**InfoTask. Описание разработки.**

**Сборка дистрибутива**

**СОДЕРЖАНИЕ**

[1. ОБЩАЯ ИНФОРММАЦИя 2](#_Toc498529364)

[1.1. Каталог репозитория InfoTask 2](#_Toc498529365)

[1.2. Каталоги на сервере 2](#_Toc498529366)

[1.3. Необходимое ПО 2](#_Toc498529367)

[2. Дистрибутивы 4](#_Toc498529368)

[2.1. Сборка дистрибутивов 4](#_Toc498529369)

[2.2. Создание лицензий 4](#_Toc498529370)

[2.3. Структура решения инсталлятора 5](#_Toc498529371)

[3. Структура решения InfoTask 6](#_Toc498529372)

[4. Разработка провайдеров 8](#_Toc498529373)

[4.1. Добавление комплекта провайдеров 8](#_Toc498529374)

[4.2. Написание программ для провайдеров 10](#_Toc498529375)

1. ОБЩАЯ ИНФОРММАЦИя
   1. Каталог репозитория InfoTask

Каталог репозитория InfoTask содержит следующие подкаталоги. Все они содержатся в каталоге InfoTask и заливаются из Git из репозитория InfoTask.

* Release – содержит все используемые файлы Access и все скомпилированные из C# библиотеки и приложения.
* MainSources – содержит основные исходные коды C# для InfoTask. Файл решения - InfoTask.sln.
* GraphicsSources – содержит исходные годы C# для компонентов отображения графиков.
* Installers – содержит исходные коды инсталляторов
* Information – содержит рабочие документы разработки
* Setup – в него формируется дистрибутив. Не заливается в Git.
* Debug – содержит рабочую отладочную копию каталога Release. Не заливается в Git.

Инсталляторы собираются исключительно из каталога Release, но перед этим должны быть построены все исходные проекты в VisualStudio.

* 1. Каталоги на сервере

Общий каталог разработки InfoTask на сервере - [\\Ute02\ute\Git\InfoTask\](file:///\\Ute02\ute\Git\InfoTask\). Он содержит следующие подкаталоги:

* Git – каталог центрального репозитория, на него настроены все локальные репозитории InfoTask сотрудников
* Licenses – список лицензий и фалы активации
* Setup – последние собранные дистрибутивы
* Test – программа тестирования и проекты и данные для нее
* Vesrions – дистрибутивы и исходники разных версий InfoTask
* Work – копии каталогов Debug разных пользователей
* Демонстрация – каталог с демонстрационными проектами
  1. Необходимое ПО

Для разработки InfoTask и сборки дистрибутива на компьютере должны быть установлены:

* GitExtensions. Файл установки

[\\Ute02\ute\Soft\Git\GitExtensions-2.48.05-SetupComplete.msi](file:///\\Ute02\ute\Soft\Git\GitExtensions-2.48.05-SetupComplete.msi).

* VisualStudio 2010. Устанавливается из [\\Ute02\ute\Soft\Microsoft\VisualStudio\SW\_DVD9\_Visual\_Studio\_Pro\_2010\_Russian\](file:///\\Ute02\ute\Soft\Microsoft\VisualStudio\SW_DVD9_Visual_Studio_Pro_2010_Russian\)

Потом устанавливается Service Pack из

[\\Ute02\ute\Soft\Microsoft\VisualStudio\SW\_DVD5\_NTRL\_Visual\_Studio\_2010SP1\](file:///\\Ute02\ute\Soft\Microsoft\VisualStudio\SW_DVD5_NTRL_Visual_Studio_2010SP1\)

* Пакеты для сборки дистрибутивов Windows Installer. Поочередно из каталога [\\Ute02\ute\Soft\Microsoft\VisualStudio\Windows Installer XML\](file:///\\Ute02\ute\Soft\Microsoft\VisualStudio\Windows%20Installer%20XML\) запустить файлы WiX36.exe и ProjectAggregator2.msi.
* Пакет для разработки F#. Установить все файлы из каталога [\\Ute02\ute\Soft\Microsoft\VisualStudio\F#\](file:///\\Ute02\ute\Soft\Microsoft\VisualStudio\F%23\)
* NuGet. Установить [\\Ute02\ute\Soft\Microsoft\VisualStudio\Примочки\NuGet.Tools.vsix](file:///\\Ute02\ute\Soft\Microsoft\VisualStudio\Примочки\NuGet.Tools.vsix)
* ReSharper – устанавливается из файла [\\Ute02\ute\Soft\JetBrains\ReSharperSetup.6.1.1000.82.msi](file:///\\Ute02\ute\Soft\JetBrains\ReSharperSetup.6.1.1000.82.msi)

Ключи лицензий лежат в файле KeysResharper.txt (скорее всего первый для Кати, второй для Жени, третий для Леши)

1. Дистрибутивы
   1. Сборка дистрибутивов

Полная сборка дистрибутивов InfoTask выполняется в следующей последовательности. Все каталоги ниже указаны относительно каталога InfoTask.

1. В Git залить текущую версию с сервера.
2. Подготовка файлов Access:

* Для приложений Access в Office 2010 создать файлы .accde из файлов .accdb (для файлов .accdb, которые менялись).
* Положить файлы .accdb и .accdе в нужные места каталога Release.
* На компьютере с Office 2007 сохранить файлы всех приложений Access в .accde. Полученные файлы положить в каталог Release\InfoTask2007Pack.

1. Подготовка документации (если документация правилась):

* Из DbDoc сохранить .docx файлы всех документов в каталог Release\Docs.
* В каталоге Release\Docs сохранить все файлы .docx как .pdf. Файлы должны называться так же как прежние файлы в каталоге Release\Docs.

1. Построение InfoTask:

* Открыть в VisualStudio 2010 решение InfoTask.sln из каталога MainSources.
* Вверху на панели инструментов выбрать «Release» (возможно вместо «Debug»).
* Нажать F6 или выбрать в меню пункт «Построить решение». Дождаться пока напишут внизу «Построение успешно завершено».
* Открыть в VisualStudio 2010 решение Graphics.sln из каталога GraphicsSources.
* Вверху на панели инструментов выбрать «Release» (возможно вместо «Debug»).
* Нажать F6 или выбрать в меню пункт «Построить решение». Дождаться пока напишут внизу «Построение успешно завершено».

1. Сборка дистрибутивов:

* Открыть в VisualStudio 2010 решение Installers.sln из каталога Installers.
* Вверху на панели инструментов выбрать «Release» (возможно вместо «Debug»).
* Нажать F6 или выбрать в меню пункт «Построить решение». Дождаться пока напишут внизу «Построение успешно завершено».

1. Дистрибутивы формируются в каталог Setup. Оттуда их имеет смысл положить на сервер в каталог \\Ute02\ute\Git\InfoTask\Setup или Versions.
   1. Создание лицензий

Создание лицензии для нового пользователя:

1. Занести лицензию в список:

* Запустить файл [\\Ute02\ute\Git\InfoTask\Licenses\VersionsLicenses.accdb](file:///\\Ute02\ute\Git\InfoTask\Licenses\VersionsLicenses.accdb)
* Выбрать слева последнюю сборку и для нее создать справа новую лицензию
* Указать все реквизиты

1. Создание активатора:

* Открыть в VisualStudio 2010 решение Installers.sln из каталога Installers
* Справа, в обозревателе решений выбрать в проекте «Activator» форму «Form1». Нажать правой кнопкой, выбрать в меню «Перейти к коду»
* В программу скопировать из списка лицензий поля № Лицензии (константа LicenseNumber), Пользователь (константа Company) и Ключ (константа Password)
* Присвоить true или false для включения или отключения активации отдельных приложений (константы ActivateAnalyzer, ActivateConstructor и т.д.)
* В меню «Построение» выбрать пункт «Построить Activator»

После построения файл активации (InfoTaskActivator.exe) лежит в каталоге Setup. Его можно переименовать и скопировать в Licenses

* 1. Структура решения инсталлятора

Решение VisualStudio для сборки инсталляторов содержится в каталоге Installers и состоит из следующих проектов:

1. BaseLibrary – общая библиотека, взята из решения InfoTask.
2. Activator – приложение для активации InfoTask, собирается отдельно от решения индивидуально для каждой лицензии.
3. ActivatorUte – простое приложения для активации InfoTask без пароля для пользователя Уралтехэнерго, собирается отдельно от решения.
4. ActivatorDrop – простое приложения для сброса активации InfoTask, собирается отдельно от решения.
5. VersionSynch – библиотека, содержащая в себе обновления версий различных файлов InfoTask, взята из решения InfoTask. Все изменения прописываются в файле DbVersion.cs. Функции из библиотеки вызываются прямо при установке.
6. KernelPart – пакет инсталляции, содержащий ядро InfoTask. Файл KernelMain.wxs описывает состав дистрибутива, ярлыков, окон установки, регистрации библиотек, дополнительных действий при установке и т.д. В нем много комментариев, можно разбираться.
7. KernelBootstraper – оболочка для пакета KernelPart, делает из него .exe-файл. Нужна для того, чтобы добавить установку сторонних пакетов (.Net и F#).
8. ReporterPart – пакет инсталляции, содержащий все файлы построителя отчетов.
9. ReporterBootstraper – оболочка для пакета ReporterPart, делает из него .exe-файл. Нужна для того, чтобы добавить установку сторонних пакетов, устанавливаемых вместе с InfoTask (VSTO) , а также запускает установку надстройки «Отчеты InfoTask» для Excel.
10. AnalyzerPart – пакет инсталляции, содержащий все файлы анализатора. Копирует только файлы Access.
11. ConstructorPart – пакет инсталляции, содержащий все файлы конструктора. Копирует только файлы Access.
12. RasKosmotronika – пакет инсталляции, содержащий все файлы РАС для Космотроники.
13. KosmotronikaDriver, KvintDriver, LogikaDriver, MirDriver, OvationDriver, SiemensDriver, WonderwareDriver – пакеты инсталляции разных комплектов провайдеров. Каждый из пакетов устанавливается в каталог Providers, А также добавляет информацию о провадерах в General\Config.accdb.
14. Структура решения InfoTask

Решение InfoTask в VisualStudio 2010 содержит в себе все основные исходники на C#. Решение содержится в каталоге InfoTask\MainSources, главный файл InfoTask.sln. Решение содержит следующие проекты.

1. **BaseLibrary** – содержит библиотеки, которые могут быть актуальны не только для InfoTask.

* Папка **Common** содержит класс Different c общими функциями, а также классы-обертки для стандартных словарей, которые используются повсюду в решении.
* Папка **Databases** содержит оболочки для работы с базами данных Access и SQL, в том числе оболочки для рекордсетов с разными возможностями (ADO, DAO, Reader, Bulk Insert). Все рекордсеты наследуют интерфейсы из файла IRecordets.
* Папка **Commands** содержит систему команд, которая обеспечивает ведение истории, выдачу ошибок, обработку опасных операций, отображение индикаторов и т.д.

1. **CommonTypes** – содержит библиотеки, базовые для InfoTask.

* Папка **Values** содержит классы мгновенных значений списков мгновенных значений.
* Папка **Tabls** содержит классы для работы с таблицами для расчета и генерации.
* Папка **Providers** содержит базовые классы для всех провайдеров и сигналов провайдеров, а также форму настройки провайдеров. В файле DllsList реализован общий механизм поздней привязки ко всем провайдерам.
* Папка **OPC** содержит реализацию записи значений в OPC-сервер.

1. **Compiller** – проект на F#, содержит описание синтаксиса языка Tablik и реализацию синтаксического анализа формул.
2. **Tablik** – проект реализующий компиляцию языка Tablik и генерацию. Использует результаты из проекта Compiller. Для этого проекта в свойствах в Сведениях о сборке галочка «Сделать сборку видимой для COM» должна быть установлена.

* Папка **Commands** содержит прежде всего сам класс TablikCompiller, который потом вызывается из приложений Access.
* Папка **Compile** содержит все классы для реализации компиляции.
* Папка **Generator** содержит реализацию генерации формул. Генерация реализована с использованием ANTLR.

1. **Calculation** – проект, реализующий расчеты, потоки расчета и запись в архив.

* Папка **General** содержит статический класс для запуска провайдеров.
* Папка **Calculation** – содержит общие объекты для расчета.
* Папка **Project** – содержит организацию классов для хранения проекта и расчетных параметров во время расчета.
* Папка **Functions** содержит реализацию всех встроенных расчетных функций.
* Папка **CommonProviders** содержит реализацию провайдеров клона, ручного ввода и имитатора.
* Папка **Savers** содержит реализацию формирования ведомостей Access и файла результатов расчета.
* Папка **ArchiveObjects** содержит объекты для записи результатов в архив
* Папка **ArchivesProviders** содержит реализацию архивов и источников архивов для Access и SQL Server
* Папка **Threads** содержит реализацию базового класса потока расчета, потока расчета для вызова из приложений и потока создания клона.

1. **ControllerClient** – содержит классы, вызываемые из приложений Access и осуществляющие расчет или чтение данных. Для этого проекта в свойствах в Сведениях о сборке галочка «Сделать сбору видимой для COM» должна быть установлена.
2. **Controller** – проект монитора расчетов, приложение WPF.

* Папка **Forms** содержит все формы монитора расчетов.
* Папка **Themes** содержит тему WPF для внешнего вида форм монитора.
* Папка **Threads** содержит реализацию потоков монитора и ручного ввода.

1. **ControllerMonitor** – приложение, которое запускается отдельно от монитора расчетов и осуществляет сигнализацию об ошибках в мониторе.
2. **ReporterExcel2010** и **ReporterExcel2007** – надстройки над Excel, добавляющие в Excel меню репортера. Вызывают команды из проекта ReporterCommon.
3. **ReporterCommon** – проект, реализующий построитель отсчетов.

* Папка **General** содержит общие классы работы с Excel.
* Папка **ReportBook** в трех фалах содержит один класс – работу с книгой Excel.
* Папка **Projects** описывает работу в отчете с проектом InfoTask.
* Папка **Objects** содержит объекты для привязки ссылок заполнения ячеек и фигур.
* Папка **Grammar** содержит реализацию языка и отображения для фигур Excel.
* Папка **FormsSetup** содержит формы создания и настройки отчетов.
* Папка **FormsLinks** содержит формы, связанные с установкой ссылок.
* Папка **FormsReport** содержит формы связанные с заполнением отчетов.

1. Проекты комплектов провайдеров, названия начинаются на **Providers** – содержат реализацию провайдеров для разных ПТК. Каждый содержит реализацию источника, сигналов, объектов и, возможно, приемника для конкретного ПТК.
2. **VersionSynch** – содержит обновления версий файлов.
3. **DebugService** – приложения, запускающее отладочные функции
4. **ControllerProba** – приложение для экспериментов.
5. Разработка провайдеров
   1. Добавление комплекта провайдеров

При разработке комплекта провайдеров всегда можно действовать по образу и подобию существующим провайдерам. В целом порядок действий следующий.

1. Придумать коды и создать папки.

* Придумать код комплекта. Создать папку с таким именем в InfoTask\Release\Providers\, это будет папка комплекта провайдеров
* Придумать коды провайдеров: коммуникатора (обычно код комплекта + Comm), источника (обычно код комплекта + Source), приемника (обычно код комплекта + Receiver).

1. Разработать коммуникатор.

* Название файла коммуникатора обычно – код коммуникатора + .accde. Файлы .accdb и .accde, вместе с файлами данных помещаются в папку комплекта провайдеров.
* Для коммуникатора формируется список настроек. Первой настройкой всегда является SourceName – имя источника. Также среди настроек может присутствовать ReceiverName – имя приемника
* В формируемых коммуникатором списках объектов и сигналов обязательно должны быть заполнены поля «Имя источника» и «Имя коммуникатора».
* Поле «Inf» в списке объектов заполняется в соответствии с договоренностями источника.

1. Добавить проект в VisualStudio:

* В VisualStudio создается проект с именем равным Providers + код комплекта провайдеров.
* Для проекта добавляются ссылки на проекты BaseLibrary и CommonTypes и на .Net сборку System.ComponentModel.Composition.
* В свойствах проекта на вкладке «Приложение» указывается требуемая версия .Net Framework - .Net Framework 4 и заполняются сведения о сборке.
* В свойствах проекта на вкладке «Построение» для конфигураций отдельно для редакций Release и Debug указывается «Путь вывода» в каталог комплекта провайдеров и его копию в каталоге InfoTask\Debug.
* Создаются классы сигналов, источника и приемника если нужно. Подробно о написании кода для провайдеров см. в следующем разделе.

1. Добавить провайдеры в Config:

* Открыть файл InfoTask\Release\General\ConfigTemplate.accdb
* Для каждого добавляемого провайдера добавить в SysTabl строку: ParamName – код провайдера, ParamType – Provider, ParamValue – тип провадера, т.е. Коммуникатор, Источник или Приемник, ParamDescription – описание.
* Для каждого добавляемого коммуникатора добавить в SysSubTabl строчки со следующими значениями полей:
* SubParamName = ProviderFile, SubParmValue = путь к файлу коммуникатора
* SubParamName = AppVersion, SubParamValue = версия InfoTask
* SubParamName = AppVersionDate, SubParamValue = дата версии InfoTask
* Остальные свойства индивидуальны для разных источников. SubParamNum заполняется, начиная с 1, в порядке, в котором нужно отображать свойства в форме настроек. Первым свойством всегда идет SourceName – имя источника
* Для каждого добавляемого источника добавить в SysSubTabl строчки со следующими значениями полей:
* SubParamName=ProviderFile, SubParmValue = путь к файлу dll комплекта провайдеров
* SubParamName=JointProviders, SubParmValue = CloneSource (обычно)
* Остальные свойства индивидуальны для разных источников. SubParamNum заполняется, начиная с 1, в порядке, в котором нужно отображать свойства в форме настроек.
* Для каждого добавляемого приемника добавить в SysSubTabl строчки со следующими значениями полей:
* SubParamName=ProviderFile, SubParmValue = путь к файлу dll комплекта провайдеров
* Остальные свойства индивидуальны для разных приемников, SubParamNum заполняется, начиная с 1, в порядке, в котором нужно отображать свойства в форме настроек.

1. Написать руководство по эксплуатации для провайдеров, сохранить в каталог InfoTask\Release\Docs, сохранить как PDF, назвать по аналогии с руководствами по другим провайдерам.
2. Написать программу обновления Config:

* В VisualStudio открыть решение Installers, проект ConfigFill, файл ConfigCA.cs.
* Нужно добавить два метода: один для заполнения Config при установке провайдера, другой для очистки Config при удалении провайдера. Методы можно добавлять по аналогии с методами ConfigFillK и ConfigFillKU (методы для заполнения и очистки Config для Космотроники). Названия методов любые, они будут использоваться в инсталляторе.
* Перед методами пишется модификатор [CustomAction]
* В первом методе копируются строчки из SysTabl и SysSubTabl файла ConfigTemplate.accdb в файл Config.accdb. По одному вызову метода CopyConfigProp для каждого провайдера из комплекта.

Во втором методе удаляются провайдеры из SysTabl файла Config.accdb. Имена провайдеров перечисляются в условии запроса.

1. Разработать отдельный инсталлятор для комплекта провайдеров:

* В VisualStudio открыть решение Installers. В обозревателе решений добавить новый проект. Тип проекта выбрать Wix Toolset \ Setup Project (проекты WiX добавляются в VisualStudio при установке Windows Installer XML). Имя проекта указать = Код комплекта + Driver.
* Зайти в свойства проекта, указать:
* На вкладке Installer: Output name = Код комплекта + Providers
* На вкладке Build для конфигураций Debug и Release: Cultures to build = ru-RU. Output path: ..\..\Setup\. Установить галочку Suppress output of the wixpdb files.
* Добавить в проект .wxs-файл.
* Заполнить .wxs-файл по аналогии с файлом KosmotronikaMain.wxs в проекте KosmotronikaDriver.
* Везде где в тексте есть «Kosmotronika» ее нужно менять на код комплекта. Также нужно указать корректные пути в к файлам коммуникатора, к dll источника и прочим файлам лежащим в каталоге InfoTask\Providers\Код комплекта, а также к файлу документации.
* Среди прочего в файл добавляются вызовы функций заполнения и очистки таблиц в Config (Custom Action, функции описаны в предыдущем пункте).
* Везде, где используются GUID каждый раз нужно вставлять новые уникальные, GUID можно сгенерировать в VisualStudio из меню Сервис\Создать GUID.
  1. Написание программ для провайдеров

1. Создание проекта.

* Для нового комплекта провайдеров в VisualStudio в решении MainSources создается новый проект с именем Providers + код комплекта.
* Для проекта добавляются ссылки на проекты BaseLibrary и CommonTypes, а также на .Net сборку System.ComponentModel.Composition
* В свойствах проекта на вкладке «Приложение» указывается требуемая версия .Net Framework - .Net Framework 4. Заполняются сведения о сборке по аналогии с другими провайдерами, при этом галочка «Сделать сбору видимой для COM» должна быть снята.
* В свойствах проекта на вкладке «Построение» для конфигураций отдельно для редакций Release и Debug указывается «Путь вывода» в каталог комплекта провайдеров (.\..\Release\Providers\ + код комплекта + ) и его копию в каталоге InfoTask\Debug соответственно.

1. Класс источника.

* В проект добавляется класс, совпадающий с придуманным кодом источника
* Класс создается в пространстве имен Provider
* Класс наследуется от класса SourceBase и интерфейса ISource из проекта CommonTypes.
* Перед описанием класса добавляются атрибуты

[Export(typeof(IProvider))]

[ExportMetadata("Code", код провайдера)]

Это нужно для того, чтобы расчет использовал dll провайдера, но сам проект провайдера был бы не привязан к расчету. Таким образом, можно разрабатывать новые провайдеры не меняя ядро InfoTask.

* Реализовать свойства Code – код провайдера
* Реализовать свойства Inf – строка свойств провайдера через ;, при описании метода set строку свойств нужно разобрать
* Реализовать метод Dispose – для закрытия всего, что должно быть закрыто при закрытии провайдера
* Реализовать методы AddMenuCommands и ComboBoxList позволяют задать пункты меню и выпадающие списки в форме настроек. Форма настроек для всех провайдеров одна и реализована она в проекте CommonTypes.
* Реализовать методы Connect, Check, CheckSettings, CheckConnection, CheckConnectionMessage, они нужны для соединения с провайдером в разных случаях: при первичном соединении, при проверке соединения, после ошибки и в для проверки соединения в форме настроек.
* Реализовать метод GetTime, возвращающий интервал – диапазон источника.
* При разработке источников можно активно использовать класс Logger, записывающий в историю, запускающий опасные операции (Danger) и управляющий процентами индикатора.

1. Сигнал источника.

* В проект добавляется класс – сигнал для данного источника (например SignalOvation)
* Класс наследуется от класса ProviderSignal
* Реализуются все свойства сигнала для получения данных
* Реализуется конструктор, инициализирующий свойства из ProviderSignal
* Могут быть реализованы отдельные операции для получения значений

1. Объект источника

* Если источник получает данные через OLE DB (а это пока было чаще всего так), то можно воспользоваться некоторыми наработками. В этом случае сигналы группируются в объекты, и данные получаются сразу для объекта. Заметим, что объеты для получения из источника – это скорее выходы в нашей терминологии, и они не всегда совпадают с объектами из таблицы объектов.
* В проект добавляется класс объекта. Класс наследуется от ProviderObject
* Объект содержит поля – ссылки на свои сигналы, могут быть коллекции сигналов.
* Также объект содержит поля вроде Id или кода для идентификации объекта при чтении данных из архива
* В объекте часто есть методы для формирования срезов (со словами Begin и End)

1. Список сигналов или объектов

* В источнике обычно присутствуют коллекции сигналов или объектов для чтения
* Всегда должны быть реализованы методы
* ClearSignals – очищает все внутренние списки сигналов и объектов
* AddSignal – добавляет сигнал в источник, вписывая его куда надо в коллекции сигналов и объектов. В метод передается код, тип данных сигнала и свойства сигнала из поля Inf таблицы сигналов.
* Prepare – подготавливает сигналы источника к расчету. Если расчет периодический, то будет вызываться один раз перед первым расчетом.

1. Чтение данных.

* В каждом источнике реализован метод GetValues, который производит считывание значений.
* На самом деле в классе SourceBase есть метод GetValues c указанием диапазона времени считывания данных, который уже вызывает метод GetValues без параметров. Кроме того, присваиваются значения полям BeginRead и EndRead, которые доступны при реализации считывания данных.
* Если данные читаются не из OLE DB, то реализация не использует никаких специальных методом из SourceBase (см. пример KvintSource)
* Обычно отдельно читается срез данных на начало периода и отдельно читаются изменения значений за период. Во время или после чтения среза, значения среза должны быть добавлены в список значений сигнала (AddBegin). После чтения изменений, сразу формируется значение среза для следующего период (MakeEnd).
* Если данные читаются не из OLE DB, то реализовано чтение данных по частям (см. пример OvatonSource).
* Коллекции объектов разбиваются на блоки. Количество объектов блоке определяется возможностью чтения из ПТК.
* Вызывается функция ReadValuesByParts, принимающая на вход функцию, запрашивающую из источника рекордсет значений, функцию выполняющую чтение значений, и сам блок значений
* Чтение данных обычно реализуется проходом по рекордсету. Для каждой строчки определяется, значения какого объекта она содержит, формируются значения всех сигналов объекта. При этом подсчитывается количество прочитанных и сформированных значений, которое потом пишется в историю.

1. Создание клонов.

* Создание клонов реализуется автоматически при реализации считывания данных, если источник является наследником SourceИфыу (уже реализовано методом MakeClone).
* Отдельно нужно только передать idInClone из конструктора сигнала в конструктор базового сигнала
* После чтения значений нужно делать MakeEnd, чтобы записать в клон недостающие значения до конца периода

1. Приемники

* Приемники делаются также как источники, также вызываются из dll уже во время выполнения, также содержат методы для настройки, соединения, списки сигналов. Приемник обычно создается в том же проекте, что и источник соответствующего ПТК.
* В приемнике всегда реализован метод WriteValues, записывающий заранее подготовленные значения сигналов в ПТК
* Все приемники на данный момент – это реализация в OPC-сервер DA. Для этого случая в проекте CommonTypes есть специальный базовый класс OpcServer, все приемники наследуются от него.
* Для приемников – наследников класса OpcServer достоточно переопределить:
* код (Code)
* настройки дополняющие базовые настройки OpcServer (GetAdditionalInf)
* способ формирования тега сигнала по настройкам приемника (GetOpcItemTag).