборе элемента интерфейса. Если оставить значение свойства пустым, то будет использоваться подсказка, указанная для соответствующего поля источника данных.
HideFirstEntry	ComboBox Listbox RadioButton	Устанавливает необходимость скрывания первой строки списка (как в полях, обязательных к заполнению).
HighlightActive	Grid	Устанавливает необходимость отображения активной строки другим цветом.
Imagemode	Window	Управляет способом отображения графики. См. свойство ImageName.
ImageName	Window	Устанавливает имя файла точечного рисунка, который будет использоваться в качестве заднего фона элемента графического

		интерфейса.
ImageResource	Window	Устанавливает идентификатор ресурса, который будет использоваться в качестве заднего фона элемента графического интерфейса.
Italic	ComboBox CommandButton DateEdit IntEdit Listbox ListView MenuButton MenuItemButton RadioButton RealEdit StaticText StringEdit TimeEdit Tree	Устанавливает необходимость вывода текста графического элемента курсивом.
Item	ComboBox Listbox RadioButton	Используется для указания значения текущего элемента списка (для указания меток элементов, к примеру). Становится неактивным при указании перечислимого типа в свойстве BaseEnum.
ItemAlign	ListView	Похоже, что не дает никакого эффекта. Должно задавать способ выравнивания списка элементов.
Items	ComboBox Listbox RadioButton	Устанавливает количество элементов списка. Становится неактивным при указании перечислимого типа в свойстве BaseEnum.
Label	CheckBox ComboBox DateEdit IntEdit Listbox RealEdit StringEdit TimeEdit Window	Метка, которая будет использоваться в форме для графического элемента. Имеет приоритет по сравнению с меткой расширенного типа данных поля.
LabelAlignment	CheckBox ComboBox DateEdit IntEdit Listbox RealEdit StringEdit TimeEdit Window	Устанавливает выравнивание метки элемента графического интерфейса.
LabelBold	CheckBox	Устанавливает степень «жирности»

	ComboBox DateEdit IntEdit Listbox RealEdit StringEdit TimeEdit Window	шрифта, которым выводится метка графического элемента.
LabelFont	CheckBox ComboBox DateEdit IntEdit Listbox RealEdit StringEdit TimeEdit Window	Устанавливает шрифт отображения текста метки элемента графического интерфейса. Если значение свойства оставить пустым, то будет использоваться шрифт по умолчанию.
LabelFontSize	CheckBox ComboBox DateEdit IntEdit Listbox RealEdit StringEdit TimeEdit Window	Устанавливает размер шрифта, которым выводится метка элемента графического интерфейса. Если оставить значение свойства пустым, то будет использоваться размер шрифта по умолчанию.
LabelForegroundColor	CheckBox ComboBox DateEdit IntEdit Listbox RealEdit StringEdit TimeEdit Window	RGB значение или название цветовой схемы Windows, которое задает цвет текста метки элемента графического интерфейса.
LabelHeight	CheckBox ComboBox DateEdit IntEdit Listbox RealEdit StringEdit TimeEdit Window	Это свойство не работает. Должно устанавливать высоту метки элемента графического интерфейса.
LabelItalic	CheckBox ComboBox DateEdit IntEdit Listbox RealEdit StringEdit TimeEdit Window	При установке этого свойства текст метки графического элемента будет выводиться курсивом.

LabelPosition	CheckBox ComboBox DateEdit IntEdit Listbox RealEdit StringEdit TimeEdit Window	Устанавливает расположение метки элемента графического интерфейса в значение «слева от элемента» или «над элементом».
LabelUnderline	CheckBox ComboBox DateEdit IntEdit Listbox RealEdit StringEdit TimeEdit Window	При установке этого свойства текст метки графического элемента будет выводиться с подчеркиванием.
LabelWidth	CheckBox ComboBox DateEdit IntEdit Listbox RealEdit StringEdit TimeEdit Window	Устанавливает ширину метки элемента графического интерфейса. Используется в случае LabelPosition = Left.
Left	Base	Устанавливает горизонтальную позицию элемента графического интерфейса. В случае если элементы формы имеют горизонтальное выравнивание, и один из них имеет фиксированную позицию слева, то для всех остальных элементов также нужно указать фиксированную позицию.
LeftMargin	Grid MenuButton RadioButton Table	Устанавливает размер левой границы элемента графического интерфейса.
LimitText	DateEdit IntEdit RealEdit StringEdit TimeEdit	Устанавливает максимальное число символов, которое может быть введено в элемент графического интерфейса.
LinesAtRoot	Tree	Устанавливает необходимость соединения объектов дерева с корневым элементом.
LookupButton	DateEdit IntEdit RealEdit StringEdit TimeEdit	Устанавливает необходимость отображения кнопки вызова выпадающего списка рядом с элементом графического интерфейса.

Loops	Animate	Устанавливает, сколько раз проигрывать ролик. Установка нулевого значения соответствует бесконечному циклическому просмотру ролика.
Mandatory	DateEdit IntEdit RealEdit StringEdit TimeEdit	Устанавливает обязательность ввода значения в элемент графического интерфейса.
MenuItemName	MenuItemButton	Устанавливает наименование пункта меню для запуска. Может быть выбран только пункт меню того типа, который указан в свойстве MenuItemType.
MenuItemType	MenuItemButton	Устанавливает тип пункта меню для вызова.
MultiLine	StringEdit	Устанавливает возможность вывода текста графического элемента в виде нескольких строк.
MultiSelect	Button CommandButton Grid MenuButton MenuItemButton	В элементе формы табличный документ [grid] это свойство устанавливает возможность одновременного выбора нескольких записей. Выключение этого свойства в оставшихся типах элементов приводит к запрету их выбора при выделении нескольких строк одновременно.
Name	Все	Устанавливает название элемента графического интерфейса. Это то название, по которому можно будет обращаться к графическому элементу из кода X++ при установке свойства AutoDeclaration в значение Yes.
NeededAccessLevel	Button CommandButton MenuButton MenuItemButton	Устанавливает необходимый уровень доступа для активации этой функции меню.
NoOfDecimals	RealEdit	Устанавливает количество цифр после запятой, которые будут отображаться в элементе графического интерфейса.
NormalImage	Button CommandButton MenuButton MenuItemButton	Путь к файлу изображения, которое будет отображаться на кнопке в активном состоянии. Свойство ButtonText при этом должно быть установлено в значение с отображением изображения.
NormalResource	Button CommandButton	Код ресурса изображения, которое будет отображаться на кнопке в

	MenuButton MenuItemButton	активном состоянии. Свойство ButtonText при этом должно быть установлено в значение с отображением изображения.
OneClickActivate	ListView	Устанавливает возможность активизации элемента по одинарному клику.
PasswordStyle	StringEdit	Устанавливает необходимость замены вводимых символов *, как при вводе пароля.
Pos	Progress	Устанавливает текущую позицию индикатора прогресса.
ProgressType	Progress	Похоже, не имеет никакого эффекта. Должно устанавливать стиль индикатора хода обработки.
RangeHi	Progress	Устанавливает максимальное значение индикатора хода обработки выполнения процесса.
RangeLo	Progress	Устанавливает минимальное значение индикатора хода обработки выполнения процесса.
RealValue	RealEdit	Устанавливает начальное значение элемента графического интерфейса.
ReplaceOnLookup	DateEdit IntEdit RealEdit StringEdit TimeEdit	Устанавливает необходимость замены всего введенного текста результатом выбора из выпадающего списка.
RightMargin	Grid MenuButton RadioButton Table	Устанавливает размер правой границы элемента.
RotateSign	IntEdit RealEdit	Устанавливает необходимость инвертирования знака числа в элементе графического интерфейса.
Row	Table	Устанавливает текущую строку.
Rows	Table	Устанавливает число строк в таблице.
RowSelect	ListView Tree	Устанавливает, выбирается ли вся строка по клику.
SaveRecord	Button CommandButton MenuButton MenuItemButton	Устанавливает необходимость сохранения изменений текущей записи перед активизацией связанной с кнопкой команды.



SearchMode	DateEdit IntEdit RealEdit StringEdit TimeEdit	Устанавливает метод поиска для нахождения записей при вводе данных в элемент интерфейса.
SecurityKey	Bce	Используется для указания ключа контроля доступа на элементе графического интерфейса.
Selection	ComboBox Listbox RadioButton	Устанавливает начальное значение для элемента графического интерфейса.
ShowColLabels	Grid Table	Устанавливает необходимость отображения заголовков столбца.
ShowLabel	CheckBox ComboBox DateEdit IntEdit Listbox RealEdit StringEdit TimeEdit Window	Устанавливает необходимость отображения метки графического элемента.
ShowRowLabels	Grid Table	Устанавливает необходимость отображения заголовков столбцов.
ShowSelAlways	ListView Tree	Устанавливает необходимость сохранения выбора элемента при смене фокуса.
ShowShortCut	Button CommandButton MenuButton MenuItemButton	Устанавливает необходимость резервирования и отображения в элементе графического интерфейса «горячей» клавиши.
ShowZero	IntEdit RealEdit	Устанавливает необходимость индикации нулевых значений.
SignDisplay	IntEdit RealEdit	Устанавливает способ индикации знака минус.
SingleSelection	ListView Tree	Установить возможность одновременного выбора более одного объекта.
Skip	Bce	Устанавливает необходимость пропуска элемента графического интерфейса при табуляции.
Sort	ListView	Устанавливает требуемый порядок сортировки.
Step	Progress	Устанавливает величину приращения показания индикатора за один шаг.

Text	Button ComboBox CommandButton Listbox MenuButton MenuItemButton RadioButton StaticText StringEdit	Устанавливает текст, отображаемый для элемента графического интерфейса.
ThousandSeparator	RealEdit	Устанавливает разделитель для разрядов тысяч при выводе действительного числа. Региональные настройки Windows используются по умолчанию.
TimeFormat	TimeEdit	Устанавливает формат вывода времени. Региональные настройки Windows используются по умолчанию.
TimeHours	TimeEdit	Устанавливает необходимость отображения часов при выводе времени. Региональные настройки Windows используются по умолчанию.
TimeMinute	TimeEdit	Устанавливает необходимость отображения минут при выводе времени. Региональные настройки Windows используются по умолчанию.
TimeSeconds	TimeEdit	Устанавливает необходимость отображения секунд при выводе времени. Региональные настройки Windows используются по умолчанию.
TimeSeparator	TimeEdit	Устанавливает разделитель при выводе времени. Региональные настройки Windows используются по умолчанию.
Top	Все	Устанавливает вертикальную позицию элемента графического интерфейса. При установке фиксированного значения оно рассчитывается от элемента формы, который находится выше данного.
TopMargin	Grid MenuButton RadioButton Table	Устанавливает размер верхней границы элемента графического интерфейса.
TrackSelect	ListView Tree	Устанавливает необходимость выделения объекта, над которым находится курсор, цветом.
Transparent	Animate	Устанавливает необходимость использования прозрачного фона.
TwoClickActivate	ListView	Устанавливает возможность

		активизации элемента двойным кликом.
Underline	Button ComboBox CommandButton DateEdit IntEdit Listbox ListView MenuButton MenuItemButton RadioButton RealEdit StaticText StringEdit TimeEdit Tree	Устанавливает необходимость подчеркивания текста, который выводится в данном элементе графического интерфейса.
Value	CheckBox IntEdit MenuItemButton TimeEdit	Устанавливает начальное значение элемента интерфейса.
VerticalSpacing	Бсе	Устанавливает расстояние сверху и снизу графического элемента.
ViewType	ListView	Устанавливает способ отображения элементов графического интерфейса.
Visible	Бсе	Устанавливает видимость элемента интерфейса. Если следующие графические элементы имеют автоматическое выравнивание, то они будут выровнены с учетом невидимого элемента.
VisibleCols	Grid	Устанавливает число строк, которые выводятся на экран.
VisibleRows	Grid	Устанавливает число колонок, которые выводятся на экран.
Width	Бсе	Устанавливает ширину графического элемента. Если в свойстве указано значение по умолчанию, то ширина будет определяться по расширенному типу данных соответствующего поля источника данных.

## 12.3 Свойства отчетов.

Свойства запроса отчета могут быть найдены в разделе **Свойства запросов**.

## Отчет

Свойство	Описание
Name	Устанавливает название отчета в АОТ.
AllowCheck	При установке этого свойства при запуске будут проверены права доступа пользователя согласно настроенным конфигурационным ключам и ключам контроля доступа.
Autojoin	При установке этого свойства запрос отчета будет автоматически привязан к запросу вызывающего объекта.
Interactive	Устанавливает необходимость открытия диалога для назначения критериев выборки записей, настройки печати при исполнении данного запроса / отчета.

## Дизайн отчета

Свойство	Описание
Name	Устанавливает название дизайна отчета. Используется для идентификации дизайна в случае наличия более одного дизайна.
AutoDeclaration	При установке этого свойства будет автоматически объявлена переменная с именем дизайна для доступа к данному элементу из кода X++.
Caption	Устанавливает текст, который будет отображаться в заголовке окна отчета. При использовании стандартного шаблона отчетов InternalList, заголовок также будет печататься в секции header отчета.
Description	Устанавливает текст, который будет отображаться в заголовке окна отчета. Если это свойство не указано, будет использоваться значение свойства Caption.
JobType	Устанавливает пользовательскую строку для целей идентификации отчета пользователем. Свойство JobType на печать не выводится.
EmptyReportPrompt	Устанавливает подсказку, которая будет отображаться при отсутствии данных для вывода в отчете. Если не заполнять это значение, то будет использовано значение по умолчанию.
ArrangeWhen	Устанавливает момент упорядочивания элементов графического интерфейса дизайна.
ColorScheme	Устанавливает способ выбора цвета для элементов отчета (RGB или цветовая схема Windows).
ForegroundColor	RGB значение или название цветовой схемы Windows, которое задает цвет активного режима элемента графического интерфейса.
ResolutionX	Похоже, не дает никакого эффекта.
ResolutionY	Похоже, не дает никакого эффекта.
Ruler	Устанавливает единицу линейки, которая используется в визуальном редакторе отчета.

ReportTemplate	Устанавливает шаблон, используемый для отчета.
TopMargin	Устанавливает размер верхней границы графического элемента.
BottomMargin	Устанавливает размер нижней границы графического элемента.
LeftMargin	Устанавливает размер левой границы графического элемента.
RightMargin	Устанавливает размер правой границы графического элемента.
Language	Устанавливает язык, на котором будет распечатан отчет. Если оставить значение свойства незаполненным, будет использован язык по умолчанию.
Font	Шрифт отображения текста в элементе графического интерфейса. Если значение свойства оставить пустым, то будет использоваться шрифт по умолчанию.
FontSize	Размер шрифта, которым выводится текст в элементе графического интерфейса. Если оставить значение свойства пустым, то будет использоваться размер шрифта по умолчанию.
Italic	Устанавливает необходимость вывода текста курсивом.
Underline	Устанавливает необходимость вывода текста с подчеркиванием.
Bold	Устанавливает «жирность» текста в элементе графического интерфейса.
PrintFormName	Устанавливает наименование формы, используемой для выбора параметров печати. При вызове отчета с помощью прикладных классов runbase значение этого свойства игнорируется.
HideBorder	Устанавливает, нужно ли показывать рамку вокруг полей и разделов.
Orientation	Устанавливает жесткую ориентацию страницы (портретную или альбомную). По умолчанию, дизайн будет распечатан в альбомной ориентации, если для размещения всех элементов в отчете с использованием портретной ориентации без масштабирования недостаточно места.
FitToPage	Устанавливает необходимость масштабирования страницы в случае, если не все элементы помещаются в отчете в ширину.
RemoveRepeatedHeaders	Устанавливает, нужно ли записывать заголовок, если за ним следует более детальный заголовок.
RemoveRepeatedFooters	Устанавливает, нужно ли записывать нижний колонтитул, если тот же самый итог только что был записан.
RemoveRedundantFooters	Устанавливает, нужно ли записывать нижний колонтитул, если только одно значение внесло вклад в сумму.

## Авто дизайн

Свойство	Описание
GrandHeader	Устанавливает необходимость печати заголовка группы. Текст этого заголовка группы будет заголовком самого верхнего уровня, и будет печататься до заголовка, который можно включить на секции body.
GrandTotal	Устанавливает необходимость печати секции нижнего колонтитула. Будет выводиться на печать секция нижнего колонтитула для элементов интерфейса, у которых установлено свойство SumAll, SumPos или SumNeg.
HeaderText	Устанавливает текст, который выводится в заголовке отчета в случае, если свойство GrandHeader установлено в значение Yes. Если оставить значение свойства пустым, будет использовано значение по умолчанию.
TotalText	При установке значения свойства GrandTotal в Yes, текст, введенный в свойство TotalText, будет печататься в нижнем колонтитуле отчета вместо текста по умолчанию.

## Элементы секций отчета

В этом разделе приведен обзор различных типов элементов секций отчета.

Свойство	Секции	Описание
ArrangeMethod	Все	Устанавливает способ упорядочивания элементов графического интерфейса в данном контейнере.
ArrangeWhen	Все	Устанавливает момент упорядочивания элементов графического интерфейса в данном контейнере.
AutoDeclaration	ProgrammableSection Body PageFooter	При установке этого свойства будет автоматически объявлена одноименная переменная для доступа к данному элементу из кода X++.
Bold	Все	Устанавливает степень «жирности» шрифта, которым выводится текст.
Bottom	ProgrammableSection Prolog Body Epilog	Устанавливает вертикальную позицию элемента графического интерфейса. При установке фиксированного значения оно рассчитывается от нижней границы страницы.
BottomMargin	Все	Устанавливает размер нижней границы секции.
ColorScheme	Все	Устанавливает способ выбора цвета для элементов секции (RGB или цветовая схема Windows).
ColumnHeadingsStrategy	ProgrammableSection	Устанавливает метод отображения

	Body PageFooter	заголовков колонок. При установке значения DisplacedLines, заголовки колонок будут распечатаны зигзагообразно в две строки, если в одну заголовок не поместился.
Columns	Все	Это свойство не работает. На формах устанавливает количество столбцов для дочерних элементов.
ColumnSpace	Все	Устанавливает расстояние между соседними столбцами.
ControlNumber	ProgrammableSection	Устанавливает целочисленный код программируемой секции. Используется для выполнения секции из кода X++.
Font	Все	Шрифт отображения текста в секции. Если значение свойства оставить пустым, то будет использоваться шрифт по умолчанию.
FontSize	Все	Размер шрифта, которым выводится текст в элементах секции. Если оставить значение свойства пустым, то будет использоваться размер шрифта по умолчанию.
FooterText	Body	Свойство доступно только для авто-дизайнов. Значение этого свойства будет печататься в секции нижнего колонтитула отчета вместо текста по умолчанию.
ForegroundColor	Все	RGB значение или название цветовой схемы Windows, которое задает цвет активного режима элемента графического интерфейса.
GrandHeader	Body	Свойство доступно только для авто-дизайнов. Устанавливает необходимость печати заголовка группы.
GrandTotal	Body	Свойство доступно только для авто-дизайнов. Устанавливает необходимость печати секции нижнего колонтитула. Будет выводиться на печать секция нижнего колонтитула для элементов интерфейса, у которых установлено свойство SumAll, SumPos или SumNeg.
HeaderText	Body	Свойство доступно только для авто-дизайнов. Устанавливает текст, который выводится в заголовке отчета в случае, если свойство GrandHeader установлено в значение Yes. Если

		оставить значение свойства пустым, будет использовано значение по умолчанию.
Height	Bce	Устанавливает высоту секции.
Italic	Bce	Устанавливает необходимость вывода текста заголовков и данных секции курсивом.
LabelBottomMargin	ProgrammableSection Body PageFooter	Устанавливает величину границы под заголовками, перед данными секции.
LabelTopMargin	ProgrammableSection Body PageFooter	Устанавливает величину границы над заголовками секции.
LeftMargin	Bce	Устанавливает размер левой границы секции.
LineAbove	Bce	Вставляет строку сверху от секции.
LineBelow	Bce	Вставляет строку снизу от секции.
LineLeft	Bce	Вставляет строку слева от секции.
LineRight	Bce	Вставляет строку справа от секции.
Name	Bce	Устанавливает название секции. Это то название, по которому можно будет обращаться к секции из кода X++ при установке свойства AutoDeclaration в значение Yes.
NoOfHeadingLines	ProgrammableSection Body PageFooter	Устанавливает количество строк заголовка. Для того, чтобы предотвратить печать заголовка, установите значение этого свойства в 0.
ResolutionX	Bce	Не дает никакого эффекта.
ResolutionY	Bce	Не дает никакого эффекта.
RightMargin	Bce	Устанавливает размер правой границы секции.
Ruler	Bce	Устанавливает единицу линейки, которая используется в визуальном редакторе отчета для секции.
Table	Body	Устанавливает таблицу, используемую как источник данных. Используется при выполнении element.send() для определения секций, подлежащих печати.



Thickness	Все	Устанавливает толщину рамки секции отчета.
Top	ProgrammableSection Prolog Body Epilog	Устанавливает вертикальную позицию секции. При установке фиксированного значения оно рассчитывается от верхней границы страницы.
TopMargin	Все	Устанавливает размер верхней границы секции.
Underline	Все	Устанавливает необходимость подчеркивания текста заголовков и данных секции.

## Шаблон секции

Шаблоны секции используются только в отчетах с авто-дизайном.

Свойство	Описание
SectionTemplate	Устанавливает название шаблона секции, используемого в секции.
Table	Устанавливает таблицу, используемую как источник данных в шаблоне секции. Таблица должна содержать mapping в карте соответствия, указанной для шаблона секции.

## Группа секций

Группы секций используются только в отчетах со сгенерированным дизайном.

Свойство	Описание
DataField	Используется для идентификации группы секций в случае наличия двух или более групп секций для одной и той же таблицы.
Name	Устанавливает название группы секций.
Table	Устанавливает таблицу, используемую как источник данных. Используется при выполнении element.send() для определения момента печати группы секций.

## Элементы секций отчета

В этом разделе будут рассмотрены свойства всех видов элементов, которые можно поместить в секцию.

Свойство	Тип	Описание
Alignment	String Real Integer	Устанавливает способ выравнивания информации, отображаемой в элементе графического интерфейса.

	Enum Date Time Sum Bitmap Prompt	Используется для левого выравнивания данных при вертикальном способе упорядочивания элементов секции.
AllowNegative	Real Integer Sum	Свойство не имеет смысла в отчетах. Устанавливает возможность ввода отрицательных значений посредством элемента интерфейса на формах.
ArrayIndex	String Real Integer Enum Date Time Sum Bitmap Prompt	Если выбранное поле является массивом, то будет распечатано только то поле массива, которое указывается в этом свойстве. Если оставить значение по умолчанию (ноль), то будут распечатаны все элементы.
AutoDeclaration	Bce	При установке этого свойства будет автоматически объявлена одноименная переменная для доступа к данному элементу из кода X++.
AutoInsSeparator	Real Sum	Используется для указания MorphX о необходимости вставки десятичного разделителя.
BackgroundColor	String Text Real Integer Enum Date Time Sum Prompt	RGB значение или название цветовой схемы Windows, которое задает фоновый цвет элемента интерфейса.
BackStyle	String Text Real Integer Enum Date Time Sum Bitmap Prompt	Устанавливает стиль фона для элемента графического интерфейса. Позволяет выбрать прозрачный фон для элемента. Это свойство используется для скрытия фона изображений на форме или для установки цвета фона для элемента согласно свойству BackGroundColor.
Bold	String Text Real Integer Enum Date Time	Устанавливает степень «жирности» шрифта, которым выводится текст в элементе графического интерфейса.

	Sum Prompt	
BottomMargin	Bce	Устанавливает размер нижней границы элемента интерфейса.
ChangeCase	String Text Enum Prompt	Устанавливает верхний или нижний регистр при выводе значения элемента графического интерфейса.
ChangeLabelCase	String Real Integer Enum Date Time Sum Bitmap Shape	Устанавливает верхний или нижний регистр при выводе значения метки элемента графического интерфейса.
ColorScheme	Bce	Устанавливает способ выбора цвета для элементов формы (RGB или цветовая схема Windows).
ConfigurationKey	Bce	Используется для указания конфигурационного ключа элемента секции.
CssClass	Bce	Класс CSS, отвечающий за генерацию элемента HTML (свойство Web).
DataField	String Real Integer Enum Date Time Sum Bitmap Prompt	Используется для указания поля из выбранного источника данных, данные которого будут отображаться в элементе отчета. Вместо выбора поля, можно указать название display метода в свойстве DataMethod.
DataFieldName	Sum	Устанавливает название поля, по которому необходимо проводить суммирование.
DataMethod	String Real Integer Enum Date Time Bitmap	Используется для указания название метода, который будет возвращать значение, отображаемое в элементе формы. Если метод принадлежит таблице, источник данных также должен быть указан. При этом метод может принадлежать как таблице, так и непосредственно источнику данных.
DateDay	Date	Устанавливает способ отображения дня в дате. Региональные настройки Windows используются по умолчанию.

DateFormat	Date	Устанавливает формат вывода даты. Региональные настройки Windows используются по умолчанию.
DateMonth	Date	Устанавливает способ отображения месяца в дате. Региональные настройки Windows используются по умолчанию.
DateSeparator	Date	Устанавливает разделители при отображении даты. Региональные настройки Windows используются по умолчанию.
DateYear	Date	Устанавливает способ отображения года в дате. Региональные настройки Windows используются по умолчанию.
DecimalSeparator	Real Sum	Устанавливает десятичный разделитель при выводе действительного числа. Региональные настройки Windows используются по умолчанию.
DisplaceNegative	Real Integer Sum	Устанавливает необходимость использования смещения при выводе отрицательных значений.
DynamicHeight	String	Устанавливает необходимость автоматического изменения размера элемента графического интерфейса для отображения всего текста, выводимого в нем.
ExtendedDataType	String Text Real Integer Enum Date Time Prompt	Укажите расширенный тип данных, который будет использоваться для элемента формы в случае, если не выбрано поле или метод источника данных.
ExtraSumWidth	Real Integer Sum	Устанавливает необходимость выделения дополнительного места для вывода суммы. Полезно при выводе сумм с большим количеством цифр в числе в указанной валюте.
Font	String Text Real Integer Enum Date Time Sum Prompt	Шрифт отображения текста в элементе графического интерфейса. Если значение свойства оставить пустым, то будет использоваться шрифт по умолчанию.
FontSize	String	Размер шрифта, которым выводится

	Text Real Integer Enum Date Time Sum Prompt	текст в элементе графического интерфейса. Если оставить значение свойства пустым, то будет использоваться размер шрифта по умолчанию.
ForegroundColor	Bce	RGB значение или название цветовой схемы Windows, которое задает цвет активного режима элемента графического интерфейса.
FormatMST	Real Sum	Форматирует содержимое элемента графического интерфейса как сумму в основной валюте компании.
Height	Bce	Устанавливает высоту элемента графического интерфейса.
ImageName	Bitmap	Устанавливает имя файла точечного рисунка для отображения (bitmap).
ImageResource	Bitmap	Устанавливает идентификатор ресурса для отображения. Перечень идентификаторов с соответствующими изображениями можно увидеть, распечатав отчет <i>Tutorial_Resources</i> .
Italic	String Text Real Integer Enum Date Time Sum Prompt	Устанавливает необходимость вывода текста графического элемента курсивом.
Label	Bce	Метка, которая будет использоваться в отчете для графического элемента. Выводится вместо метки расширенного типа данных поля отчета или если расширенный тип не указан. Метка не будет распечатана, если свойство ShowLabel установлено в значение No.
LabelBold	String Real Integer Enum Date Time Sum Bitmap Shape	Устанавливает степень «жирности» шрифта, которым выводится метка графического элемента.

LabelCssClass	Bce	Свойство касается только Web. Класс CSS, отвечающий за генерацию метода в HTML.
LabelFont	String Real Integer Enum Date Time Sum Bitmap Shape	Устанавливает шрифт отображения текста метки элемента графического интерфейса. Если значение свойства оставить пустым, то будет использоваться шрифт по умолчанию.
LabelFontSize	String Real Integer Enum Date Time Sum Bitmap Shape	Устанавливает размер шрифта, которым выводится метка элемента графического интерфейса. Если оставить значение свойства пустым, то будет использоваться размер шрифта по умолчанию.
LabelItalic	String Real Integer Enum Date Time Sum Bitmap Shape	При установке этого свойства текст метки графического элемента будет выводиться курсивом.
LabelLineBelow	String Real Integer Enum Date Time Sum Bitmap Shape	Устанавливает необходимость печати строки такой же длины, как и у элемента графического интерфейса, ниже метки.
LabelLineThickness	String Real Integer Enum Date Time Sum Bitmap Shape	Устанавливает толщину строки под меткой графического элемента.
LabelPosition	String Real Integer Enum Date	Устанавливает расположение метки элемента графического интерфейса в значение «слева от элемента» или «над элементом».

	Time Sum Bitmap Shape	
LabelTabLeader	String Real Integer Enum Date Time Sum Bitmap Shape	Используется при установке значения свойства LabelPosition в «слева от элемента». Устанавливает необходимость вывода символов табуляции или многоточия, завершающихся двоеточием, справа от метки элемента графического интерфейса.
LabelUnderline	String Real Integer Enum Date Time Sum Bitmap Shape	При установке этого свойства текст метки графического элемента будет выводиться с подчеркиванием. Свойство LabelLineBelow используется для вывода подчеркивания по всей длине метки.
LabelWidth	String Real Integer Enum Date Time Sum Bitmap Shape	Устанавливает ширину метки элемента графического интерфейса. Используется в случае LabelPosition = Left.
Left	Все	Устанавливает горизонтальную позицию элемента графического интерфейса. В случае если элементы формы имеют горизонтальное выравнивание, и один из них имеет фиксированную позицию слева, то для всех остальных элементов также нужно указать фиксированную позицию.
LeftMargin	Все	Устанавливает размер левой границы элемента графического интерфейса.
Line	Shape	Устанавливает тип строки фигуры элемента графического интерфейса.
LineAbove	Все	Вставляет строку сверху от элемента графического интерфейса.
LineBelow	Все	Вставляет строку снизу от элемента графического интерфейса.
LineLeft	Все	Вставляет строку слева от элемента графического интерфейса.

LineRight	Bce	Вставляет строку справа от элемента графического интерфейса.
MenuItemName	Bce	Свойство Web. Наименование пункта меню для запуска.
MenuItemType	Bce	Свойство Web. Тип пункта меню для запуска.
ModelFieldName	Bce	Устанавливает название элемента графического интерфейса, в соответствии с которым необходимо позиционировать данный графический элемент. Свойство часто используется для позиционирования элементов типа prompt или sum в том же столбце, что и выводимые данные.
Name	Bce	Название графического элемента.
NoOfDecimals	Real Sum	Устанавливает количество цифр после запятой, которые будут отображаться.
ResizeBitmap	Bitmap	Устанавливает необходимость масштабирования графического изображения, если для элемента графического интерфейса указаны фиксированные ширина и высота.
RightMargin	Bce	Устанавливает размер правой границы элемента.
RotateSign	Real Integer Sum	Устанавливает необходимость инвертирования знака числа в элементе графического интерфейса.
SecurityKey	Bce	Используется для указания ключа контроля доступа на элементе графического интерфейса.
ShowLabel	String Real Integer Enum Date Time Sum Bitmap Shape	Устанавливает необходимость отображения метки графического элемента.
ShowPicAsText	Bitmap	Устанавливает необходимость отображения пути к файлу или идентификатору ресурса вместо графического изображения в предварительном просмотре (по клику мыши загружается изображение).
ShowZero	Real	Устанавливает необходимость



	Integer Sum	индикации нулевых значений.
SignDisplay	Real Integer Sum	Устанавливает способ индикации знака минус.
SumAll	Real Integer	Устанавливает необходимость суммирование всех значений. Используется в элементах типа sum для определения тех графических элементов, по которым производится суммирование. Если это свойство установлено в авто-дизайне, то значения элемента будут просуммированы в том случае, если пользователь добавит вывод итога по этому полю.
SumNeg	Real Integer	Установить необходимость суммирования только отрицательных значений.
SumPos	Real Integer	Установить необходимость суммирования только положительных значений.
SumType	Sum	Устанавливает тип суммирования: все числа, положительные или отрицательные.
Table	String Real Integer Enum Date Time Sum Bitmap Prompt	Устанавливает источник данных, который используется в элементе графического интерфейса.
Text	Text	Устанавливает текст, который будет выводиться на печать.
Thickness	Bce	Устанавливает толщину рамки элемента графического интерфейса.
ThousandSeparator	Real Sum	Устанавливает разделитель для разрядов тысяч при выводе действительного числа. Региональные настройки Windows используются по умолчанию.
TimeFormat	Time	Устанавливает формат вывода времени. Региональные настройки Windows используются по умолчанию.
TimeHours	Time	Устанавливает необходимость отображения часов при выводе

		времени. Региональные настройки Windows используются по умолчанию.
TimeMinute	Time	Устанавливает необходимость отображения минут при выводе времени. Региональные настройки Windows используются по умолчанию.
TimeSeconds	Time	Устанавливает необходимость отображения секунд при выводе времени. Региональные настройки Windows используются по умолчанию.
TimeSeparator	Time	Устанавливает разделитель при выводе времени. Региональные настройки Windows используются по умолчанию.
Top	Все	Устанавливает вертикальную позицию элемента графического интерфейса. При установке фиксированного значения оно рассчитывается от элемента формы, который находится выше данного.
TopMargin	Все	Устанавливает размер верхней границы элемента графического интерфейса.
Type	Shape	Устанавливает тип фигуры элемента графического интерфейса.
TypeHeaderPrompt	Text Prompt	Устанавливает необходимость вывода символов табуляции или многоточия, завершающихся двоеточием, справа от элемента графического интерфейса.
Underline	String Text Real Integer Enum Date Time Sum Prompt	Устанавливает необходимость подчеркивания текста, который выводится в данном элементе графического интерфейса.
Visible	Все	Устанавливает видимость элемента интерфейса. Если следующие графические элементы имеют автоматическое выравнивание, то они будут выровнены с учетом невидимого элемента.
WarnIfMissing	Bitmap	Устанавливает необходимость вывода предупреждения в Infolog при отсутствии выводимого изображения, указанного в ImageName, ImageResource или DataMethod.

WebTarget	Все	Свойство Web. Устанавливает, какой Web Target использовать в качестве функции меню.
Width	Все	Устанавливает ширину графического элемента. Если в свойстве указано значение по умолчанию, то ширина будет определяться по расширенному типу данных соответствующего поля источника данных.

## Группа полей

Свойство	Описание
AutoFieldGroupOrder	Устанавливает, сохранять ли свойства и порядок сортировки полей. Группа полей будет автоматически обновлена при внесении изменений в группу полей таблицы словаря данных.
DataGroup	Устанавливает название группы полей таблицы.
Table	Устанавливает таблицу, группа полей которой будет выводиться в отчете.

## 12.4 Свойства запроса

### Запрос

Свойство	Описание
AllowCheck	Определяет, будут ли проверять конфигурационные ключи и ключи контроля доступа при запуске отчета.
Form	Устанавливает наименование формы запроса для интерфейса с пользователем. Это свойство обычно не меняется.
Interactive	Устанавливает необходимость открытия диалога для назначения критериев выборки записей, настройки сортировки данных при исполнении данного запроса.
Literals	Устанавливает формат представления символов в SQL-запросах. Возможные значения: default, forcелiterals или forceplaceholders. Все часто исполняемые запросы следует выводить с параметром forceplaceholders, а forcелiterals можно использовать при редко используемых ассиметричных данных.
Name	Устанавливает название запроса в AOT.
Title	Устанавливает заголовок данного запроса.
UserUpdate	Устанавливает возможность сохранения пользовательских настроек запроса.

## Источники данных

Свойство	Описание
AllowAdd	Устанавливает возможность включения полей источника в условия фильтрации запроса. При отключении этого свойства пользователь не будет иметь возможности добавлять поля. То же касается и полей сортировки.
Company	Устанавливает, данные которой компании следует использовать. Это свойство редко используется, так как пользователь обычно ожидает получить данные из текущей компании.
Enabled	Устанавливает, должен ли использоваться данный источник данных. При отключении источника он и все вложенные источники будут исключены из sql запроса.
FetchMode	Это свойство присутствует только у вложенных источников данных. Иногда используется для управления способом связывания источников данных («один ко многим» или «один к одному»), но обычно достаточно использования свойства JoinMode.
FirstFast	Устанавливает необходимость считывания первой записи быстрее последующих.
FirstOnly	Устанавливает необходимость считывания только первой записи.
JoinMode	Устанавливает стратегию формирования набора записей из нескольких «сцепленных» источников данных.
Name	Устанавливает название источника данных в АОТ.
OrderMode	Устанавливает необходимость сортировки или группировки записей источника данных.
Relations	При включении данного свойства отношение между источником данных и источником на один уровень выше будет установлено в соответствии со связями, указанными в базе данных.
Update	При включении данного свойства записи, считываемые из источника данных, будут выбираться на обновление (forupdate), что позволит редактировать данные источника.

## Поля

Свойство	Описание
Dynamic	При установке значения Yes устанавливает список полей в соответствии с определенным списком полей таблицы. В случае группировки данных отчета, свойство автоматически устанавливается в значение No, и в узел <i>Fields</i> могут быть добавлены только групповые функции (агрегаты).

## Поля сортировки

Свойство	Описание
AutoHeader	Устанавливает необходимость печати заголовка группы при изменении значения в этом поле. Используется только запросами отчетов.
AutoSum	Устанавливает необходимость печати промежуточного итога при изменении значения в этом поле. Используется только запросами отчетов.
HeaderDetailLevel	Используется в запросах отчетов при установке свойства AutoHeader. По умолчанию заголовок группы печатается при изменении значения в поле. Данное свойство позволяет устанавливать, при изменении какой части поля должен печататься заголовок группы. Поля, в значении которых используются символы точки, пробела, дефиса, обратной косой, состоят из нескольких под-полей. К примеру, если значение кода клиента равно "CUST100.10", то значение поля состоит из двух под-полей. При установке 2 в качестве значения свойства HeaderDetailLevel заголовок группы будет печататься при изменении значения после точки.
Ordering	Устанавливает порядок сортировки записей по данному полю по возрастанию или убыванию.
SumDetailLevel	Используется в запросах отчетов при установке свойства AutoSum. По умолчанию промежуточный итог печатается при изменении значения в поле. Данное свойство позволяет устанавливать, при изменении какой части поля должен печататься промежуточный итог. Поля, в значении которых используются символы точки, пробела, дефиса, обратной косой, состоят из нескольких под-полей. К примеру, если значение кода клиента равно "CUST100.10", то значение поля состоит из двух под-полей. При установке 2 в качестве значения свойства HeaderDetailLevel промежуточный итог будет печататься при изменении значения после точки.

## Критерии выборки

Property	Description
Enabled	При установке значения No данное поле будет игнорироваться.
Label	Устанавливает метку для критерия выборки.
Name	Устанавливает название диапазона для критерия в АОТ.
Status	Устанавливает статус критерия. Доступные статусы: Открыто, скрыто или закрыто. Открытые критерии могут быть изменены или удалены. Закрытые статусы не могут меняться или быть удалены. Скрытые критерии вовсе не отображаются в диалоге настройки запроса.
Value	Укажите фиксированное значение критерия выборки.

## 12.5 Свойства Меню

Свойство	Описание
----------	----------

ChangedBy	Пользователь, который последний вносил изменения в Меню.
ChangedDate	Дата последнего изменения Меню.
ChangedTime	Время последнего изменения Меню.
ConfigurationKey	Используется для указания конфигурационного ключа на меню.
CreatedBy	Имя пользователя, создавшего Меню.
CreatedTime	Время создания Меню.
CreationDate	Дата создания Меню.
HelpText	Устанавливает краткую подсказку, отображаемую в статусной строке.
Label	Устанавливает метку Меню.
LockedBy	Имя пользователя, который заблокировал меню.
Name	Устанавливает наименование Меню.
NeededAccessLevel	Устанавливает уровень доступа, необходимый для активации Меню.
SecurityKey	Используется для указания ключа контроля доступа для Меню.
SetCompany	Устанавливает, будет ли изменяться компания при активации формы.

## 12.6 Свойства пунктов меню

Свойство	Описание
ChangedBy	Имя пользователя, который редактировал пункт меню последним.
ChangedDate	Дата последнего изменения пункта меню.
ChangedTime	Время последнего изменения пункта меню.
Class	Устанавливает тип объекта, который будет запущен при активации пункта меню.
ConfigurationKey	Используется для указания конфигурационного ключа на пункте меню
CountryConfigurationKey	Используется для указания конфигурационного ключа страны на пункте меню.
CreatedBy	Имя пользователя, создавшего пункт меню.
CreatedTime	Время создания пункта меню.
CreationDate	Дата создания пункта меню.
EnumParameter	Предоставляет выбор элемента перечислимого типа, передаваемого в вызываемый объект в качестве параметра. Связано со свойством EnumTypeParameter.

EnumTypeParameter	Предоставляет выбор перечислимого типа, передаваемого в вызываемый объект в качестве параметра. Метод <code>args.parmEnum()</code> возвращает значение этого свойства. Объект, который будет запущен, указывается с помощью свойств <code>Class</code> и <code>Object</code> .
HelpText	Устанавливает краткую подсказку, отображаемую в статусной строке.
Label	Метка на элементе меню или кнопке.
LockedBy	Имя пользователя, который заблокировал пункт меню.
MultiSelect	Устанавливает возможность запуска данного пункта меню при выборе нескольких записей в форме.
Name	Устанавливает наименование пункта меню.
NeededAccessLevel	Устанавливает уровень доступа, необходимый для активации данного пункта меню.
Object	Устанавливает наименование объекта, который будет запущен при активации данного пункта меню.
Parameters	Устанавливает строку параметров, передаваемых вызываемому объекту. Метод <code>args.parm()</code> возвращает значение данного свойства в вызываемом объекте. Объект, который будет запущен, указывается с помощью свойств <code>Class</code> и <code>Object</code> .
RunOn	Устанавливает место исполнения вызываемого объекта.
SecurityKey	Используется для указания ключа контроля доступа для пункта меню.
Web	Используется для классов веб-приложений.
WebConfigurationKey	Соответствующий пункту меню конфигурационный ключ Веб.
WebPage	Свойство Web. Веб-страница, привязанная к данному пункту меню. В версии 3.0 не используется.
WebSecureTransaction	Требуется ли данная функция шифрования по стандарту SSL?





## 13 Приложение. Средства разработки MorphX

Средства разработки MorphX – это набор инструментов, созданных с помощью MorphX, который находится в меню **Сервис | Средства разработки**. Все эти инструменты предоставляют доступ к информации об объектах приложения. Большая часть из них может быть вызвана из АОТ или из редактора кода по клику правой кнопки мыши. Также из этого меню можно вызвать и Отладчик кода, но это не имеет большого смысла, поэтому информацию об отладчике ищите в разделе **Введение в MorphX**. Для просмотра средств разработки из АОТ воспользуйтесь меню Средства разработки [DevelopmentTools] в АОТ.

### 13.1 Перекрестные ссылки

Система перекрестных ссылок позволяет отслеживать использование в приложении объектов, переменных и меток. Если Вы собрались изменить название какого-либо объекта или переменной, или хотите удалить одну из меток из меточного файла, инструментарий перекрестных ссылок поможет оценить последствия таких изменений.

Прежде чем начать пользоваться перекрестными ссылками, необходимо выполнить полную перекомпиляцию объектов системы с включенным обновлением перекрестных ссылок. Обновление перекрестных ссылок может проводиться во время компиляции, включив соответствующий пункт в настройке компилятора. Однако это значительно увеличит время компиляции объектов. Одноразовое обновление перекрестных ссылок можно выполнить из меню **Сервис | Средства разработки | Перекрестные ссылки | Периодические операции | Обновление**. Для построения перекрестных ссылок по всем объектам выберите *Обновить все*. Обновление перекрестных ссылок является очень ресурсоемкой операцией и требует несколько часов на выполнение. Такое обновление действительно очень трудоемко, поскольку в процессе обновления будут проиндексированы все объекты АОТ. Для получения максимальной выгоды от использования перекрестных ссылок необходимо периодически выполнять их обновление. Этого можно добиться настройкой пакетного задания на обновление перекрестных ссылок в нерабочие часы суток.

Перекрестные ссылки можно просмотреть как из верхнего меню приложения, так и из АОТ: для этого необходимо выбрать объект, сделать по нему правый клик мыши и выбрать Перекрестные ссылки в контекстном меню Add-ins. Все перекрестные ссылки хранятся в таблицах, названия которых начинаются с xRef.

---

**Примечание:** После установки стандартной версии приложения Ахарт, следует построить все перекрестные ссылки, после чего выгрузить таблицы с этой информацией. В следующий раз, когда будет производиться установка нового приложения, достаточно будет импортировать эти таблицы и не придется повторно ожидать построения перекрестных ссылок.

---

Формы *Объекты по именам* и *Путь к объекту*, которые можно вызвать из верхнего меню перекрестных ссылок, отображают полный список перекрестных ссылок системы. Форма *Объекты по именам* выводит этот список по типу

объекта, а форма Путь к объекту показывает путь в АОТ к каждому из выводимых объектов. Если Вам необходимо отфильтровать только определенные строки перекрестных ссылок или же Вы ищите конкретный объект АОТ, то указанные формы могут оказаться полезными. В случае же, когда известен путь к объекту в АОТ, намного проще перейти к нему в дереве репозитория и выбрать Перекрестные ссылки из контекстного меню Add-ins. При вызове с объекта Репозитория, форма Путь к объекту будет отображать только АОТ пути к объектам, в которых используется выбранный объект. Из указанной формы можно получить дополнительную информацию: например, список объектов, используемых выбранным, список объектов, использующих выбранный объект. По нажатию кнопки Правка можно просмотреть код X++ тех методов, в которых используется ссылка на выбранный объект. В случае, если выбранный объект является свойством, панель его свойств можно открыть, нажав на кнопку Утилиты и выбрав Свойства.

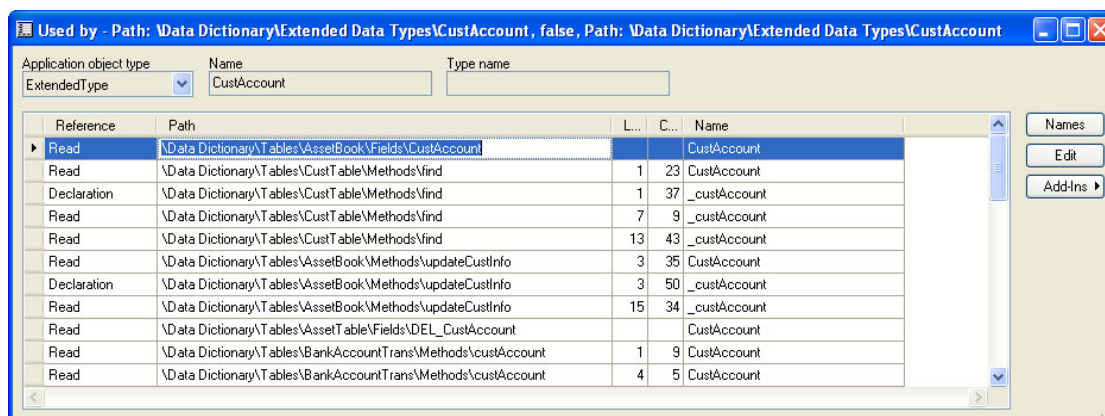


Рисунок 48: Перекрестные ссылки с информацией об использовании EDT CustAccount

## 13.2 Объекты приложения

Подменю Объекты приложения находится в узле *Menus* в АОТ под названием *ApplicationObjects*. Инструменты этого меню сложно назвать жизненно необходимыми и вряд ли Вы будете их часто использовать, однако это те «мелочи», которые приятно иметь под рукой. Так как меню находится в АОТ, то в него можно добавлять свои пункты меню.

### Формы объектов приложения

Формы *Объекты приложения* и *Объекты старых приложений* отображает информацию из двух системных таблиц *UtilElements* и *UtilElementsOld*, которые содержат информацию практически обо всех объектах АОТ, а также обо всех объектах старых слоев. Форму объектов приложения можно вызвать из подменю Add-ins в АОТ, при этом данные в ней будут отфильтрованы по выделенному узлу Репозитория прикладных объектов.

При использовании формы Объекты старых приложений придется запастись терпением, так как она открывается очень продолжительное время. В ней объединяются источники данных UtilElements и UtillementsOld для отображения отличий между объектами. Эта функциональность могла бы очень пригодиться при сравнении обновленной версии приложения с предыдущей версией, но форма настолько медленно работает, что ее использование теряет всякий смысл. Более детальную информацию о таблицах Util\* можно найти в главе **Словарь Данных [Data Dictionary]**.

## Администрирование объектов

С помощью этого инструмента можно назначить людей, ответственных за отдельные объекты приложения. Форма Администрирование объектов предоставляет обзор ответственных за объекты с указанием команды, к которой принадлежит ответственное лицо. В связанных формах можно настроить ответственных разработчиков и группы сотрудников. Этот инструмент, скорее всего, был создан для использования командой разработчиков Ахapta, поэтому объекты имеют довольно простую структуру и, возможно, не полностью оттестированы. Стоит хотя бы взглянуть на форму Администрирование объектов. Список типов объектов приложения расположен вторым по порядку, тогда как заполнять его необходимо в первую очередь, до выбора имени объекта приложения. Однако можно отредактировать соответствующие объекты под свои нужды. Названия всех объектов начинаются с SysUtilMangement\*.

## Использование данных

Форма Использование данных отображает содержимое системной таблицы SysLastValue в разрезе всех пользователей. Более детальная информация об использовании данных находится в разделе **Настройки пользователя**.

## Мониторинг объектов

Форма используется для сбора статистики о количестве объектов, используемых в каждом из модулей системы. В расчет попадают только модули стандартного приложения. Для того, чтобы получить возможность собирать статистику по Вашему собственному модулю, необходимо модифицировать класс SysUtilCount. Результаты сбора статистики накапливаются в таблице SysCountTable.

## Блокированные объекты

При использовании функциональности блокирования объектов в АОТ данная форма может оказаться очень полезной. Форма показывает перечень заблокированных объектов с указанием пользователя, заблокировавшего объект. При удалении записи из данной формы будет снята блокировка с соответствующего объекта. Однако в АОТ объект будет по-прежнему отображаться как заблокированный, так как изменения, сделанные в форме

Блокированные объекты, отразятся на АОТ только после перезапуска клиентского приложения Ахартa.


## Инструменты обновления

Три пункта меню, Обновление словаря, Обновление данных и Обновление AOD, используются при разработке web интерфейса. Используются они для обновления объектов и данных для того, чтобы в web интерфейсе отразились последние изменения.

## Реиндексация

Функция реиндексации перестроит индексный файл слоя прикладных объектов. Никогда не следует проводить реиндексацию слоев прикладных объектов при наличии активных пользователей в приложении, так как это приведет к повреждению файла индекса. При необходимости проведения реиндексации более предпочтительным является удаление файла индекса АХАРD.АОI. Файл может быть удален только при отсутствии активных пользователей системы и выключенных серверах AOS и соединений COM connector. При запуске клиентского приложения индексный файл будет автоматически построен заново.

## 13.3 Системный монитор

Форма системного мониторинга может быть вызвана как из верхнего меню, так и по двойному щелчку на иконке компьютера  в статусной строке справа. Форма отображает информацию об обращениях к базе данных с указанием размеров каждого обращения. При использовании сервера AOS в форме будет показана дополнительная информация об обмене данными между сервером и клиентом, а также дополнительная закладка, в которой будет представлена информация о латентности соединения. Вся эта информация может очень пригодиться при оптимизации трафика между клиентом, сервером и базой данных.

## Отслеживание обращений к базе данных

Для запуска последовательности сбора статистики нажмите кнопку *Продолжение* и запустите объекты исследования, к примеру, откройте форму заказов и создайте накладную. Затем нажмите кнопку *Пауза* для окончания отслеживания запросов к БД. При этом будет подсчитано общее количество различных типов обращений к базе данных. Очень часто управление количеством операций *select* является способом значительной оптимизации быстродействия системы.

## Отслеживание обращений к серверу приложений

При использовании Сервера приложений Ахартa (AOS), объекты могут исполняться как на клиенте, так и на сервере. При сборе статистики будут

подсчитан трафик к серверу и от сервера. Информацию, полученную после сбора статистики системным монитором, можно использовать для уменьшения количества обращений к клиенту. Интерактивные объекты, такие как формы или диалоги, всегда будут выполняться на стороне клиента. Однако является более предпочтительным исполнение на сервере кода в классах и обращений к БД. Для детального описания того, как установить необходимость выполнения класса на стороне сервера приложений, читайте в главе **Классы [Classes]**.

На закладке *Удаленное соединение* можно проверить задержку соединения с сервером приложений. Для получения правильных значений задержки тест следует выполнить несколько раз. Если Ваше приложение будет устанавливаться у клиентов с ограниченной пропускной способностью линии, то с этой закладки можно промоделировать работу пользователя через удаленное соединение. Установить производительность системы и задержку, после чего нажмите кнопку *Установить как текущее*. Этот механизм позволяет протестировать производительность системы в различных условиях связи, и даст необходимые данные для оптимизации Вашего кода.

## 13.4 Профайлер кода

Форма *Профайлер кода* используется для подсчета времени выполнения кода и времени обработки запросов к базе данных. Как и инструменты для мониторинга системы, форма профайлера кода активируется нажатием кнопки *Начало*. После окончания работы функциональности, время выполнения которой нужно исследовать, необходимо нажать кнопку *Стоп*. Опция глубины запроса используется для указания глубины запроса, до которой Вы хотите исследовать работу Вашего кода. По завершении сбора данных профайлером необходимо ввести название, под которым будут сохранены собранные данные. Каждая сессия работы профайлера сохраняется в базе данных. Для просмотра этой информации необходимо нажать кнопку *Сессии профайлера*. Форма *Сессии профайлера* показывает все завершенные на сегодня сессии. Вы сможете полностью просмотреть все строки кода, которые выполнялись в данной сессии профайлера, и даже отредактировать каждую строку. Для более удобного просмотра строк воспользуйтесь деревом кода. Показывается время выполнения каждой строки кода, а также, если установлена соответствующая опция, итоговое время выполнения повторных вызовов строки кода. Итоговое время выполнения особенно полезно при оценке кода, который выполняется в цикле.

```
static void Intro_CodeProfiler(Args _args)
{
    CustTable    custTable;
    Counter      counter;
;

    #profileBegin("Test of profiler")
    while select custTable
    {
        info(strFmt("Customer name: %1", custTable.name));
    }
}
```

```
counter++;  
  
if (counter >= 10)  
    break;  
}  
#profileEnd  
}
```

Другой способ активации профайлера кода – непосредственно из кода с использованием макросов. Точки начала и окончания сессии профайлера необходимо установить в коде так, как показано в примере выше. Установка выполнения сессий профайлера непосредственно из кода полезно в том случае, когда Вам необходимо исследовать только отдельную часть кода. При запуске же профайлера из меню Вы затронете больше кода, чем необходимо. Обратите внимание, что итоги придется рассчитывать вручную из формы Сессии профайлера после завершения сессии.

При использовании сервера приложений также будет рассчитываться и время AOS. На закладке Обзор формы сессий профайлера будет помимо всего прочего указан тип клиента, которым был запущен профайлер. Попробуйте запустить указанный пример как в двух, так и в трехуровневой конфигурации. Обратите внимание, что при запуске примера клиентом AOS генерируется меньше строк профайлера. Причина в том, что код выполняется на сервере приложений и меньше обращается к клиенту.

Использование профайлера кода – это хорошая практика программирования, так как Вы видите время выполнения каждой строки кода и в процессе этого обучаетесь поиску узких мест кода, которые следует оптимизировать. Учтите, что рассчитывается не общее время, а только время выполнения кода. Профайлер кода генерирует огромное количество строк даже для простых участков кода, к примеру, выполнения нескольких заданий из формы. После запуска профайлера система может немного подтормаживать, а если вы запустите профайлер для анализа громоздкого пакетного задания, то обработка может вообще не завершиться. При необходимости анализа кода пакетного задания будет, возможно, более рациональным ограничить количество прогонов задания, или использовать макрокоманды для запуска и останова профайлера кода.

## 13.5 Иерархия объектов

Перед использованием *Иерархии объектов*, необходимо обновить иерархию типов в перекрестных ссылках. Сделать это можно из меню **Сервис | Средства разработки | Перекрестные ссылки | Периодические операции | Обновление**, отметив только *Обновить иерархию типов*.

Правилом хорошего тона программирования в Ахарт является использование расширенных типов данных вместо базовых типов системы. За время работы Вы, скорее всего, будете создавать большое количество расширенных типов, так как они содержат информацию о метках, форматировании текста и отношениях с

таблицами. Вот здесь и пригодится построение иерархии типов, так как она предоставит обзор расширенных типов с детализацией наследования типов. После создания новой таблицы Вы встречаетесь с необходимостью выбора: создавать новый расширенный тип или постараться найти уже существующий расширенный тип данных, который подходит для Вашей задачи – при этом очень удобным будет просмотр иерархии существующих типов данных. Названия базовых типов данных не полностью соответствуют названиям, используемым в АОТ. Поначалу это может сбить с толку, однако очень поможет понимание того, что базовый тип *varstring* соответствует базовому типу *memo* в АОТ.

Таблицы и классы также присутствуют в Иерархии объектов. Все таблицы принадлежат верхнему узлу *Common*. Все таблицы находятся на одном уровне, так как таблицы не могут наследоваться одна от другой. Список таблиц можно использовать для получения информации о таблицах, такой как перечень методов, полей и индексов таблицы. В списке присутствуют как прикладные, так и системные таблицы. Индексы и методы системных таблиц также видны в описании, чего Вы не найдете в АОТ. В узле *Object* Вы найдете перечень классов с указанием наследования. По каждому классу доступна информация о методах и уровне, на котором находится каждый метод в иерархии. Поэтому очень просто определить, к какому именно классу принадлежит тот или иной метод.

## 13.6 Визуальное моделирование с MorphXplorer

Используйте *Визуальное моделирование с MorphXplorer* для визуализации модели данных системы Ахарта, вычерчивая диаграммы отношений между элементами и иерархии классов (Entity Relation Diagram). Построение диаграмм происходит непосредственно на первой закладке формы. Указать название для Вашей диаграммы можно на закладке Разное. Закладка Цвета используется для изменения цветов по умолчанию, используемых при отображении диаграмм. Построенные диаграммы можно сохранить в файл или вывести на печать. Хорошей деталью при печати больших диаграмм является возможность указания используемого количества страниц в длину и в ширину.

Для того чтобы добавить новый объект в диаграмму, перейдите на закладку отображения диаграммы и выберите Новый Класс или Таблица из контекстного меню. При выборе таблицы или класса на экран будет выведен перечень объектов, из которого Вы сможете выбрать нужный. Перетащите прямоугольный образ объекта на закладку Диаграмма и кликните для позиционирования объекта в выбранной области диаграммы. Контекстное меню объекта содержит перечень отношений выбранного объекта с другими объектами системы. Обратите внимание, что в диаграмме нельзя комбинировать таблицы и классы. Форму Визуальное моделирование с MorphXplorer возможно использовать для построения диаграммы с отображением или отношений между таблицами, или иерархии классов. Если список связанных объектов пуст, то это означает, что необходимо построить перекрестные ссылки модели данных системы. Сделать это можно из меню **Сервис | Средства разработки | Перекрестные ссылки | Периодические операции | Обновить**, выбрав *Обновить модель данных*.

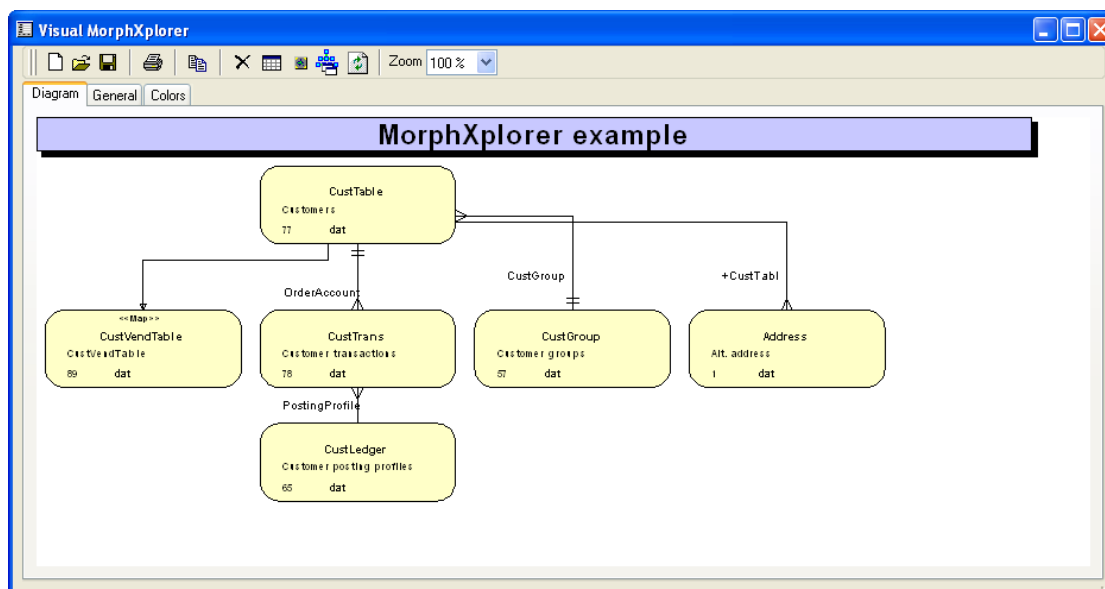


Рисунок 49: Визуальное моделирование с MorphXplorer

Связанные таблицы разделены на 2 категории: связь 1-n и n-1. Таким же способом связанные таблицы представлены в запросах Ахарта. Если таблица является частью карты соответствий, то Вы также будете иметь возможность выбрать в списке саму карту соответствия. Для ознакомления с символами, которые используются в диаграмме, обратитесь к **Рисунку 50: Символы, используемые в Визуальном моделировании с MorphXplorer**. При добавлении связанной таблицы будет также отображаться поле, по которому связаны выбранные таблицы. Если отношение состоит из нескольких полей, то поля связи не будут отображаться рядом со связью. Вместо этого будет выводиться название таблицы с префиксом в виде +. Эта особенность является ограничением Визуального моделирования с MorphXplorer, так как было бы предпочтительно получить весь список полей отношения, хотя бы во всплывающей подсказке при наведении мыши на отношение между таблицами.

При построении диаграммы иерархии классов для выбранного класса можно добавить классы-предки и классы-наследники. Если у Вас построены перекрестные ссылки, то помимо этого вы сможете добавить классы, которые используются или которые используют выбранный класс.

Символ	Объект	Описание
	Таблица	Указывает на отношение «один ко многим».
	Таблица	Связь «один к одному».
	Таблица Класс	Для таблиц указывает, что связанный объект – карта соответствия. Для классов указывает на класс-предок и класс-наследник.
	Класс	Классы, которые используются или которые используют данный класс.



**Рисунок 50: Символы, используемые в Визуальном моделировании с MorphXplorer**

Визуальное моделирование с MorphXplorer можно вызвать из контекстного меню Add-ins в АОТ. Пункт меню для вызова формы Визуальное моделирование с MorphXplorer будет доступен только в том случае, если выбран ровно один класс или одна таблица. Если вы загрузите проект MORPHXIT\_VisualMorphXplorer, то сможете строить диаграмму со всеми отношениями в Визуальном моделировании, выбрав сразу несколько таблиц в АОТ. Обратите также внимание на то, что если отношение между таблицами построено на нескольких полях, как, к примеру, между таблицами CustTable и CustTrans, то будут отображены оба отношения. Эта модификация поможет Вам получить быстрый обзор модели данных системы. С помощью этой модификации Вы сможете посмотреть подмодуль системы, однако у нее также есть ограничения. Хотя Вы и можете выбирать несколько таблиц одновременно, однако если Вы выберете слишком много таблиц, то Ваше клиентское приложения прервет выполнение с ошибкой.

Для отображения диаграмм форма Визуальное моделирование с MorphXplorer использует компонент *VarChart*. Помимо этого, данный компонент используется и в других формах системы, однако он плохо документирован, то есть об этом компоненте отсутствует достаточное количество информации. Однако форма Визуального моделирования с MorphXplorer – отличное место для того, чтобы познакомиться с возможностями компонента *VarChart*.

## 13.7 Анализатор кода

*Анализатор кода* позволяет просматривать содержимое АОТ в виде HTML документа. Можно спуститься по дереву вплоть до конечного узла и получить информацию о слоях, свойствах объекта, а также просмотреть код методов выбранного объекта. Если Вы обновили перекрестные ссылки, то информация о связях объекта также будет доступна. Для отображения информации Анализатор кода использует справочную систему. При желании взглянуть на пример кода работы со справочной системой из X++ можно обратиться к классам с префиксом *SysCodeExplorer\** в названии.

## 13.8 Описания таблиц

Пункт меню Описания таблиц показывает содержимое таблицы *UtilElements*. Отчет, который выдается по нажатию на пункт меню, организован таким образом, что отображает наиболее важную информацию о таблицах, такую как перечень полей, свойства таблицы и отношения. При печати отчета лучше указать диапазон названий таблиц, данные о которых необходимо получить, иначе будет напечатан отчет по всем таблицам, что составляет более 2000 страниц печатного текста и надолго оставит Ваш принтер без отдыха. Этот отчет стоит использовать только в том случае, когда Вам необходимо получить информацию о нескольких таблицах или подмодуле системы в печатном виде.

## 13.9 Количество записей в таблицах

Форма *Количество записей в таблицах* подсчитывает количество записей во всех таблицах текущей компании. В форме также перечислены временные таблицы и карты соответствия, но суммирование количества записей проводится только по обычным таблицам. При настройке кэширования на таблицах эта форма может сослужить хорошую службу. К примеру, Вы могли установить для таблицы кэширование всей таблицы [entire table], но в том случае, если таблица содержит большое количество записей, Вы сможете об этом узнать и изменить способ кэширования данных этой таблицы. Более детальную информацию о кэшировании можно найти в разделе **Приложение. Свойства объектов**.

## 13.10 Тексты справки

Тексты справки представлены в виде HTML документа. При нажатии клавиши F1 на форме или узле АОТ откроется справка по выбранному элементу. Содержимое справочной системы можно просмотреть из формы Тексты справки.

Существует три различных типа интерактивной справки: System Documentation, Application Developer Documentation и Application Documentation (Система, Разработка и Приложение). Файлы интерактивной справки в АОТ находятся в самом конце списка. Системная документация является источником информации об объектах ядра, таких как системные классы и таблицы. Узел документации прикладной разработки используется для создания интерактивной справки для созданных Вами классов и таблиц. При создании новой таблицы или класса в этот узел будет автоматически добавлена информация о созданном объекте.

Документация приложения содержит тексты интерактивной справки, представляемые пользователю при работе с системой. Редактирование текстов любой из трех справок можно выполнить как из формы Тексты справки, так и при помощи вышеуказанных узлов в Репозитории прикладных объектов.

---

**Примечание:** В данный момент очень небольшое количество таблиц и классов имеют описание в интерактивной справке. Говорят, что этот момент должен быть доработан в версии системы 4.0.

---

## 13.11 Переход к новой версии

По мере выхода новых версий системы или пакетов обновления к существующей версии Вам предстоит обновлять Ваше приложение. Технология слоев в системе Ахарта облегчает этот процесс, так как Ваши модификации находятся на слое отличном от базового кода, который поставляется компанией Microsoft. Тем не менее, огромное количество строк кода придется выверять вручную. К примеру, если Вы изменили форму на прикладном слое VAR и произошли изменения в той же форме на слое SYP нового пакета обновления, то Вам вручную придется учесть изменения в форме. Для этого можно воспользоваться утилитой сравнения

объектов, которая описана в разделе **Сравнение объектов**. Но перед тем, как сравнивать измененный объект этой утилитой, можно получить обзор изменений в новой версии объекта с помощью инструментов пункта меню Переход к новой версии.

## Переименованные прикладные объекты

Все объекты, которые были переименованы в новой версии или пакете обновления, перечислены в форме *Переименованные прикладные объекты*. Построение строк формы необходимо вызвать вручную. Для построения списка переименованных объектов необходимо запустить приложение на обновление и нажать кнопку Обновление на форме. По нажатию кнопки будет создан текстовый файл, содержащий список переименованных объектов. После этого текстовый файл может быть импортирован в новую версию приложения нажатием кнопки Обновление.

Обзор объектов с измененными названиями полезен, так как сравнивать придется только те объекты, у которых названия остались прежними. К примеру, если вы модифицировали объект слоя SYS, который в пакете обновления был переименован, то после загрузки пакета обновления останется только объект на Вашем слое. Список элементов с измененными именами поможет Вам вести учет изменений при переименовании или удалении объекта.

## Создание проекта обновления приложения

При обновлении приложения первым шагом является создание проекта обновления. При этом будет создан проект в АОТ, содержащий все объекты, в которых есть изменения в текущем слое. В диалоге подтверждения создания проекта можно отметить опцию удаления устаревших изменений в текущем слое, которые теперь является частью более низкого уровня. Если Вы импортировали hot fix в текущий слой, то следует включить эту опцию, так как изменения в hot fix обычно включаются в выпускаемый пакет обновления или новую версию.

После создания проекта обновления приложения Вы сможете продолжить сравнение объектов с помощью инструмента сравнения.

## Сравнение слоев

Для полного сравнения изменений двух различных слоев приложения можно воспользоваться инструментом *Сравнение слоев*. В результате будет создан проект в АОТ содержащий объекты, которые существуют только на выбранном слое и те объекты, в которых существуют отличия от ссылочного слоя.

Если Вы проводили обновления до бета-версии приложения или устанавливали пакет обновления до официального выпуска, то Вы сможете с помощью утилиты сравнения слоев увидеть изменения относительно новой версии.

## 13.12 Мастера

*Мастера* в меню Мастера верхнего меню помогут Вам при создании модификаций, подготавливая базу объектов, таких как отчеты и классы. Использование мастера в начале может оказаться неплохим упражнением, так как Вы узнаете, какие базовые свойства и методы должны существовать у объекта, какие свойства обычно устанавливаются и какие графические элементы добавляются в отчет. Мастер отчетов особенно полезен, так как он поможет Вам узнать, каким образом используются определенные свойства отчета, и Вам не придется тратить время в поисках нужных ответов, перебирая узлы AOT.

Если Вы раскроете узел меню в AOT, то там сможете отыскать меню Wizards. В это меню можно добавлять собственные мастера.

### Мастер отчетов

Для знакомства с генератором отчетов воспользуйтесь пунктом меню Мастер отчетов. С помощью этого мастера Вы сможете построить основную часть отчета. Для простых отчетов Вам, возможно, даже не придется после этого вносить изменения в AOT. Полный обзор последовательности шагов для создания отчета с помощью мастера отчетов находится в **Приложение. Мастер отчетов**.

### Мастер мастеров

До версии системы Ахарта 2.5 правилом хорошего тона считалось обеспечение пользователей приложения возможностью использования мастеров для создания записей в сложных формах системы. Начиная с версии 3.0, это правило изменили, так как были внедрены шаблоны записей. Форма Мастер мастеров была создана специально для ускорения разработки таких мастеров.

Несмотря на то, что использование мастеров для добавления данных уже не является правилом хорошего тона, могут существовать специфические случаи, когда предпочтительным является именно мастер, особенно в случаях, когда пользователю необходимо будет добавить данные в редко используемую форму.

### Мастер меточных файлов

Это единственный из мастеров, который используется не для создания объектов Репозитария. С помощью этого мастера создаются новые меточные файлы. Для того, чтобы создать меточный файл с помощью мастера меточных файлов, следует войти в двухуровневой среде, к которой не подключены другие пользователи системы. После завершения работы мастера необходимо перезапустить клиентское приложение до того, как какие-либо другие пользователи будут подключаться к системе. После выполнения указанных действий меточный файл будет готов к использованию. Обратите внимание на то, что меточный файл будет обновлен в первый раз только после того, как из

системы выйдут все пользователи. Именно поэтому рекомендуется создавать меточные файлы, когда к системе Ахартa не подключены другие пользователи.

Более детальную информацию о меточной системе можно найти в разделе **Метка**.

## Мастер создания классов

Мастер создания классов довольно простой. С помощью этого мастера можно создать класс в АОТ и методы из интерфейсных классов. При создании доступен только один шаблон класса. Однако Вы можете расширить возможности мастера, добавляя свои собственные шаблоны, иначе же, мастер создания классов больше подходит для целей обучения.

## Мастер оболочек для СОМ-объектов

Если Вам необходимо написать интерфейс для ActiveX компонента, использующего СОМ интерфейс, то начать следует именно с этого мастера. На первом шаге данный мастер представит список всех установленных на вашем компьютере библиотек. Выберите библиотеку, и система Ахартa создаст классы обертки для всех классов и методов выбранной библиотеки. Это действительно отличный мастер, который очень сильно ускоряет создание интерфейса СОМ.

## 13.13 Метка

Система меток является одной из мощных особенностей системы Ахартa, которая позволяет облегчить работу приложения с несколькими языками. Вместо ввода текста для поля или подсказки статусной строки непосредственно в коде, туда добавляется код метки. Код метки выбирается из меточной системы, в которой содержится информация о соответствующей метке на каждом из выбранных языков.

Код метки имеет следующий синтаксис: @<код меточного файла><номер метки>, к примеру, @SYS1002. Код меточного файла состоит из 3 символов. Номер метки – число из последовательности, которое увеличивается каждый раз при добавлении метки. В папке приложения Ахартa существует набор меточных файлов для каждого из установленных языков. Меточные файлы называются AX<код меточного файла><код языка>.<расширение>, к примеру, AXSYSEN-US.ALD. Обзор различных расширений меточных файлов представлен на рисунке 51: Меточные файлы.

Расширение файла	Описание
*.ALD	Текстовый файл, содержащий метки указанного языка.
*.ALC	Комментарии, добавляемые к метке в системе меток, записываются в данный файл.
*.ALI	Индексный файл для меточного файла. Если указанный индексный файл не найден, то он автоматически будет создан в момент первого

	обращения к системе меток.
--	----------------------------

**Рисунок 51: Меточные файлы**

Значения меток в меточных файлах обновляются после выхода последнего из пользователей из системы. Если, по какой-то причине, последний из подключенных пользователей завершит соединение с системой неверно, то изменений в меточных файлах не произойдет. Так как меточные файлы являются частью Ваших модификаций совместно с модификациями в АОТ, очень важно быть уверенным в том, что меточные файлы обновились всеми добавленными значениями меток. Перейдите к редактированию файла \*.ALD для проверки того, что последние из созданных меток были успешно добавлены. Если метка в файле отсутствует, то вместо текста метки пользователи системы будут видеть код соответствующей метки.

---

**Примечание:** Причиной для использования меток является обеспечение возможности для пользователей запускать приложение на их языке предпочтения. Даже если Вам не требуется переводить Ваши модификации на другой язык, все равно следует подумать об использовании меток, так как использование меток обеспечит согласующиеся наименования объектов в разрезе всей системы. Если Вам понадобится изменить какой-то из терминов, достаточно будет изменить только соответствующую метку вместо того, чтобы искать все строки, где используется данный термин.

---

В стандартном приложении существует по одному меточному файлу для каждого из прикладных слоев, которые называются в соответствии с названием слоя. Не рекомендуется изменять эти меточные файлы, так как они могут обновляться с каждым новым пакетом обновления или версией системы. Вместо этого следует создать новый меточный файл специально для ваших модификаций.

## Поиск меток

Эта форма используется для поиска меток. Эта же форма открывается при поиске меток из окна свойств объекта и кода X++.

Поиск меток производится в том языке, который выбран в верхней части формы. Если Вы хотите совершить поиск метки на английском языке, но хотите помимо этого видеть значение этой метки и на других языках, то для этого необходимо включить соответствующие языки на закладке Расширенно. Поиск все равно будет выполнен на английском языке, но в придачу в нижней части формы будут перечислены значения метки на выбранных языках. Это поможет Вам убедиться, что метки были правильно добавлены на всех языках, используемых в формах, таких как накладная на заказ. При поиске метки с использованием символов < и > будут наложены фильтры, и результат будет получен быстрее. Если Вам необходимо найти метку для текста 'Customer', поиск будет выполнен значительно быстрее, если ввести в поле поиска значение <Customer>, так как будут выбраны только те метки, которые содержат только слово 'Customer'. Для того, чтобы найти все метки, которые начинаются со слова 'Customer', необходимо ввести значение <Customer.

Обычно меточная система вызывается из окна свойств объекта или из редактора кода, в которых Вы производите поиск по выделенному тексту для замены его соответствующим кодом метки. Если соответствующая метка найдена, необходимо нажать кнопку *Вставка метки* для вставки кода метки вместо текста. Если же метка не была найдена, то Вы можете добавить новую метку с соответствующим значением, нажав кнопку *Создать*. Новая метка будет создана в меточном файле по умолчанию, который указан на закладке *Расширенно*.

### Журнал изменений меток

Все изменения в метках по сравнению со стандартным приложением отображаются в этой форме. В ней запоминаются все созданные, измененные и удаленные метки. Если Вы случайно удалите нужную метку, то ее можно воссоздать из формы Журнал изменений меток.

### Мастер меточных файлов

Описание мастера меточных файлов было приведено выше в разделе **Мастера**.

### Меточные интервалы

Форма Меточные интервалы используется для администрирования кодов меток в Ваших меточных файлах. При работе в среде, где модификации выполняются более чем на одном приложении, и при этом используется один и тот же меточный файл, форму Меточные интервалы следует использовать для задания интервалов меточных файлов для каждого из приложений. Введите код меточного файла. Статус интервала установите в значение «Доступный идентификационный номер метки». В каждом из приложений необходимо затем указать интервал кодов меток и установить последний номер метки интервала. При следующем создании метки, код метки будет выбран с учетом указанного интервала, и значение последнего номера метки будет увеличено соответственно.

Предпочтительным решением, конечно же, будет создание всех используемых меток в одном приложении, но приведенное решение можно использовать как обходной путь. Затем, конечно же, придется слить все меточные файлы \*.ALD в один файл вручную.





## 14 Приложение. Мастер отчетов

В этом приложении приведена пошаговая инструкция создания отчетов с помощью Мастера отчетов. Мастер отчетов – это инструмент, разработанный для написания простых отчетов в Ахартa людьми, не имеющими технических знаний разработки. С другой стороны, этот инструмент будет полезен и тем, кто только учиться создавать отчеты в Ахартa.

Результат работы мастера можно сохранить как новый отчет в АОТ, или же просто выполнить созданный отчет после завершения работы мастера. При использовании мастера отчетов, отчет будет сохранен в АОТ с такой же структурой, как и при использовании генератора отчетов. Поэтому, до тех пор, когда Вы освоитесь с созданием отчетов из АОТ, Мастер отчетов может оказаться очень полезным, так как Вам необходимо всего лишь следовать пошаговой инструкции для указания необходимой структуры отчета.

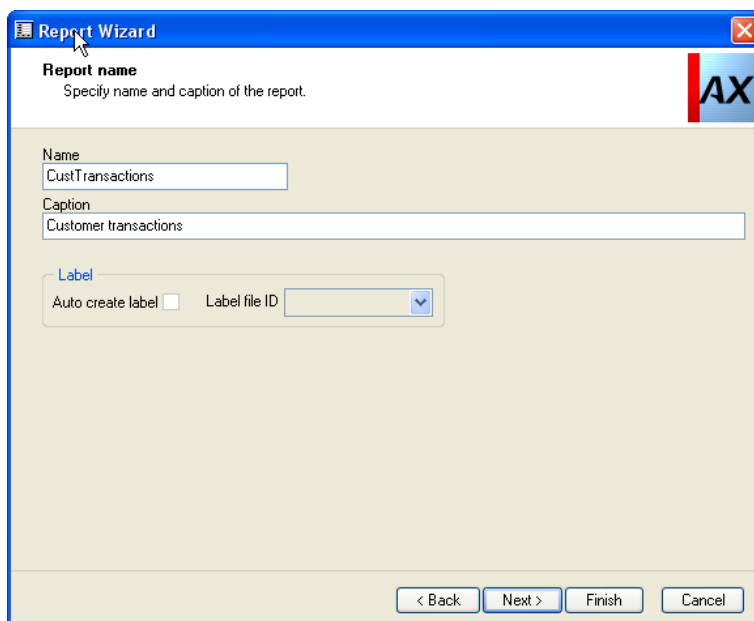
### Шаг 1

Укажите, хотите ли вы выбирать таблицы по системным именам или их меткам. Под системными именами подразумеваются названия таблиц в АОТ. Предпочтительным является использование системных имен, так как они уникальны в разрезе приложения, тогда как метки таблиц могут повторяться.

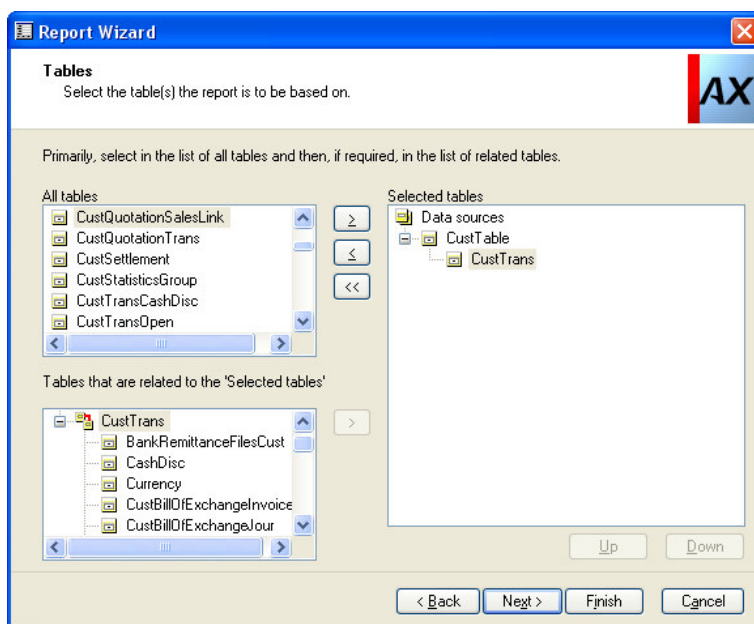


**Шаг 2**

Укажите название отчета. Это имя будет использовано при сохранении отчета в АОТ. Заголовок отчета будет использован при печати.

**Шаг 3**

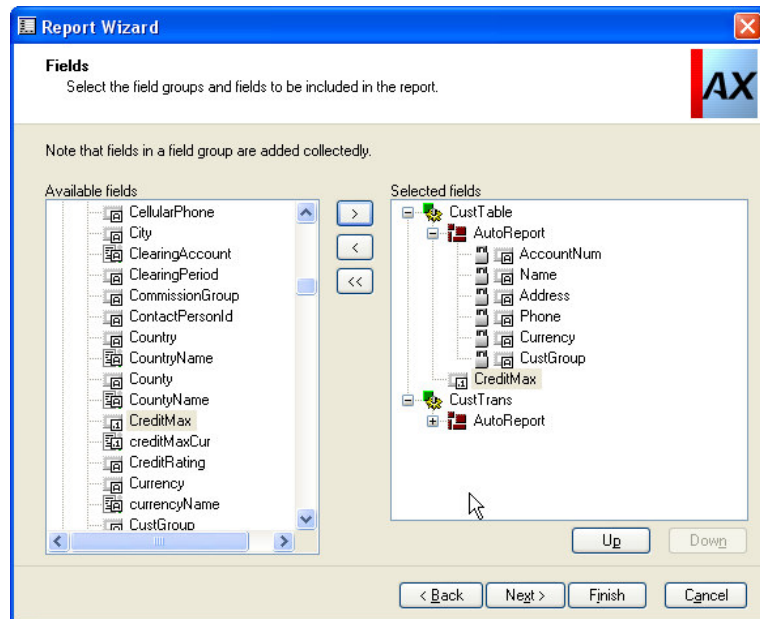
В окно «Выбранные таблицы» добавьте те таблицы, из которых будут выбираться данные. К примеру, выберите таблицу CustTable в качестве таблицы первого уровня, а затем CustTrans как связанную таблицу. При отсутствии перечня в окне таблиц, связанных с CustTable, необходимо обновить перекрестные ссылки модели данных. Это действие следует выполнять при установке



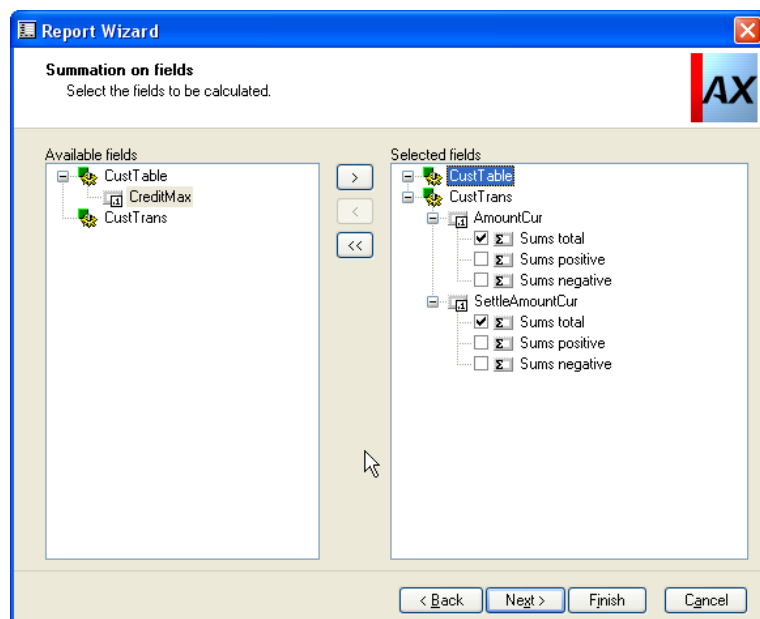
Ахарта, или каждый раз при внесении изменений в модель данных системы. Для обновления перекрестных ссылок выполните пункт меню **Сервис | Средства разработки | Перекрестные ссылки | Периодические операции | Обновление**. В диалоге настроек обновления выберите один пункт «Обновить модель данных».

**Шаг 4**

Укажите поля, которые будут выведены в отчете для каждого из выбранных источников данных. По умолчанию, группа полей *AutoReport* будет выбрана для каждого из источников. Доступен выбор из групп полей, отдельных полей источника и display-методов. Кнопки *Вверх* и *Вниз* служат для задания порядка печати полей в отчете.

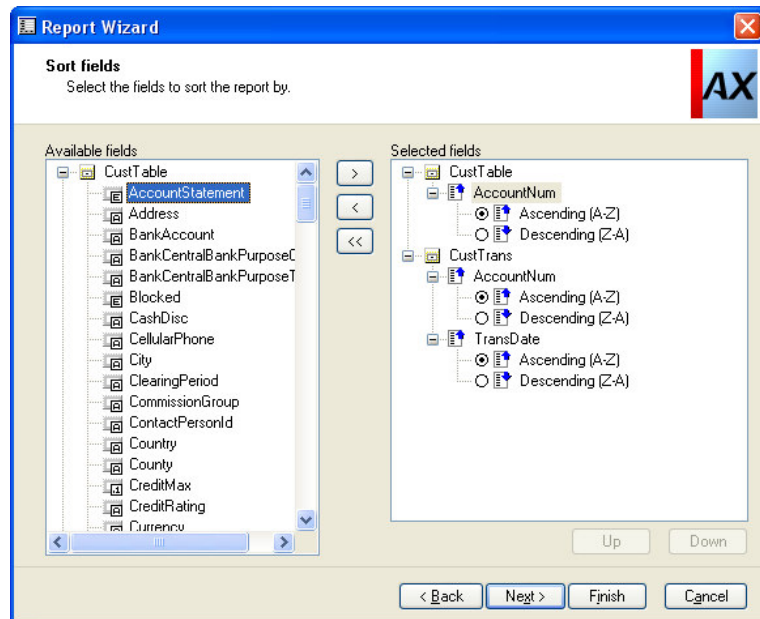
**Шаг 5**

Выберите поля целочисленного и действительного типов, по которым необходимо рассчитывать итоговые суммы. По умолчанию будут выбраны все доступные для расчета итогов поля.

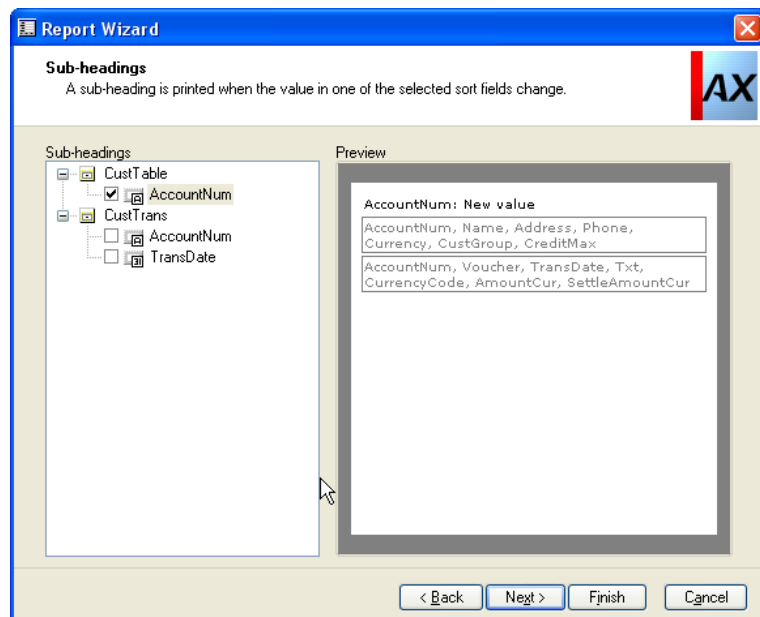


**Шаг 6**

Выберите поля, по которым будет осуществляться сортировка. По умолчанию будут добавлены все поля первого индекса каждого из источников данных. Кнопки *Вверх* и *Вниз* используются для указания порядка сортировки полей.

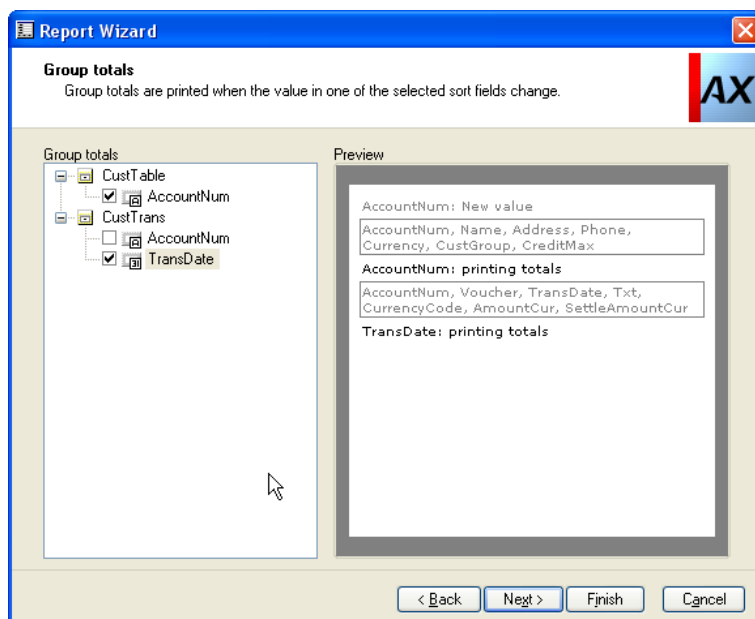
**Шаг 7**

Укажите, будут ли печататься подзаголовки при изменении значения одного из выбранных полей сортировки. В приведенном примере подзаголовков печатается каждый раз, когда изменяется номера счета в таблице клиентов.

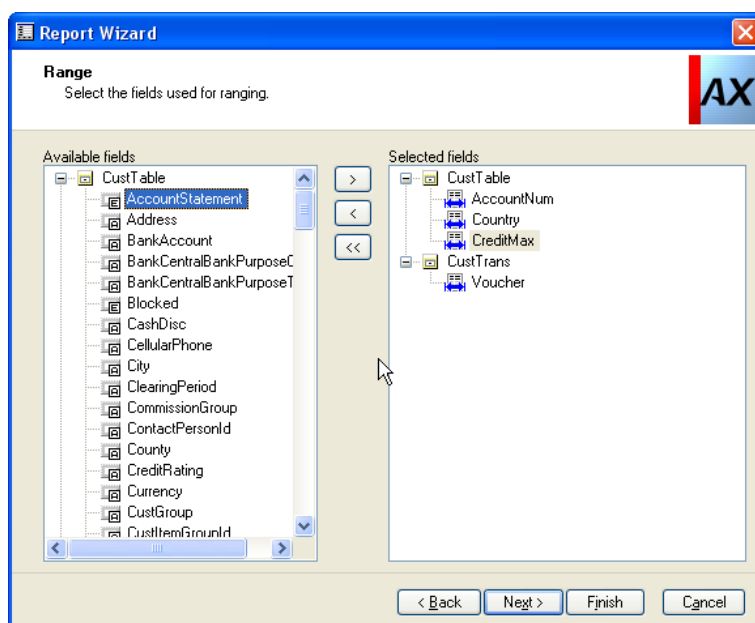


**Шаг 8**

Укажите, будут ли печататься итоги группы при изменении значения одного из выбранных полей сортировки. В приведенном примере групповой итог печатается каждый раз, когда изменяется номера счета в таблице клиентов или дата проводки в таблице проводок клиентов.

**Шаг 9**

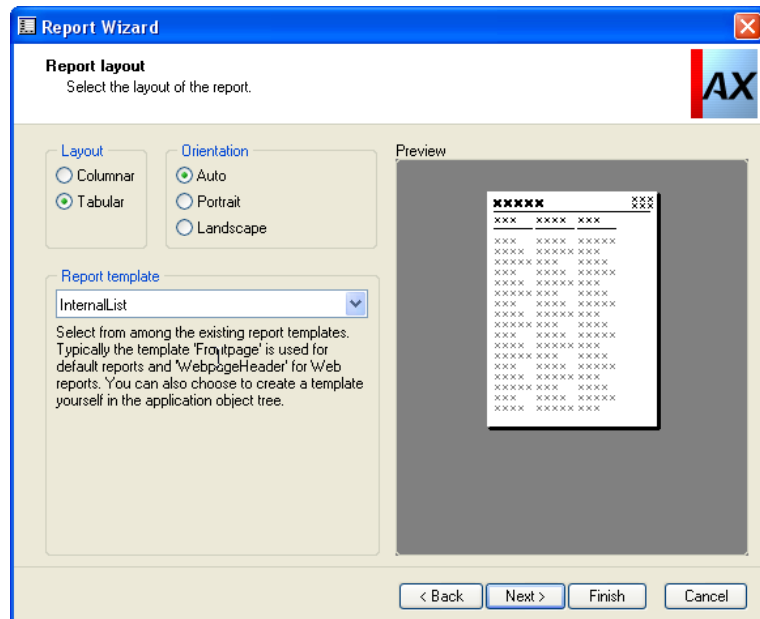
Укажите критерии выборки отчета. По умолчанию будут добавлены все первые поля индексов выбранных источников данных.



**Шаг 10**

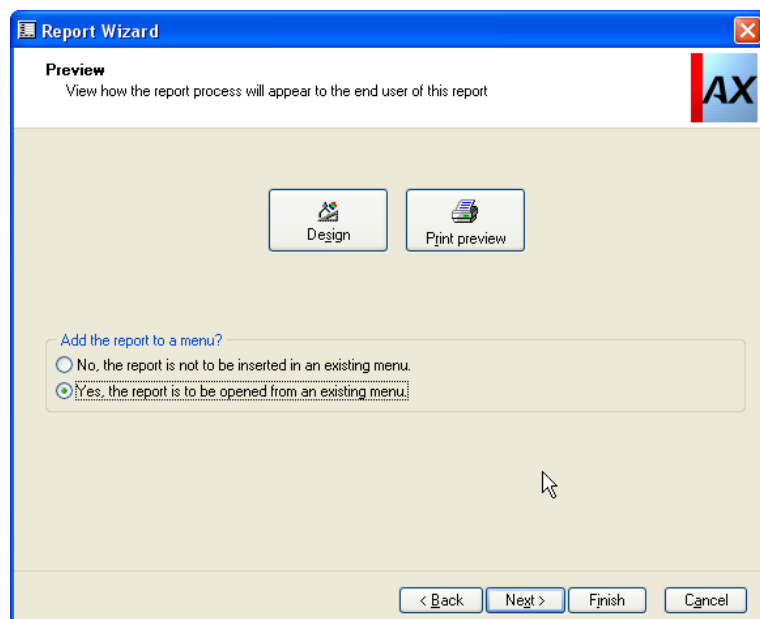
Выберите ориентацию страниц и шаблон для отчета, если он необходим.

Шаблоны отчетов находятся в узле *Report Templates* в AOT (подузел *Reports*).

**Шаг 11**

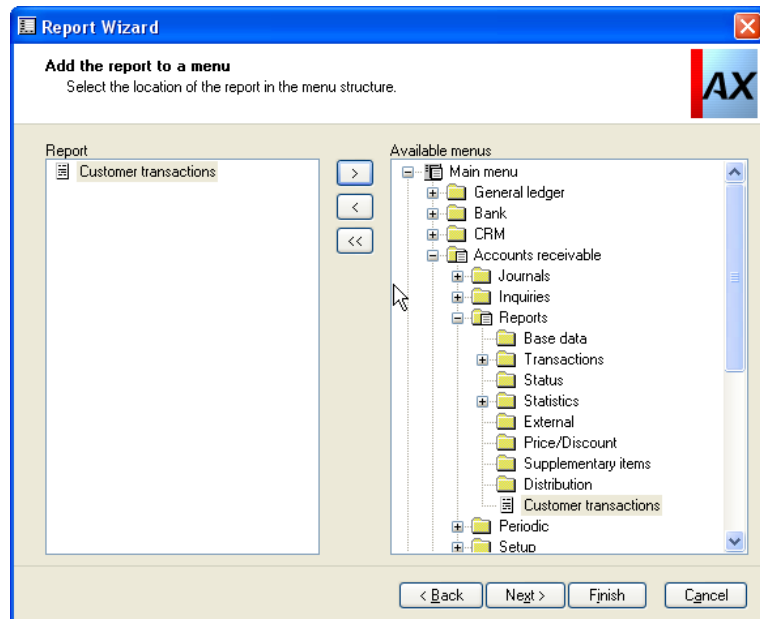
Укажите, будет ли отчет вставлен в одно из существующих меню.

По нажатию кнопки *Дизайн* будет открыт режим просмотра дизайна отчета в AOT. Кнопка *Предварительный просмотр* используется для просмотра созданного отчета с точки зрения конечного пользователя.



**Шаг 12**

Выберите расположение отчета в структуре меню. При добавлении отчета в одно из меню будет автоматически создан пункт меню типа *Output* для вызова отчета.

**Шаг 13**

Все шаги, необходимые Мастеру для создания отчета, выполнены. Нажмите кнопку *Завершение* для создания отчета и сохранения его в АОТ.

