

Опция «API на запись и чтение для АИК «ДЕЛО-WEB»

Опция «API на запись и чтение для АИК «ДЕЛО-WEB» предназначена для разработки прикладных программ, формирования произвольных отчетов, справок, получения статистических данных, а также для экспорта данных из системы "Дело" в другие системы.

Опция разработана для осуществления доступа к данным системы "Дело" из прикладной программы пользователя, исполняемой на "клиентской" рабочей станции.

API на запись и чтение для АИК «ДЕЛО-WEB» представляет собой набор объектов, выполненных по ActiveX технологии. Объекты API могут быть использованы в прикладной программе пользователя, написанной на любом языке программирования, поддерживающем работу с ActiveX объектами. Также объекты API могут быть использованы в системах, имеющих в своем составе язык MS Visual Basic for Application - MS WORD for WINDOWS, MS EXCEL, MS ACCESS.

1.1.1. Объекты подключения к БД и поиска данных

Объект типа "Head" - головной объект, предназначен для:

- установления связи с БД системы и хранения информации о ней;
- подключения пользователя в БД ДЕЛО и хранения информации об этом пользователе;
- создания объектов остальных типов.

Объект типа "SearchVocab" предназначен для задания и хранения опций выбора записей (элементов) справочников.

Объект типа "SearchTables" предназначен для задания и хранения опций выбора документов или резолюций из БД системы "Дело".

Объект типа "ResultSet" предназначен для осуществления отбора и хранения результатов этого отбора.

1.1.2. Объекты хранения информации об элементах линейных справочников

Объект типа "Delivery" - элемент справочника Виды доставки;

Объект типа "Citizen" - элемент справочника Граждане;

Объект типа "Security" - элемент справочника Грифы доступа;

Объект типа "Link" - элемент справочника типов связей;

Объект типа "RollType" - элемент справочника Типы реестров;

Объект типа "User" - элемент справочника Пользователей;

1.1.3. Объекты хранения информации об элементах иерархических справочников

Объект типа "Department" - элемент справочника Подразделений;

Объект типа "Organiz" - элемент справочника Организаций;

Объект типа "Rubric" - элемент справочника Рубрик;

Объект типа "DocGroup" - элемент справочника Группы документов;

Объект типа "Regions" - элемент справочника Регионы.

1.1.4. Объекты хранения информации основных данных

Объект типа "Rcln" - РК входящего документа (и письма);

Объект типа "RcOut" - РК исходящего (внутреннего) документа;

Объект типа "RcLet" - РК письма гражданина;

Объект типа "Resolution" - резолюции документа (РК);

Объект типа "RcPrj" - РК проекта документа;

Объект типа "Roll" - реестр отправки документов;

1.1.5. Объекты хранения информации, детализирующие другие объекты

Объект типа "UserCard" - права пользователя в Картотеке;

Объект типа "Contacts" - контакты Организации;

Объект типа "BankRec" - банковские реквизиты Организации;

Объект типа "AccessCard" - принадлежность РК картотекам;

Объект типа "Correspondent" - корреспонденты и Авторы сопроводительного документа РК;

Объект типа "Author" - заявители письма (РК);

Объект типа "LinkCard" - связки РК;

Объект типа "Addr" - адресаты РК;

Объект типа "Files" - прикрепленные к РК файлы;

Объект типа "Contents" - содержание прикрепленного файла;

1.2. Технология выборки данных из системы "Дело"

Для того чтобы начать работу с БД системы "Дело" необходимо:

- создать головной объект: `Set MyHead = CreateObject("eapi.head")`
- установить связь с БД: `MyHead.Open`

На этом этапе объект MyHead запускает процесс установления связи с БД и подключения пользователя к системе "Дело". Пользователь должен ввести свой идентификатор и пароль. Процесс проверит, что БД запущена, представившийся пользователь зарегистрирован в системе, и передаст всю информацию о связи с БД и работающем пользователе в объект MyHead.

Успешное выполнение метода приводит к созданию объекта типа "User", присвоении ссылки на этот объект свойства UserInfo и установлению значений свойств объекта MyHead:

```
Active = True
```

```
ErrCode = 0
```

```
ErrText = ""
```

В случае ошибки – свойствам ErrCode и ErrText присваиваются соответствующие значения.

Для того чтобы отобразить нужную информацию из БД (документы, резолюции или элементы справочников), необходимо создать хранилище для них – объект типа "ResultSet". Это можно сделать следующим образом:

```
Set MyResultSet = MyHead.GetResultSet
```

Далее нужно определить какую информацию необходимо получить и задать критерии отбора.

Если необходимо получить информацию из справочников системы, то нужно создать объект типа "SearchVocab" и установить ссылку на него в свойстве Source объекта MyHead:

```
Set MyResultSet.Source = MyHead.GetCriterion("Vocabulary")
```

Если необходимо получить перечень документов или резолюций, то нужно создать объект типа "SearchTables" и установить ссылку на него в свойстве Source объекта MyHead:

```
Set MyResultSet.Source = MyHead.GetCriterion("Table")
```

После этого необходимо задать критерии отбора.

Теперь можно приступать к отбору информации из БД ДЕЛО и записи ее в хранилище:

```
MyResultSet.Fill
```

Выполнение этого метода приводит к формированию одного или нескольких SQL запросов к БД, если это необходимо, к вызову визуального интерфейса с пользователем. В результате выполнения этого метода создадутся объекты, удовлетворяющие критерию отбора, и на них установятся ссылки в множественном свойстве Item(), в множественном свойстве ObjectType(), будут находиться наименования типов соответствующих объектов, а в свойстве ItemCnt – количество созданных объектов.

В случае ошибки – свойствам ErrCode и ErrText присваиваются значения отличные от нулевого (0) и пустого (""), значений соответственно.

Если количество отобранных элементов больше 0, то приложение может обращаться к свойствам созданных объектов. Например, если был отобран один входящий документ и необходимо получить его регистрационный номер, аннотацию и наименования всех корреспондентов, необходимо выполнить следующие команды:

```
/*Обращение к свойствам Рег.№ и Аннотация объекта типа RcIn */
MyRegNum = MyResultSet.Item(0).RegNum
MyContents = MyResultSet.Item(0).Contents
/*Обращение к множественному свойству*/
For i = 0 to MyResultSet.Item(0).CorrespCnt - 1
    /*i - й корреспондент, и через справочник Организаций к его наименованию*/
    MyCorresp(i) = MyResultSet.Item(0).Corresp(i).Organiz.Name
Next i
```

Если же были отобраны элементы справочника "Пользователей", и в приложении необходимо получить Фамилию i – го пользователя и перечень его прав в главной картотеке, следует записать следующую последовательность операторов:

```
/*Обращение к свойству Имя пользователя объекта типа "User"*/
MyUserName = MyResultSet.Item(i).Name
For j = 0 to MyResultSet.Item(i).CardCnt - 1
    /*Обращение к множественному свойству доступные пользователю картотеки */
    If MyResultSet.Item(i).Card(j).Main Then
        /*и поиск главной из них*/
        MyRight(1) = MyResultSet.Item(i).Card(j).FuncList("Admin")
        MyRight(2) = MyResultSet.Item(i).Card(j).FuncList("DelRc")
        MyRight(3) = MyResultSet.Item(i).Card(j).FuncList("RegDoc")
        MyRight(4) = MyResultSet.Item(i).Card(j).FuncList("EditRc")
        MyRight(5) = MyResultSet.Item(i).Card(j).FuncList("Search")
        MyRight(6) = MyResultSet.Item(i).Card(j).FuncList("Reports")
        MyRight(7) = MyResultSet.Item(i).Card(j).FuncList("ExecResol")
        MyRight(8) = MyResultSet.Item(i).Card(j).FuncList("AllCard")
        MyRight(9) = MyResultSet.Item(i).Card(j).FuncList("RemarkDoc")
        MyRight(10) = MyResultSet.Item(i).Card(j).FuncList("SendDoc")
    End If
```

Next j

По миновании надобности хранения результатов выборки из БД, для освобождения оперативной памяти рекомендуется очищать хранилище `MyResultSet`, либо применив к нему метод `Clear`, либо разорвав ссылку на ненужный объект, создав ссылку на новый объект.

Если приложение "знает" системный номер или код элемента справочника, системный номер записи о РК документа или резолюции (их значения были присвоены переменным приложения до очистки хранилища), то по мере надобности информацию об элементе, РК или резолюции можно получить, создав соответствующие объекты непосредственно из головного объекта:

```
Set MyRcIn = MyHead.GetRow("RcIn", MyRcISN)
```

или

```
Set MyDepartment = MyHead.GetRow("Department", MyDCode)
```

Метод `GetRow` имеет два обязательных параметра: первый определяет тип создаваемого объекта, а второй – уникальный номер или код элемента, информацией о котором заполняется создаваемый объект. Для всех типов объектов, кроме иерархических справочников, второй параметр должен быть числовым и он интерпретируется как системный номер записи об элементе (свойство `ISN`). Для иерархических справочников – второй параметр может быть как числовым и тогда он интерпретируется как системный номер записи об элементе (свойство `ISN`), так и строковым, тогда он интерпретируется как уникальный код элемента (свойство `DCode`).

В случае ошибки, системной (потеряна связь с БД) или программиста (в метод был передан несуществующий системный номер или код), метод создаст "пустой" объект, т.е. объект, имеющий только два свойства `ErrCode` и `ErrMsg`.

1.2.1. Задание критериев отбора элементов справочников

Задание критериев отбора элементов справочников осуществляется заданием свойств объекта типа `"SearchVocab"`, на который ссылается свойство `Search` объекта `"ResultSet"`.

Обязательно задаваемым свойством объекта типа `"SearchVocab"` является свойство `Vocabulary`, определяющее справочник из которого будут отбираться элементы. Остальные свойства – являются необязательными определяемыми пользователем т.к. они имеют значение по умолчанию.

Для линейных справочников (`"CardIndex"`; `"Citizen"`; `"Delivery"`; `"FileType"`; `"Link"`; `"RollType"`; `"Security"`; `"User"`;) могут быть заданы следующие свойства:

Deleted – определяет, подлежат ли отбору логически удаленные элементы справочника или нет;

Select – определяет количество элементов подлежащих отбору (один, несколько или все) и определяет режим отбора: автоматический - `Select = All` или с использованием визуального интерфейса - `Select = One` или `Select = Many`).

Для иерархических справочников (`"Department"`; `"DocGroup"`; `"Nomencl"`; `"Organiz"`; `"Rubric"`;) могут быть заданы следующие свойства:

Deleted – определяет, подлежат ли отбору логически удаленные элементы справочника или нет;

ItemType – определяет тип элементов, подлежащих отбору (листья, вершины или и те и другие);

Base – определяет элемент иерархического справочника и ограничивает отбор поддеревом, начинающимся с этого элемента;

Select – определяет количество элементов подлежащих отбору: один, несколько, все или все элементы, расположенные непосредственно под элементом, указанным в свойстве `Base` и определяет режим отбора (автоматический `Select = All` или `Select = OneLevel`; или с использованием визуального интерфейса `Select = One` или `Select = Many`).

В том случае, если встроенный визуальный интерфейс, отображающий элементы справочника и позволяющий пользователю отметить элементы, подлежащие отбору (`Select = One` или `Select = Many`), не устраивает пользователя, то в приложении можно построить свой визуальный интерфейс, используя отбор из справочника (`Select = All` или `Select = All OneLevel`).

1.2.2. Задание критериев отбора документов и резолюций

Задание критериев отбора документов или резолюций осуществляется заданием свойства "Params" объекта типа "SearchTables", на который ссылается свойство Search объекта "ResultSet".

Сначала необходимо указать, что будет являться результатом отбора: документы или резолюции. Для этого необходимо определить свойство "Params" с параметром "Result". Если результатом отбора будут резолюции, то следует, например, записать:

```
MyResultSet.Source.Params("Result") = "Resol"
```

Если этого не делать, то результатом отбора будет совокупность документов.

Если необходимо отобразить документы, то в некоторых случаях можно ускорить получение результата, уточнив какой вид документов необходимо отобразить: исходящие, входящие или письма. Это делается заданием свойства "Params" с параметром "DocKind". Например, для исходящих документов:

```
MyResultSet.Source.Params("DocKind") = "Out"
```

Для задания условий отбора необходимо определить значение свойства "Params" с одним или несколькими параметрами, приведенными в таблице "Перечень параметров свойства Search.Params и их значений". Например, для того чтобы отобразить документы, зарегистрированные в январе 1998 года, и содержащие в поле "Краткое содержание" слово "договор", необходимо записать:

```
MyResultSet.Source.Params("Rc.DocDate") = "01/01/1998:31/01/1998"
```

```
MyResultSet.Source.Params("Contents") = "договор"
```

1.3. Технология изменения данных системы "Дело"

Сначала необходимо установить соединение с БД системы "Дело", так же, как описано в разделе **"Технология выборки данных из системы "Дело"**.

Для того чтобы выполнить изменение данных, необходимо создать объект процедуру – модификатор. Например, чтобы добавить в РК документа одну или несколько тематических рубрик, необходимо записать:

```
Set proc = objHead.GetProc("add_rubric")
```

Далее нужно определить аргументы модификатора.

```
proc.Parameters.Append proc.CreateParameter("aIsnDoc", 3, 1, 0, 4731)
```

```
proc.Parameters.Append proc.CreateParameter("codes", 200, 1, 2000, "0.2UY.|0.2VA.")
```

```
proc.Parameters.Append proc.CreateParameter("acard", 200, 1, 48, Null)
```

Теперь можно выполнить хранимую процедуру для изменения данных:

```
objHead.ExecuteProc(proc)
```

Выполнение этого метода приводит к вызову соответствующей хранимой процедуры (в данном случае add_rubric).

В случае ошибки – свойствам ErrCode и ErrText присваиваются значения отличные от нулевого (0) и пустого (""), значений соответственно.

Описание всех имеющихся в системе модификаторов данных см. в разделе **"Ошибка! Источник ссылки не найден."**

1.4. Администрирование приложений с обращением к системе "Дело"

Создавая новое приложение по формированию отчетной формы или проведению справочно-аналитической работы на данных из системы "Дело" необходимо помнить, что с этим приложением будут работать разные пользователи системы. Эти пользователи могут иметь в системе различные права доступа к документам. Это ограничения по грифу документа; по доступу к картотекам и право поиска документов во всех картотеках. Поэтому предоставляя пользователю право формирования запроса на поиск документов или резолюций к ним необходимо добавлять автоматические ограничения на поиск.

Права текущего пользователя, после успешного установления связи с БД, можно получить в свойствах:

MyHead.UserInfo.Security(i) – перечень грифов доступа документов, доступных данному пользователю;

MyHead.UserInfo.Card(i) – перечень доступных данному пользователю картотек. Если доступных картотек несколько (больше одной), то перед началом поиска информации в БД нужно установить одну из них используемой по умолчанию и дать пользователю поменять картотеку на одну из доступных ему.

Например, запрос, описанный в разделе **"Задание критериев отбора документов и резолюций"**, справедлив только в том случае, если необходимо найти все документы во всех картотеках системы. Как правило, пользователя интересуют только документы его картотеки и ему нельзя предоставлять документы с грифами, отличными от доступных ему. Поэтому к запросу необходимо добавить дополнительные условия:

```
MyResultSet.Source.Params("Access.CardIndex") = MyHead.UserInfo.Card(0).CardIndex.Dcode
MyStr = ""
For i = 0 to MyHead.UserInfo.SecurCnt - 1
    MyStr = MyStr + MyHead.UserInfo.Security(i).ISN
    If i < MyHead.UserInfo.SecurCnt - 1 Then
        MyStr = MyStr + "|"
    End If
Next i
MyResultSet.Source.Params("Rc.Security") = MyStr
```

1.5. Особенности использования Объектов

Наименования свойств и методов всех объектов, а также значения всех вводимых параметров (кроме значений полей БД) могут быть записаны в любом регистре.

Все строковые значения, возвращаемые объектами (речь идет о "параметрических" свойствах, а не об "информационных" свойствах – значениях полей БД) записаны в ВЕРХНЕМ регистре (прописные буквы).

Полное описание объектов см. в разделе **"Ошибка! Источник ссылки не найден."**

1.5.1. Объект типа "Head"

Если в приложении уже существует объект типа "Head" с установленной связью с БД, то при создании нового объекта типа "Head" он унаследует информацию о связи с БД и подключенным пользователем от уже существующего объекта.

При повторном использовании метода "Open", предыдущая связь с БД прерывается и осуществляется новая связь с БД и новое подключение пользователя к системе.

После окончания работы с БД рекомендуется прервать связь с БД, используя метод "Close". Если этого не сделать, то связь прекращается только при выходе из приложения.

1.5.2. Объект типа "ResultSet"

Результат отбора сортируется следующим образом:

- для справочников при отборе пользователем в порядке заданным пользователем;
- для справочников при автоматическом отборе (без визуального интерфейса):
 - для линейных справочников в порядке возрастания их веса (свойство Weight), для справочника "Граждане" (Citizen) по свойствам Name и City, для CardIndex по свойству Name;
- для иерархических справочников:
 - если Select = All - в произвольном порядке;
 - если Select = OneLevel – сначала вершина, а потом подчиненные ей элементы в порядке возрастания их веса (свойство Weight);
- для других объектов (РК или резолюции)- порядок сортировки должен задаваться в "SearchTables".

1.5.3. Объект типа "SearchTables"

Для формирования критериев отбора документов или резолюций можно определять ограничения на любую совокупность реквизитов, указанных в таблице. Если условия определены для нескольких реквизитов одновременно, то они объединяются между собой логическим "И" ("AND"). Если одновременно определены условия для свойств объектов разного уровня (например, для реквизитов основного раздела РК и раздела "Корреспондент"), то они соединены логическим "И хотя бы в одной подчиненной записи".

Например, следующие: два условия:

```
Params("Rc.DocDate") = "01/01/1998:31/01/1998"
```

```
Params("Resolution.PlanDate") = "IsNotNull"
```

означают, что будут отбираться документы, зарегистрированные в январе 1998 года, и имеющие хотя бы одну контрольную резолюцию. Результат отбора (документы или резолюции) – не отсортирован.

1.6. Введение в процессный API

Подсистема управления процессами предоставляет возможность разрабатывать, выполнять и контролировать ход выполнения процессов (автоматических процедур) для системы Дело.

Процессы предназначены для решения таких задач, как автоматизация маршрутизации и обработки документов, интеграция со сторонними системами, оповещение пользователей о событиях в системе по электронной почте.

Процессы выполняются специальной службой, входящей в состав серверной части системы Дело. Для того чтобы служба могла начать выполнение процесса, библиотека, содержащая процесс, должна быть предварительно загружена¹ в базу данных и сконфигурирована. После этого служба может начать выполнение процессов, содержащихся в загруженных библиотеках, автоматически, либо по запросу пользователя через интерфейс приложений системы Дело.

Для разработки процессов с использованием процессного API, разработчик должен владеть технологией Workflow Foundation (WF) входящей в состав Microsoft .NET Framework 3.5.

Процессный API предоставляет следующие возможности:

- Работать с объектами БД с использованием строготипизированной модели данных
- Выполнять поиск объектов в БД по заданным критериям
- Создавать и модифицировать объекты в БД.
- Получать оповещение о событиях создания объектов БД и изменения их состояния.
- Работать с контекстом выполняющегося процесса.

Все перечисленные возможности предоставляются через специализированные интерфейсы, доступ к которым из процесса осуществляется посредством стандартных активностей библиотеки WF: `CallExternalMethodActivity` и `HandleExternalEventActivity`. Также в состав API входит комплект активностей, позволяющих выполнять взаимодействие с БД Дело более удобным способом, и предоставляющих дополнительные функции, такие как работа с сообщениями электронной почты. Полный перечень этих активностей приведен в разделе **"Ошибка! Источник ссылки не найден."**.

1.7. Модель данных, используемая в процессном API

Модель данных строится на следующих понятиях:

- Атрибут – обычно соответствует полю таблицы БД. Атрибут может быть одного из следующих типов:
 - Строка
 - Целое число
 - Дата (и время)
 - Дробное (десятичное) число
 - Перечисление
 - Коллекция сущностей
 - Ссылка на сущность

¹ Загружать в систему можно только скомпилированные процессы, загрузка процессов в формате xml не поддерживается.

- Сущность – соответствует строке таблицы БД. Все сущности имеют общего предка – т.е. наследуются от базового класса EntityBase. Сущности, представляющие справочники, наследуются от класса ClassifEntity или от ClassifTreeEntity².
- Коллекция сущностей – объединяет множество сущностей одного типа. Тип сущностей, находящихся в коллекции может быть абстрактным. Соответствует набору строк таблицы.
- Ссылка на сущность – представляет идентификатор сущности. Каждый класс сущности имеет соответствующий ему класс ссылки на эту сущность. Классы ссылок на сущности образуют иерархию наследования, аналогичную иерархии классов сущностей. В вершине иерархии наследования находится абстрактный класс EntityBaseRef.
- Объект системы – представляет иерархическую структуру из сущностей с единственной сущностью в корне структуры – корневой сущностью объекта. Корневая сущность объекта может иметь атрибуты типа "Коллекция сущностей", в которых хранятся дочерние сущности объекта, которые, в свою очередь, также могут иметь свои дочерние сущности. Объект системы является единицей загрузки данных из БД.

Классы модели данных системы Дело определены в сборке Eos.Delo96.Api, а базовые классы в сборке Eos.Delo.Common.

Рассмотрим описанные понятия на примере некоторых сущностей системы (пропущенные части кода обозначены многоточием).

```
namespace Eos.Delo96.Api.Entities
{
    [Serializable]
    public class DOC_RC_Ref : Eos.Delo.Common.EntityBaseRef
    {
        public DOC_RC_Ref(int ISN_DOC) { ... }
        public int ISN_DOC { get { ... } }
    }

    [Serializable]
    public class DOC_RC_Entity : Eos.Delo.Common.EntityBase
    {
        ...
        public int ISN_DOC { get { ... } }
        public KIND_DOC KIND_DOC { get { ... } }
        public String DUE_DOCGROUP { get { ... } }
        public DateTime DOC_DATE { get { ... } }
        public String FREE_NUM { get { ... } }
        ...
        public List<AR_RC_VALUE_Entity> AR_RC_VALUE_List { get { ... } }
        public List<DOC_WHO_Entity> DOC_WHO_List { get { ... } }
        public List<DOC_SIGN_Entity> DOC_SIGN_List { get { ... } }
    }
}
```

² Класс ClassifTreeEntity унаследован от класса ClassifEntity, а класс ClassifEntity, в свою очередь, от класса EntityBase. Таким образом любая сущность имеет общего предка EntityBase.

```

    public List<FORWARD_Entity> FORWARD_List { get { ... } }
    public List<JOURNAL_Entity> JOURNAL_List { get { ... } }
    public List<REF_ACCESS_CARD_Entity> REF_ACCESS_CARD_List { get { ... } }
    ...
}

[Serializable]
public class FORWARD_Ref : Eos.Delo.Common.EntityBaseRef
{
    public FORWARD_Ref(int ISN_FORWARD) { ... }
    public int ISN_FORWARD { get { ... } }
}

[Serializable]
public class FORWARD_Entity : Eos.Delo.Common.EntityBase
{
    ...
    public int ISN_FORWARD { get { ... } }
    public int? ISN_DOC { get { ... } }
    ...
    public String DUE_ADRESAT { get { ... } }
}

[Serializable]
public class SECURITY_CL_Ref : Eos.Delo.Common.EntityBaseRef
{
    public SECURITY_CL_Ref(int SECURLEVEL) { ... }
    public int SECURLEVEL { get { ... } }
}

[Serializable]
public class SECURITY_CL_Entity : ClassifEntity
{
    ...
    public override int ISN_LCLASSIF { get { ... } }
    public override string CLASSIF_NAME { get { ... } }
    public int SECURLEVEL { get { ... } }
    public String GRIF_NAME { get { ... } }
    ...
}
}

```

Класс DOC_RC_Entity является сущностью, соответствующей таблице DOC_RC. Атрибутами этого объекта являются, например, свойства DOC_DATE и FREE_NUM – соответствующие полям таблицы DOC_RC. Сущность DOC_RC_Entity является головной сущностью агрегатного объекта "ПК документа". Объект является агрегатным, т.к. сущность имеет дочерние сущности, хранящиеся в атрибутах типа "Коллекция сущностей", таких как DOC_SIGN_List, FORWARD_List, и прочие.

Класс DOC_RC_Ref – это ссылка на сущность, в данном случае ссылка имеет единственный атрибут – ISN_DOC, что соответствует первичному ключу таблицы DOC_RC.

Класс FORWARD_Entity является дочерней сущностью "Журнал пересылки документа", входящей в состав агрегатного объекта "ПК документа". Эта сущность, соответствует таблице FORWARD.

Класс SECURITY_CL_Entity сущность, соответствующая таблице SECURITY_CL. Эта сущность является простым объектом "Гриф доступа", т.к. не имеет в своем составе списков дочерних сущностей. В отличие от сущностей рассмотренных ранее, данная сущность наследуется от класса ClassifEntity, и имеет в своем составе атрибуты, общие для всех справочников системы – такие как ISN_LCLASSIF и CLASSIF_NAME³.

Полный перечень определенных в API сущностей системы приведен в разделе **"Ошибка! Источник ссылки не найден."**

1.8. Разработка процесса

Взаимодействие с системой Дело из процессов осуществляется с помощью следующих интерфейсов: IDeloApi, IDeloEvents, IDeloService. Эти интерфейсы, их методы и используемые в них вспомогательные классы, описаны полностью в разделе **"Ошибка! Источник ссылки не найден."**

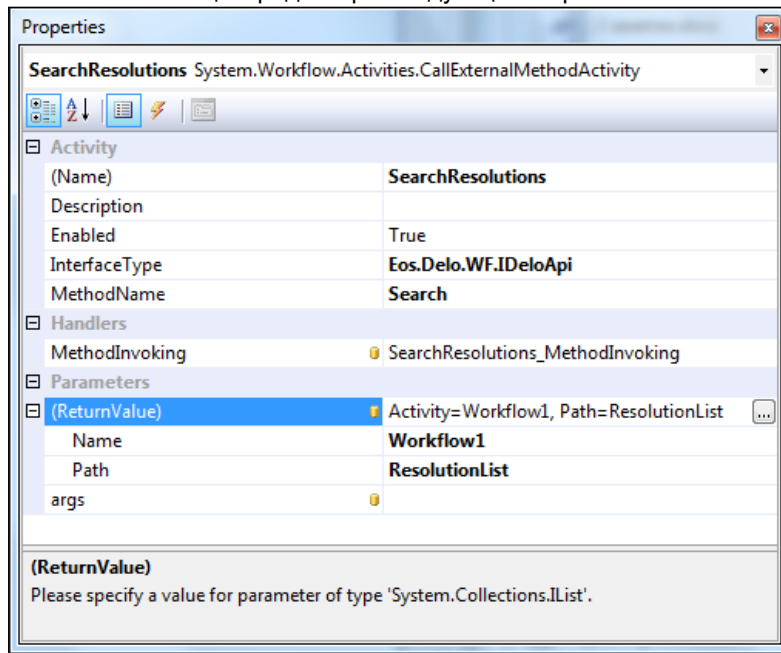
1.8.1. Поиск объектов

Для того чтобы выполнить поиск объектов нужно сначала объявить в процессе свойство типа "список сущностей" для хранения найденных в результате выполнения поиска объектов. Например, чтобы искать поручения, нужно объявить в процессе свойство:

```
public List<RESOLUTION_Entity> ResolutionList;
```

³ Для данного справочника, эти атрибуты будут иметь то же значение что и атрибуты SECURLEVEL и GRIF_NAME (и одноименные поля таблицы SECURITY_CL), соответственно

После этого нужно добавить в процесс активность CallExternalMethodActivity и заполнить ее свойства с помощью редактора следующим образом:



Как видно, активность при выполнении должна будет вызвать метод Search интерфейса IDeloApi, результат выполнения поиска будет помещен в определенную ранее переменную ResolutionList.

Также создан обработчик события MethodInvoking, который будет вызван непосредственно перед вызовом метода IDeloApi.Search – в этом обработчике нужно задать критерии, по которым будет осуществляться поиск поручений:

```
private void SearchResolutions_MethodInvoking(object sender, EventArgs e)
{
    var args = new SearchArguments();
    args.Result = "RESOLUTION";
    args.AppendCriterion("Res.Rc.ISN_DOC", RcEntity.ISN_DOC.ToString());
    ((CallExternalMethodActivity)sender).ParameterBindings["args"].Value = args;
}
```

Критерии поиска задаются в виде экземпляра класса SearchArguments. Обязательно должно быть заполнено свойство Result данного класса, задающее объект, который нужно искать. Значение, задаваемое в этом свойстве должно быть строкой, соответствующей имени искомой сущности без суффикса "_Entity". В качестве результатов поиска можно задавать только сущности, соответствующие объектам системы Дело, т.е. нельзя указывать дочерние сущности, входящие в состав агрегатных объектов, такие как FORWARD, REF_LINK и т.п.

После задания типа результата поиска, нужно задать критерии поиска. В примере выше задан поиск всех поручений относящихся к РК с заданным системным кодом.

Процессный API, поддерживает критерии аналогичные используемым в AcitveX API для поиска объектов и записей справочников, которые описаны в разделах **"Ошибка! Источник ссылки не найден."** и **"Ошибка! Источник ссылки не найден."**. Однако, соответствие не полное.

Во-первых, при поиске поручений, все критерии должны иметь префикс "Res.". Следующие два критерия задают поиск резолюции, дата которой должна находиться в заданном диапазоне, и РК, которой она принадлежит должна иметь группу документов с заданным кодом:

```
args.AppendCriterion("Res.Resolution.ResDate", "01/01/1998:31/01/1998");
```

```
args.AppendCriterion("Res.Rc.DocGroup", "0.2TZ.2U1.");
```

Также, есть исключения, перечень которых приведен в разделе "**Ошибка! Источник ссылки не найден.**". Например, в ActiveX API для поиска по системному номеру РК нужно задать критерий "ISN", а в процессном API критерий называется "ISN_DOC" – по названию поля в таблице DOC_RC. Поэтому в примере поиска поручений критерий выглядит как "Res.Rc.ISN_DOC", а не "Res.Rc.ISN".

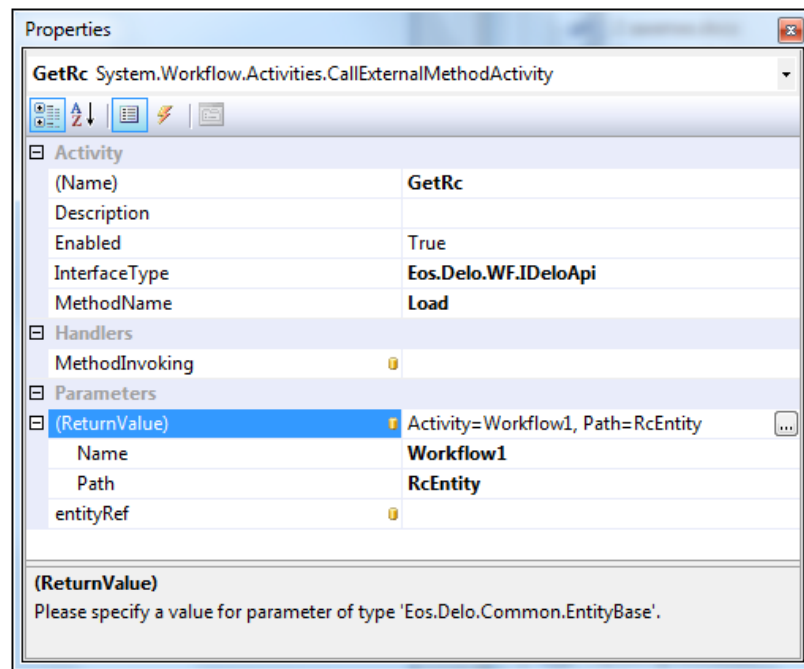
1.8.2. Загрузка объекта по идентификатору

Часто при разработке процессов необходимо получить (загрузить) данные единственного экземпляра объекта по известному системному коду. Частным случаем этого является получение данных объекта, для обработки которого был создан процесс – рабочий объект.

Для того чтобы загрузить данные рабочего объекта нужно сначала объявить в процессе свойство имеющее тип корневой сущности загружаемого объекта:

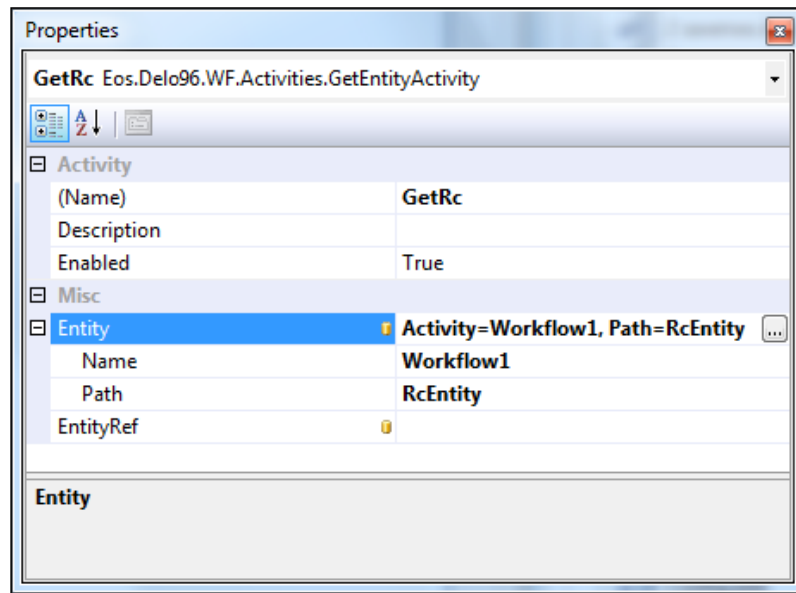
```
public DOC_RC_Entity RcEntity;
```

После этого нужно добавить в процесс активность CallExternalMethodActivity и заполнить ее свойства с помощью редактора следующим образом:



Активность при выполнении будет вызывать метод IDeloApi.Load, возвращенные методом данные объекта, будут помещены в переменную RcEntity.

То же самое, но с немного меньшими усилиями можно выполнить, используя активность Eos.Delo96.WF.Activities.GetEntityActivity:

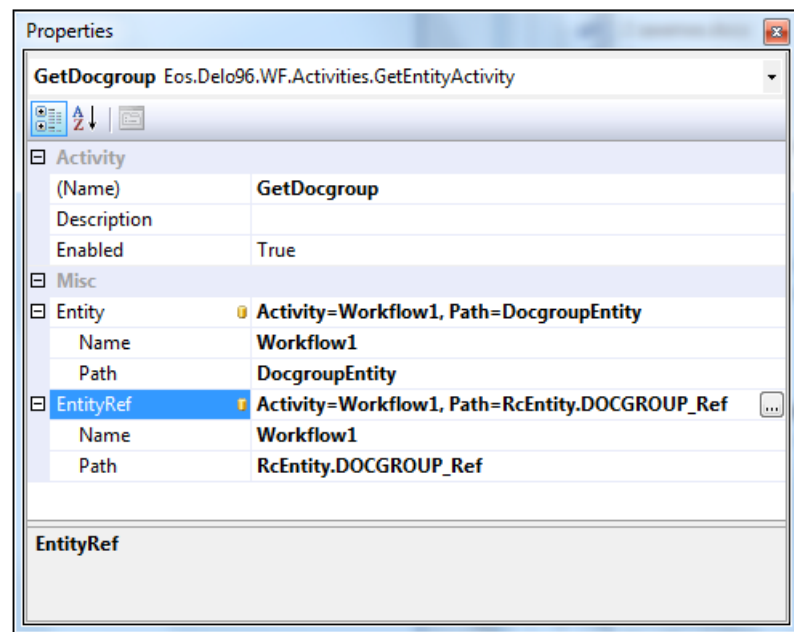


При использовании данной активности не требуется искать используемый интерфейс и метод, что немного ускоряет разработку.

Для того чтобы получить загрузить объект по известному коду, необходимо создать из кода ссылку на сущность – объект одного из типов <ИМЯ_СУЩНОСТИ>_Ref. В большинстве случаев нужная ссылка доступна как свойство уже загруженной ранее сущности. Например, если загружена РК документа, как в предыдущем примере, и требуется проанализировать ее группу документов, нужно объявить переменную для хранения сущности DOCGROUP_CL_Entity

```
public DOCGROUP_CL_Entity DocgroupEntity;
```

и сконфигурировать активность GetEntityActivity следующим образом:



В отличие от предыдущего примера, теперь свойство EntityRef заполнено, и при вызове метода IDeloApi.Load, в параметр entityRef будет передана ссылка на искомую запись справочника групп документов.

1.8.3. Модификация данных объекта

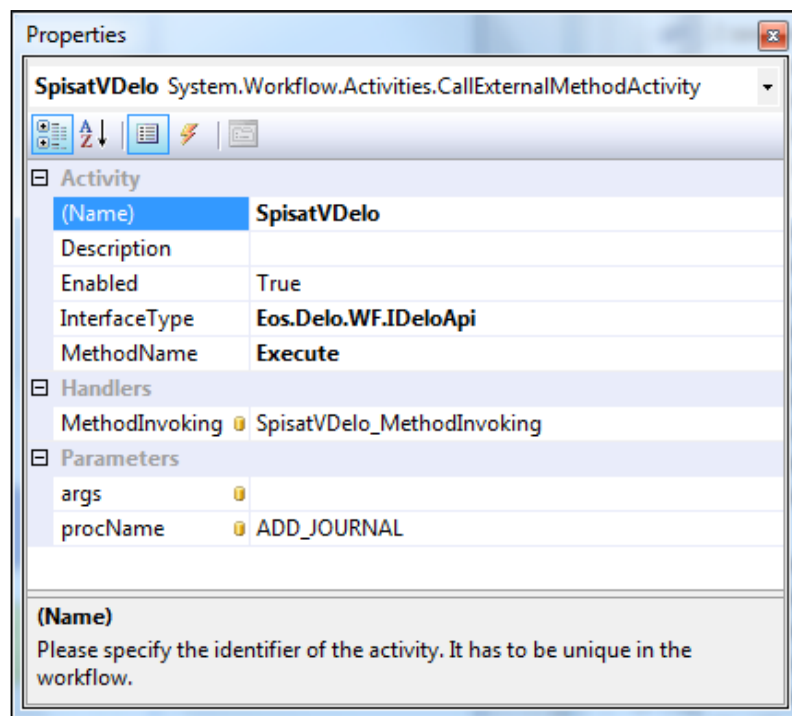
Изменения в БД вносятся с помощью модификаторов данных – хранимых процедур, предоставляющих возможность добавлять, модифицировать и удалять сущности в базе данных. После выполнения модификатора, данные, хранящиеся в памяти приложения, становятся устаревшими, и не соответствуют данным в БД – их необходимо перечитывать.

Ряд модификаторов (в основном модификаторы добавления) позволяют за один вызов выполнить добавление (или изменение) нескольких сущностей.

Для использования модификаторов соединение с БД обязательно должно выполняться с заданием текущего пользователя. То есть, все изменения в БД вносятся от имени какого-либо пользователя.

Полный список модификаторов приведен в **Ошибка! Источник ссылки не найден.**

Для того чтобы вызвать модификатор из процесса нужно добавить в процесс активность CallExternalMethodActivity и заполнить ее свойства с помощью редактора следующим образом:



Активность при выполнении будет вызывать метод IDeloApi.Execute, и выполнять процедуру ADD_JOURNAL, для списания документа в дело. Параметры процедуры задаются в обработке события MethodInvoking:

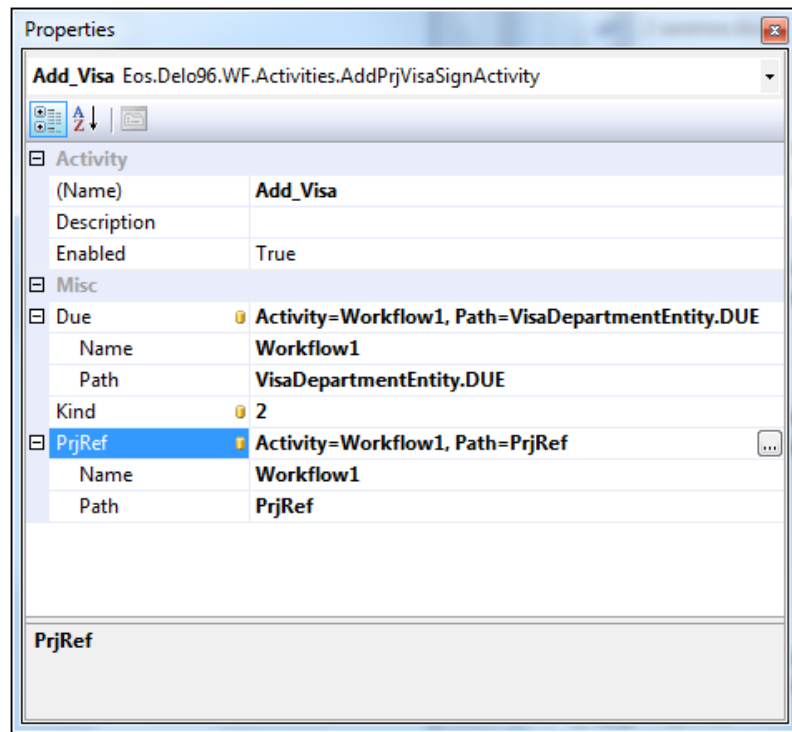
```
private void SpisatVDelo_MethodInvoking(object sender, EventArgs e)
{
    var args = new CmdArguments();
    args.Add("aIsnDoc", RcEntity.ISN_DOC);
    args.Add("aCard", RcEntity.DUE_CARD_REG);
    args.Add("aCab", 0);
    args.Add("aCodes", null);
    args.Add("aIsnNomencl", 3526);
    args.Add("aDates", DateTime.Today.ToString("yyyyMMdd HH:mm:ss"));
}
```

```

args.Add("aFlagsOriginal", 1);
args.Add("aOrigNums", RcEntity.SPECIMEN);
args.Add("aCopyNums", null);
args.Add("aClearFldr6", 1);
args.Add("aDescructFlag", null);
((CallExternalMethodActivity)sender)
    .ParameterBindings["args"].Value = args;
}

```

Для некоторых модификаторов в библиотеке Eos.Delo96.WF.Activities есть активности, упрощающие разработку процессов. Например, чтобы добавить визирующее должностное лицо в РКПД, можно в процессе использовать активность AddPrjVisaSignActivity:



При этом все параметры модификатора задаются в дизайнера, без написания кода.

1.8.4. Подписка и реагирование на события

События позволяют процессам не занимать системные ресурсы до тех пор, пока в этом не возникнет надобности. Например, процесс, назначением которого является маршрутизация РКПД, может быть выгружен из памяти подсистемы, до тех пор, пока в пользователи не работают с этой РКПД. Такой процесс может подписаться на события модификации одной этой РКПД, и в этом случае система будет иметь возможность выгрузить его из памяти, а при получении события восстановить состояние процесса и продолжить его выполнение.

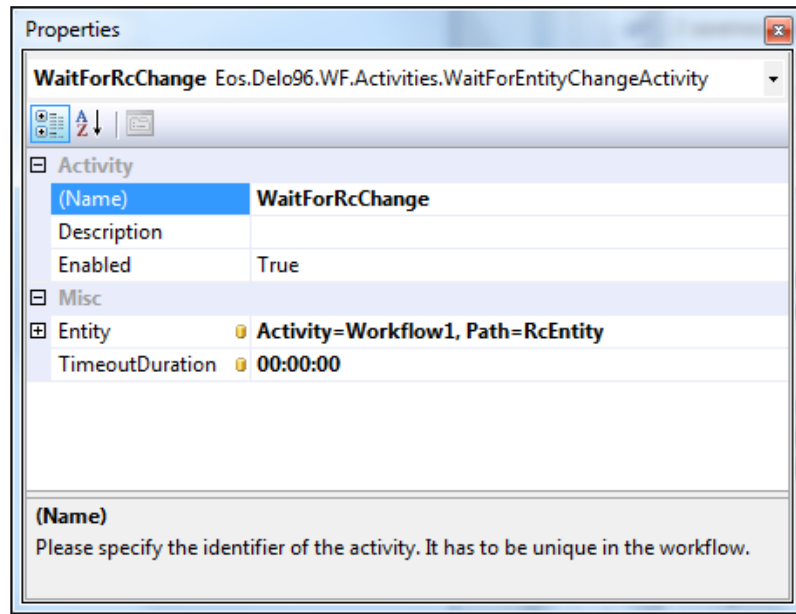
Взаимодействие с механизмом событий системы Дело обеспечивается методами интерфейса IDeloEvents. Порядок использования этого интерфейса должен быть следующий:

1. Подписка на события с помощью вызова одного из методов `Subscribe` или `SubscribeForEvent`, с передачей идентификатора подписки, служащего идентификатором корреляции⁴.
2. Ожидание срабатывания события `ObjectChanged`.
3. Когда событие получено, необходимо подтвердить его получение с помощью вызова `Confirm`.
4. Если необходимости отслеживания событий больше нет, отписаться с помощью вызова метода `Unsubscribe`.

Если между моментом отправки сообщения в процесс и подтверждением получения сообщения произошел сбой, повлекший за собой перезагрузку среды выполнения процессов, то уведомление будет послано повторно. Таким образом, гарантируется доставка уведомлений обо всех событиях в процесс, однако, возможна ситуация, когда уведомление об одном событии будет доставлено в процесс дважды.

Чаще всего в процессе требуется отслеживание состояния данных объекта, для обработки которого был создан процесс – подключенного объекта. В библиотеке `Eos.Delo96.WF.Activities` есть активность `WaitForEntityChangeActivity`, которая полностью реализует описанный выше алгоритм, позволяя легко отслеживать состояние рабочего объекта.

При использовании этой активности нужно создать в процессе свойство, которое содержит данные отслеживаемого объекта, и связать это свойство в редакторе со свойством `Entity` активности.



Также можно указать время, в течение которого ожидать изменений в данных объекта, значение `00:00:00`, как в примере, означает, что ждать нужно бесконечно.

Подробно механизм событий описан в разделе "**Ошибка! Источник ссылки не найден.**".
Расширенные примеры работы с событиями можно найти в учебном проекте `DeloWFTTest.sln`.

1.8.5. Использование активностей `GetEmailActivity` и `SendEmailActivity`

Общий принцип работы данных активностей заключается в том, что они используют файловую систему для хранения сообщения электронной почты в стандартном формате `.eml`.

⁴ Использование корреляции событий WF позволяет в одном процессе подписаться на события нескольких объектов. Без этого, при возникновении события `ObjectChanged`, система не будет знать, в какую активность `HandleExternalEventsActivity` нужно передать данные, если их несколько.

Рассмотрим подробнее отправку сообщения с помощью активности `SendEmailActivity`. Выполнение этой задачи состоит из следующих шагов:

1. Сформировать сообщение (тема, тело письма, адресат, вложения и пр.) и сохранить его в файловую систему в стандартном формате `.eml`. Для этого можно использовать любые сторонние компоненты, например, стандартный класс `.NET System.Net.Mail.SmtpClient`, настроенный на сохранение в каталог файловой системы: `DeliveryMethod = SpecifiedPickupDirectory`, `PickupDirectoryLocation = <путь к временному каталогу>`.

2. Использовать активность `SendEmailActivity` со следующими параметрами:

- `AttemptsCount = 1`

Количество попыток отправки, которое необходимо предпринять в случае ошибки, прежде чем процесс перейдет в состоянии "приостановлен", либо будет осуществлен выход с сообщением об ошибке (зависит от значения параметра `SuspendOnError`)

- `AuthMethod = NONE`

Метод авторизации на smtp сервере. Возможные значения: пустая строка, `NONE`, `NTLM`, `LOGIN`. В случае `NTLM` и `LOGIN` необходимо определить значения параметров `UserName` и `Password`

- `EmailFileSourcePath = <путь к временному файлу>`

Путь к файлу `.eml`, который был сформирован на шаге №1

- `Host = smtpserver01`

Адрес smtp сервера

- `Port = 25`

Порт smtp сервера

- `SendEncrypted = false`
- `SendSigned = false`
- `SuspendOnError = true`

Этот параметр определяет поведение активности в случае возникновения ошибки при попытке отправки письма. Возможны следующие варианты: немедленный выход из активности либо приостановка процесса. В случае выхода, сообщение об ошибке будет сохранено в свойство `LastErrorText`. В случае приостановки, процесс может быть возобновлен технологом. При этом отработает событие `OnResumeAfterError`, на которое можно подписаться и добавить дополнительную логику, например, перечитать параметры сервера отправки из конфигурации процесса на случай, если они были изменены.

Прием сообщения с помощью активности `GetEmailActivity` аналогичен, с разницей в том, что в параметрах передаются настройки подключения к pop3 серверу, в `EmailStorageDir` путь к папке, куда будет сохранено email сообщение в формате `.eml` и выходной параметр `CurrentEmailSourcePath` содержит путь к сформированному файлу `.eml`

1.9. Подготовка процесса к выполнению

1.9.1. Описание метаданных процесса

Метаданные представляют собой xml-документ, который хранится в виде ресурса в сборке, в которой определен процесс. Для того, чтобы подсистема распознала ресурс как метаданные процесса, ресурс должен иметь название <название_типа_процесса>.Meta.xml. Например, если тип процесса называется DeloWFTest.AutostartTest, то ресурс с метаданными должен быть назван DeloWFTest.AutostartTest.Meta.xml.

Метаданные процессов описывают ряд аспектов взаимодействия процессов и подсистемы:

- Допустимые для процесса типы объектов – используется для ограничения списка типов объектов при создании конфигурационной ассоциации;
- Допустимость одновременной работы нескольких экземпляров процесса – используется при определении возможности запуска на выполнение созданных процессов;
- Значение по умолчанию для признака удаления процесса после завершения – используется для установки признака удаления в момент запуска экземпляра на выполнение;
- Форма редактирования параметров запуска процесса – используется при запуске процесса на выполнение вручную;
- Описание процесса – используется как начальное значение поля "Описание процесса".

Схема файла метаданных следующая:

```
<?xml version="1.0" encoding="utf-8" ?>
<Meta>
  <!--Описание процесса -->
  <Description></Description>
  <!--Значение по умолчанию отсутствует-->

  <!--Возможность автоматического удаления процесса после успешного завершения-->
  <DeleteCompleted>0</DeleteCompleted>
  <!--0 - Не удалять, по умолчанию; 1 - Удалять;-->

  <!--Возможность декларативного ограничения количества одновременно выполняющихся процессов-->
  <InstanceLimitType>0</InstanceLimitType>
  <!--0 - Без ограничений, по умолчанию;-->
  <!--1 - Единственный выполняющийся экземпляр в системе;-->
  <!--2 - Единственный выполняющийся экземпляр для типа объектов;-->
  <!--3 - Единственный выполняющийся экземпляр для экземпляра объекта;-->

  <!--Название ресурса содержащего форму редактирования параметров запуска процесса-->
  <StartForm>LibraryX.EditParmsForm1.html</StartForm>
  <!--Полный строковый путь до ресурса-->
```

```

<!--По умолчанию - форма отсутствует-->

<!--Допустимые для процесса типы объекта-->
<KindObject>DOC_RC;PRJ_RC</KindObject>

<!--Список идентификаторов из таблицы EOS_OBJECT через разделитель, или-->
<!--None - процесс не предназначен для работы с каким-либо объектом-->
<!--По умолчанию - допустим любой объект-->

</Meta>

```

1.9.2. Подготовка конфигурационного файла процесса

Конфигурационный файл – это xml документ, который загружается при создании конфигурационной ассоциации.

В конфигурационном файле можно определять произвольные параметры, необходимые процессу. Для доступа к параметрам, определенным в конфигурационном файле из процесса служат методы интерфейса IDeloService.GetConfig и IDeloService.GetConfigString. Параметры могут быть оформлены в произвольном виде, но должны соблюдаться следующие условия: конфигурационный файл должен быть правильным xml-документом, и корневым элементом должен быть элемент Configuration.

Также в конфигурационном файле могут быть заданы условия для автоматического запуска процессов с помощью предопределенных структур:

- Элементы StartForDocgroup и Start/OnEvent позволяют запускать процессы, предназначенные для обслуживания объектов системы Дело (РК, РКПД, поручения, справочники). Процессы могут быть запущены службой автоматически при возникновении в системе событий создания либо редактирования этих объектов.
- Элементы Start/Job позволяют запускать процессы, не предназначенные для работы с конкретными объектами. Такие процессы могут запускаться службой автоматически без дополнительных условий – они являются, по сути, фоновыми заданиями (job).

Далее описаны примеры использования конфигурационного файла для задания автоматического запуска процесса.

Секция **StartForDocgroup** позволяет запустить процесс при создании РК, РКПД или поручения определенной группы документов. При этом можно задавать как лист из справочника "Группы документов", так и вершину. Если указана вершина, то процесс будет автоматически запускаться для всех объектов, группа документов которых является дочерней по отношению к заданной вершине.

Перечень групп документов задается дочерними элементами Docgroup, как показано в примере:

```

<?xml version="1.0" encoding="utf-8" ?>
<Configuration>
  <StartForDocgroup>
    <Docgroup>0.2UB.2UN.- Служебные записки</Docgroup>
    <Docgroup>0.2UB.2UD.- Организационно-распорядительные документы</Docgroup>
  </StartForDocgroup>
</Configuration>

```

Значение элемента Docgroup имеет формат: <due>[-<comment>], где

- **due** – значение поля due из таблицы docgroup_cl,
- **comment** - необязательный комментарий, обычно значение поля classif_name соответствующей записи таблицы docgroup_cl.

Примечание: Конфигурирование автозапуска с помощью секции StartForDocgroup не рекомендуется использовать, т.к. он является режимом совместимости с версией Дело 8.9.9.

Секция **Start/OnEvent** позволяет настроить автоматический запуск процесса, как при возникновении события создания, так и при модификации любого объекта системы Дело. Элемент OnEvent определяет условия фильтрации событий, в случае возникновения которых нужно запускать процесс на выполнение. Условия задаются в атрибутах элемента:

- Тип объекта – **kind_object**. Название "головной" таблицы объекта системы Дело. Обязательный критерий.
- Идентификатор объекта – **object_id**. Не допускается одновременно с критерием due_docgroup.
- Группа документов – **due_docgroup**. Вершины при проверке разворачиваются в список лежащих под ними листов. Событие соответствует данному критерию, если значение в поле evnt_queue_item.due_docgroup входит в заданный список. Не допускается одновременно с критерием object_id.
- Флаг операции – **flag**. Строковый идентификатор, например: "forward", "calcsend". Событие соответствует данному критерию, если значение входит во множество флагов в поле evnt_queue_item.flags.
- Тип события – **kind_event**. 1 – создание объекта, 2 – модификация объекта.

Подробно механизм событий описан в разделе "**Ошибка! Источник ссылки не найден.**".

Пример конфигурации:

```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration>
  <Start>
    <!-- При редактировании РК с изменением раздела "Адресаты" -->
    <OnEvent kind_event="2" kind_object="DOC_RC" due_docgroup="0." flag="CALCSEND" />
  </Start>
</Configuration>
```

Если за период опроса службой произошло несколько событий удовлетворяющих условиям, заданным в элементах Start/OnEvent, то будет создан только один новый экземпляр процесса.

Секция **Start/Job** позволяет запускать фоновые процессы в системе для выполнения различных задач, не связанных однозначно ни с каким конкретным объектом системы Дело. Если в конфигурационном файле процесса есть элементы Start/Job, подсистема запускает по одному экземпляру процесса для каждого из элементов.

В случае если процесс прекратит работу, например, в результате сбоя, подсистема запустит новый экземпляр, выступая в роли "защитника". Таким образом, для каждого элемента Start/Job в конфигурационных файлах процессов гарантируется наличие одного активного экземпляра процесса.

Пример конфигурации:

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<Configuration>
  <Start>
    <Job id="1" />
    <Job id="2" />
  </Start>
</Configuration>
```

В элементе Job обязательно должен быть указан атрибут id, значение которого передается в процесс как параметр запуска в следующем виде:

```
<?xml version="1.0" encoding="utf-8" ?>
<params><id>...</id></params>
```

где вместо многоточия будет записано значение атрибута Configuration/Start/Job/@id из конфигурационного файла. Во время выполнения процесс может получить эти данные с помощью обращения к методу IDeloService.GetParams. Далее, по значению параметра id процесс может, например, считывать разные конфигурационные параметры:

```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration>
  <Start>
    <Job id="1" />
    <Job id="2" />
  </Start>
  <WatchFolder jobId="1" path="C:\IncomingMessages\FromBranch1" interval="10"/>
  <WatchFolder jobId="2" path="C:\IncomingMessages\FromBranch2" interval="50"/>
</Configuration>
```

1.9.3. Загрузка процесса в систему и создание конфигурационной ассоциации

Подсистема выполняет процессы, исполняемый код которых находится в сборках .NET. Соответственно, для того, чтобы процесс мог быть запущен на выполнение, его сборка должна быть загружена в память службы подсистемы. Для обеспечения доступности сборок с процессами на всех службах, подключенных к конкретной БД Дело, сборки процессов сохраняются в БД, и, при создании среды исполнения процессов, каждая служба загружает сборки из БД в собственную память. В момент сохранения в базу данных сборка анализируется на предмет наличия в ней процессов, и, в случае их наличия, анализируются метаданные этих процессов.

Подсистема позволяет загружать сборки, не содержащие процессы, являющиеся библиотеками, совместно используемыми в нескольких других сборках. Это позволяет разработать несколько библиотек с процессами, основанных на общей библиотеке активностей.

Загрузка сборок (библиотек) процессов осуществляется через WEB-интерфейс подсистемы. Подробно данная функция описана в "Руководстве технолога".

После загрузки библиотеки с процессами в БД, процессы из этой библиотеки становятся доступными для конфигурирования параметров их запуска – для создания конфигурационных ассоциаций. Без наличия конфигурационной ассоциации ни один экземпляр процесса не может быть запущен.

В конфигурационной ассоциации указываются следующие параметры запуска процессов:

-
- На какой из служб, подключенных к БД, должны запускаться экземпляры данного процесса;
 - Для обработки какого типа объектов БД данный процесс должен запускаться;
 - Описания условий автоматического запуска экземпляров данного процесса (задаются в конфигурационном файле в секциях StartForDocgroup и Start);
 - Признак разрешения запуска процессов вручную пользователями через приложения системы Дело;