Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

Д. С. КУЗЬМЕНКОВ, А. А. РОДИОНОВ

LOTUS DOMINO/NOTES

ПРАКТИЧЕСКОЕ РУКОВОДСТВО

по выполнению лабораторных работ

В 2-х частях
Часть 2

Гомель
УО «ГГУ им. Ф. Скорины»
2011

Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

Д. С. КУЗЬМЕНКОВ, А. А. РОДИОНОВ

LOTUS DOMINO/NOTES

ПРАКТИЧЕСКОЕ РУКОВОДСТВО

по выполнению лабораторных работ

для студентов математического факультета специальностей
1-31 03 03-02 «Прикладная математика (научно-педагогическая деятельность)»
1-31 03 06-01 «Экономическая кибернетика (математические методы
и компьютерное моделирование в экономике)»
1-40 01 01 «Программное обеспечение информационных технологий»

В 2-х частях
Часть 2

Гомель
УО «ГГУ им. Ф. Скорины»
2011

УДК 004.43 (075.8)
ББК 32.973–018.1я73
К 893

Рецензенты:
А.И. Рябченко – заведующий кафедрой информационных технологий учреждения образования «Гомельский государственный технический университет им. П.О. Сухого», кандидат физико-математических наук;
кафедра вычислительной математики и программирования учреждения образования «Гомельский государственный университет им. Ф. Скорины»

Рекомендовано к изданию научно-методическим советом учреждения образования «Гомельский государственный университет им. Ф. Скорины»

Кузьменков, Д. С.
К 893 Lotus Domino/Notes: практическое руководство для студентов математического факультета специальностей 1-31 03 03-02 «Прикладная математика (научно-педагогическая деятельность)», 1-31 03 06-01 «Экономическая кибернетика (математические методы и компьютерное моделирование в экономике)», 1-40 01 01 «Программное обеспечение информационных технологий»: в 2 ч. Ч.2 / Д. С. Кузьменков, А. А. Родионов; М-во образования РБ, Гомельский государственный университет им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2011. –48 с.

Во второй части практического руководства изложены основные вопросы, касающиеся проектирования приложений в среде Lotus Domino/ Notes, описан синтаксис, основные функции и классы языка Lotus Script, рассмотрена работа с ответными документами, подчиненными формами, общими полями и программами-агентами, подробно описаны вопросы, касающиеся обеспечения безопасности в среде Lotus Domino/ Notes, приведены алгоритмы решения наиболее типовых практических задач. Руководство адресовано студентам математического факультета и призвано оказать помощь студентам в овладении и закреплении базовых знаний и умений в проектировании приложений в среде Lotus Domino/ Notes.

УДК 004.43 (075.8)
ББК 32.973–018.1я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Вторая часть руководства призвана помочь студентам овладеть основами современных компьютерных технологий и программного обеспечения, проектирования и разработки приложений в среде Lotus, приобрести навыки работы с одной из самых современных компьютерных систем управления документооборотом.

Практическое руководство составлено в соответствии с учебными программами спецкурса «Lotus Domino/Notes» для студентов 4 курса специальности 1-31 03 03-02 «Прикладная математика (научно-педагогическая деятельность)», 4 курса специальности 1-31 03 06 01 «Экономическая кибернетика (математические методы и компьютерное моделирование в экономике)», 3 курса специальности 1-40 01 01 «Программное обеспечение информационных технологий» специализации «Компьютерные системы и Internet технологии», утвержденными научно-методическим Советом Учреждения образования «Гомельский государственный университет».

Вторая часть практического руководства структурно содержит четыре темы. В первой теме приводится описание объектно-ориентированного языка программирования Lotus Script. Вторая тема посвящена работе с документами-ответами, подчиненными формами и общими полями. В третьей теме описывается работа с программами-агентами, приведен типичный пример программы-агента, написанной на языке Lotus Script. Последняя тема посвящена организации безопасности в среде Lotus Domino\ Notes. В этой теме рассмотрены общие вопросы организации безопасности в Notes, уровни доступа пользователей к базе данных, ограничения доступа к формам и представлениям с использованием ролей, скрытие различных объектов в зависимости от роли. В конце каждой темы приводится список вопросов для самоконтроля и задание, направленное на закрепление приведенной в теме информации. В трех последних темах, содержащих наиболее сложные задания, приведены также алгоритмы выполнения заданий.

Практическое руководство по курсу «Lotus Domino/ Notes» направлено на формирование умений и навыков в проектировании приложений в среде Lotus Domino/ Notes, на усвоение современной компьютерной технологии разработки информационных систем.

Практическое руководство может быть использовано преподавателями при проведении практических занятий и студентами в их самостоятельной работе над предметом.

Тема 1 ЯЗЫК LOTUS SCRIPT

Lotus Script – это объектно-ориентированный язык программирования, совместимый с языком *Basic*. Используя язык Lotus Script можно разработать повторно используемые программы, которые могут совместно использоваться многими объектами, приложениями и разработчиками, разработчик может получать доступ к объектам внутри приложения такими методами, которые невозможны в языке формул.

Преимуществами языка Lotus Script являются:

– модульность приложения;
– наличие циклов и ветвлений;
– лучшая обработка ошибок;
– возможность отладки (*File ->Tools -> Debug Lotus Script*);
– возможность работы с внешними по отношению к Notes файлами;
– доступ к большинству скрытых элементов Lotus.

Lotus Script используется для написания кода, который можно применять к ряду объектов внутри приложения Notes (к действиям, кнопкам, событиям поля). Lotus Script также можно применять в программах-агентах или на уровне формы.

Сценарий – ряд операторов или выражений на языке Lotus Script, которые выполняют требуемые действия.

Когда происходит некоторое событие (например, создание нового документа, установка указателя мыши на поле или выход из поля), то выполняется сценарий, связанный с данным событием.

1.1 Синтаксис языка

Язык Lotus Script состоит из следующих элементов: идентификаторы, метки, операторы, ключевые слова, константы, литералы, переменные.

1.1.1 Данные

Идентификаторы – это имена, которые даются константам, переменным, подпрограммам, типам, классам и функциям. Идентификаторы в Lotus Script **нечувствительны к регистру**. 1-ый символ идентификатора должен быть буквой. Длина идентификатора не может превышать 40 символов + суффикс типа, если он есть (см. таблицу 1.1). Все остальные символы, кроме 1-го могут быть буквами, числами, символами подчеркивания или символами, коды ANSI кото-

рых >127 (это символы, которые можно ввести с клавиатуры с помощью сочетания клавиш alt+ код ANSI).

Таблица 1.1 – Суффиксы типа

Суффикс	**Тип данных**
%	Integer
&	Long
!	Single
#	Double
@	Currency
$	String

Названия внутренних полей в Lotus Notes могут начинаться со знака $, например $FIO. Чтобы использовать их в Lotus Script, перед ними необходимо поставит тильду (~).

Метка также является идентификатором, поэтому она подчиняется тем же правилам, что и для построения идентификаторов (метки не описываются). После метки ставится двоеточие, после которого на той же строке может расположиться оператор. Когда происходит переход на метку с помощью оператора **GoTo**, то последующий возврат на исходное место программы не выполняется. Оператор **GoSub** отличается от **GoTo** тем, что управление возвращается на то место, откуда произошел переход.

Ключевые слова – это зарезервированные слова языка Lotus Script, которые относятся к встроенным функциям, таким как *Trim$* и *Open$*. Значение ключевых слов установлено заранее и не может быть изменено программистом. Все ключевые слова отображаются голубым цветом.

Существует 3 способа определить константу в Lotus Script:

1. Существует несколько встроенных (зарезервированных) констант: **Empty**, **Nothing**, **Null**, **Pi**, **True**, **False**;
2. Имеются файлы с предопределенными константами, например, *LSConst.LSS*. Разработчик может включать файлы с расширением **.***LSS* в свои приложения;
3. Можно определить свои собственные константы, используя ключевое слово **Const**.

Пример. В событии *Options* некоторого объекта (например, поля) запишем: **use** “LsConst.lss”.

Пример. Опишем несколько пользовательских констант:

Const x = 123.45, MyCur@=2740.5, str$=”Lotus”

Литерал – элемент данных, значение которого не изменяется, но в отличие от констант он не имеет имени, по которому на него можно ссылаться. Литералы могут быть строковыми величинами или числами.

Пример. Dim MyVar **as String**

MyVar=”Opel”

После выполнения переменная MyVar содержит текстовую строку (литерал) “Opel”

Простые типы данных (данные таких типов представляют собой одно значение) языка Lotus Script и диапазоны значений для переменных различных типов приведены в таблице 1.2.

Таблица 1.2 – Простые типы данных

Тип данных	**Диапазон значений**
Boolean	0 (False) or -1 (True)
Byte (короткое целое без знака)	от 0 до 255
Integer (короткое целое число со знаком)	от -32,768 до 32,767
Long (длинное целое число со знаком)	от -2,147,483,648 до 2,147,483,647
Single (число с плавающей точкой обычной точности)	от -3.402823E+38 до 3.402823E+38
Double (число с плавающей точкой удвоенной точности)	от -1.7976931348623158E+308 до 1.7976931348623158E+308
Currency (число с фиксированной точкой и 4 знаками после запятой)	от -922,337,203,685,477.5807 до 922,337,203,685,477.5807
String (строка)	от 0 до 32КБ символов

Переменные могут использоваться без предварительного описания. Тогда такая переменная будет иметь тип ***Variant*** и содержать тот тип данных, которые вы в нее внесете. Но в сложных программах при отсутствии описания переменных очень трудно отыскать ошибку. Хорошим стилем программирования является явное описание переменных. Сложные типы данных языка Lotus Script приведены в таблице 1.3.

Для объявления переменных какого-нибудь типа используется конструкция:

Dim имя переменной as имя типа.

Пример. Dim S1,S2 **as String**, I **as integer**

Таблица 1.3 – Сложные типы данных

Тип данных	**Описание**
Array (массив)	Именованный набор элементов одного типа данных. Диапазон индексов от -32768 до 32767;
List (список)	Одномерный массив элементов одного типа. Доступ к элементам осуществляется не по индексам, а по уникальным именам (дескрипторам);
Variant (вариант)	Переменная с необъявленным типом. Может содержать данные любого другого типа;
User-defined data type (определяемый пользователем тип данных)	Набор любого числа переменных любого типа, определяемый как единое целое;
User-defined class (определяемый пользователем класс)	Определяемый пользователем тип данных вместе со своими свойствами и методами;
Object reference (ссылка на объект)	Указатель на отдельный объект некоторого класса (или заголовок этого объекта).

Если требуется чтобы все переменные, начинающиеся с букв от I до N, были целочисленные, то можно воспользоваться оператором **Deftype**, где вместо слова *type* необходимо поставить сокращенное название типа данных из таблицы 1.4. Операторы **Deftype** можно использовать только на уровне модуля, они должны следовать раньше всех других операторов объявлений, за исключением объявлений констант. Если после оператора **Deftype** объявить переменную, вступающую в противоречие с оператором **Deftype**, то будет действовать явное объявление переменной, так как оно имеет более высокий приоритет по сравнению с оператором **Deftype**.

Таблица 1.4 – Сокращения, применяемые в операторе **DefType**

Тип данных	**Сокращение**
Currency	Cur
Double	Dbl
Integer	Int
Long	Lng
Single	Sng
String	Str
Variant	Var

Пример. Опишем переменные явно и неявно:

Def int I-N (все переменные с именами, начинающимися с букв от I до N будут иметь тип Integer по умолчанию)

Dim i1 (переменная типа Integer, это соответствует оператору Defint)

Dim N2 as double (явное объявление переменной типа Double)

Массив может быть многомерным, может иметь до 8 измерений.

Формат описания массива:

Dim имя переменной-массива (нижняя граница индекса **to** верхняя граница индекса) **as** тип данных

Пример. Dim Myarray (0 **to** 20) **as string**
Myarray(0)=”Lotus”

Можно массив описать так: **Dim** Myarray (20) **as string.** Тогда нумерация элементов массива начинается с нуля и заканчивается 19.

Массив может быть не только из простых типов, но и из сложных, например, массив объектов.

Массив может быть динамическим, тогда он изначально определяется имеющим некоторое количество элементов, а потом может быть переопределен.

Пример. Dim s (0 **to** 0) **as string** – задаем динамический массив
или **Dim** s () **as string**
Redim s (0 to 8) – переопределяем массив

При переопределении сохраняется тип массива. Если перед переопределением в массив уже были внесены значения, например, S(0)=”привет”, и их необходимо сохранить, то запишем:

Redim preserve s (0 to 8) **as string**

Пример. Dim A(2,2) – объявляем двумерный массив вариантного типа.
A(1,1)=1 – присваиваем значение элементу массива a_{11}.

Замечание. Для того, чтобы нумерация элементов в массиве начиналась не с нуля, а с единицы, к событию (*Options*) необходимо добавить оператор: **Option Base 1**.

Список – это одномерный массив элементов одного типа, которые вызываются не по индексу, а по уникальным именам.

Формат описания списка:

Dim sp **List as String**

Создание нового элемента списка или присвоение значения существующему элементу:

Sp(”jan”)=”Январь”

Для обращения к элементу списка и получения его значения запишем: Sp(”jan”).

Для проверки наличия элемента в списке можно использовать функцию **ISElement**.

Пример. If IsElement (Sp(”jan”)) then

…

end if

Чтобы по элементу списка получить его название, используется функция **listTag** (она работает только в цикле **forall**, который позволяет пробегать по всем элементам списка или массива).

Пример. Forall jan in sp
Name=ListTag(jan)
End forall

Для очистки списка или удаления конкретного элемента списка используется функция **Erase**, например, **Erase** sp() или **Erase** sp(“jan”).

Переменные типа *Variant* могут содержать данные любых типов, определенных в Lotus Script, за исключением типов, определяемых пользователем. Эти переменные также могут содержать булевские данные и данные даты/времени (такого типа в Lotus Script нет). Тип данных, хранимых в переменной типа Variant, может изменяться в зависимости от того, какие данные заносятся в эту переменную. Главное не забывать данные какого типа сейчас хранятся в переменной типа *Variant*, чтобы при работе с этой переменной не получить ошибку *Type Mismatch*.

Пример. Dim S (0 to 10) **as string**
S(0)=”Lotus”
S(1)=”Notes”
Tmp=s

Переменная tmp типа Variant (так как эту переменную использовали без предварительного описания) содержит массив.

1.1.2 Операторы

В Lotus Script существуют следующие виды операторов: комментарии, директивы компилятора, объявления, определения, блочные операторы, блочные операторы цикла, операторы ветвления и операторы прерывания. Операторы состоят из ключевых слов. На одной строке должен находиться только один оператор. Если требуется разместить оператор на нескольких строках (или из-за длины оператора, или из соображения эстетики), то последним символом в строке должен быть символ подчеркивания. Если на одной строке необходимо расположить несколько операторов, то они отделяются друг от друга двоеточием. Длинные текстовые материалы можно располагать на нескольких строках, заключая их в фигурные скобки, или между вертикальными линиями.

Если один из ограничителей строки («, |, }) присутствует внутри строки, то он должен быть воспроизведен дважды. Только открывающая фигурная скобка в строке заключенной в фигурные скобки может присутствовать в одном экземпляре.

Операторы бывают унарные (операция выполняется над одним операндом, например, **not**) и бинарные (операция выполняется над двумя операндами).

Основные арифметические, логические и строковые операторы и соответствующие им операции приведены в таблице 1.5.

С приоритетом операций в языке Lotus Script можно ознакомиться в [7, стр. 514, табл. 22.3].

Комментарии

В Lotus Script, в текст программы можно включать комментарии тремя способами:

1. С помощью оператора **REM**, но в отличие от языка формул, текст комментария в Lotus Script не заключают в кавычки.

2. С помощью « ’ » (апострофа) все, что после него – комментарии.

3. Применение директив компилятора *% REM*, *% END REM*. Все операторы между этими директивами считаются комментариями и игнорируются компилятором.

Директивы компилятора (*% REM*, *% Include*) могут присутствовать в сценарии как в разделе *Options,* так и в разделе *Declarations*.

Операторы управления ходом выполнения сценария

Они определяют порядок выполнения операторов сценария в зависимости от некоторых значений и условий.

Таблица 1.5 – Операторы языка Lotus Script

Оператор	**Операция**
^	Возведение в степень
–	Вычитание, минус
+	Сложение
*	Умножение
/	Деление с плавающей точкой
\	Целочисленное деление
Mod	Деление по модулю (остаток)
=	Равно (присваивание)
>	Больше чем
<	Меньше чем
<>, ><	Не равно
>=, =>	Больше или равно
<=, =<	Меньше или равно
Not	Логическое отрицание
And	И
Or	Или
Xor	Исключающее или
Eqv	Эквивалентность
Imp	Импликация
&, +	Конкатенация
Like	Содержит (contains)

К операторам управления ходом выполнения сценария относятся:

1 Блочные операторы:

а) **If…Then**

б) **If…Then…Else**

в) **If…Then…Elseif**

г) **Select Case**

2 Блочные операторы цикла:

а) **Do…Loop**

б) **Do While…Loop**

в) **Do Loop…While**

г) **Do Until…Loop**

д) **Do Loop…Until**

е) **While…Wend**

ж) **For – Next**

з) **ForAll – End ForAll**

3 Операторы ветвления:

а) **GoTo**

б) **GoSub**

в) **If…GoTo…Else**

г) **On…GoTo**

д) **On…GoSub**

е) **Return**

4 Операторы прерывания:

а) **End**

б) **Exit**.

Блочные операторы

Операторы **If…Then**, **If…Then…Else**, **If…Then…Elseif** заканчиваются оператором **End If**. Если операторы **If…Then**, **If…Then…Else** полностью располагаются на одной строке, то оператор **End If** не нужно указывать. В этом случае, для расположения нескольких операторов, например, после ключевого слова **Then** необходимо их разделять двоеточием. Оператор **If…Then…Else** равносилен цепочке операторов **If…Then**.

Оператор **Select Case** заканчивается оператором **End Select**. Блочный оператор **Select Case** эквивалентен цепочке операторов **If…Then**. До начала выполнения оператора после ключевых слов **Select Case** указывается переменная-селектор, используемая для проверки условий. В каждой строке **Case** содержится условие (или условия), на соответствие которых проверяется указанная переменная. Если в процессе проверки одного из условий **Case** возвращается значение *True*, то выполняются операторы, содержащиеся в данном блоке **Case**. Если ни одно из условий указанных после **Case** не выполняется, то выполняются операторы, расположенные в блоке **Case Else** (он не является обязательным). Затем управление передается оператору, стоящему за оператором **End Select**.

Пример.
```
Dim kol, cena as long
Dim uidoc as NotesUIDocument
…
retcode = Inputbox$(“Введите количество”)
kol = clng(retcode)
If kol<100 Then
        Print “Количество меньше 100!”
        Print “Оно должно быть в диапазоне от 100 до 1000!”
        Kol=1000
        Print “Количество установлено равным 1000!”
```

```
ElseIf kol>1000 then
        Print “Количество больше 1000!”
        Print “Оно должно быть в диапазоне от 100 до 1000!”
        Kol=1000
        Print “Количество установлено равным 1000!”
Else
        Call uidoc.FieldSetText (“Count”, kol)
        Print “Количество успешно сохранено!”
End If
retcode = Inputbox$(“Введите цену товара”)
cena = clng(retcode)
Select Case cena
Case =0
        Print “Установите цену!”
Case < min
        Print “Цена меньше минимальной!”
        Cena=min
        Print “Цена установлена равной минимальной цене!”
Case > max
        Print “Цена больше максимальной!”
        Cena=max
        Print “Цена установлена равной максимальной цене!”
Case Else
        Call uidoc.FieldSetText (“Price”, cena)
        Print “Цена успешна сохранена!”
End Select
```

Блочные операторы цикла

Операторы, расположенные между ключевыми словами **While…Wend**, будут выполняться до тех пор, пока выполняется условие заданное после **While**. Если условие ложно, то операторы не выполняются ни разу.

Существует несколько вариантов цикла **Do**. Условие, управляющее циклом **Do**, может проверяться или вначале цикла или в его конце, а сам цикл может выполняться до тех пор, пока выполняются условия **While** или **Until**. Если условие проверяется в конце блока операторов, то блок будет выполнен, по крайней мере, один раз. Блочные операторы с **While** выполняются, пока условие указанное после While истинно, а с **Until** – пока условие ложно.

Пример. Подсчитаем сумму четных чисел в диапазоне от 0 до 100.

```
Dim sum, i as integer
i=2
Do While i <= 100
        sum=sum +i
        i=i+2
Loop
```

В операторе **for** блок операторов выполняется заданное число раз (пока переменная цикла не достигнет конечного значения), а в операторе **forall** – один раз для каждого элемента в заданном списке или массиве.

Формат описания оператора **for**

```
for переменная = начальное_значение to конечное_значение [step приращение]
        [операторы]
next [переменная]
```

Переменная, начальное и конечное значение, приращение должны быть целых типов. Если приращение не указано, то переменная цикла будет увеличиваться на 1 на каждом шаге. Приращение может быть отрицательным

Пример. Построим вектор, содержащий квадраты чисел от 1 до 10

```
Dim  i as integer
Dim a (1 to 10) as integer
For i =1 to 10
        A(i) = i*i
Next i
```

Операторы ветвления

Они передают управление из данного места программы в какое-либо другое место. Оператор **GoTo** передает управление метке, расположенной в каком-либо месте программы, возврат в исходную точку не происходит. Такой возврат можно осуществить с помощью оператора **GoSub**, который передает управление именованной подпрограмме или метке, но после завершения вызванной подпрограммы выполнение сценария продолжается со следующего за **GoSub** оператора.

Оператор **If…GoTo…Else** похож на оператор **If…Then…Else.** Различие лишь в том, что когда проверяемое условие выполняется, то происходит переход к метке, имя которой указано после оператора **GoTo**. Оператор **On…GoTo** аналогичен оператору **If**. После проверки условия в операторе **On** управление передается одной из меток, имена которых указываются в операторе **GoTo**.

Оператор **On…GoSub** функционирует подобно оператору **On…GoTo**, разница состоит в том, что после выполнения операторов, указанных после метки, управление передается оператору, следующему за оператором **On…GoSub**. Оператор **Return** передает управление оператору, следующему за вызвавшим ветвление оператором.

Операторы прерывания

Они применяются для того, чтобы преждевременно выйти из процедуры или из циклов **Do**, **For** или **ForAll**. Оператор **End** также обозначает конец подпрограммы или функции.

Оператор **Exit** используется для прерывания выполнения текущего блочного оператора **Do**, **For** или **ForAll** или для прекращения выполнения процедуры или функции. После выполнения оператора **Exit** управление предается оператору, следующему за блочным оператором **Do**, **For** или **ForAll**, или оператору, следующему за оператором, который вызвал функцию или подпрограмму. Далее будет продолжено выполнение последующих строк программы; т.е. прекращается выполнение лишь текущего блока, подпрограммы или функции.

Оператор **End** – средство для немедленного прекращения работы. Он прекращает работу не только текущего оператора **Do**, **For** или **ForAll**, подпрограммы или функции, но и всей программы. Выполнение оператора **End** означает окончание сценария.

1.2 Работа с полями в Lotus Script

В Lotus Script на поля всегда ссылаются по их именам, и ссылка на поле возвращает его содержимое.

В Lotus Script для самого поля тип данных не устанавливается (в Notes мы его выбирали). Поле приобретает тип данных, когда в него помещаются данные или когда оно отображается в интерфейсе пользователя. **Все поля трактуются как одномерные массивы с неизвестным числом элементов**. Если необходимо сослаться на содержимое поля, то ссылаются на нулевой элемент массива (т.е. поля). Чтобы сослаться на все содержимое поля (независимо от того, представляет ли оно собой элемент или множество элементов) можно ссылаться на поле по имени.

Для ссылки на определенный элемент поля, содержащего множество элементов необходимо указывать индекс элемента (нумерация индексов в Lotus Script по умолчанию начинается с нуля).

1.3 Функции, подпрограммы, объекты, классы и события

В Lotus Script существует ряд предопределенных встроенных функций (напр. *Input*, *Month*, *Today* и др.), которые нельзя изменять, но можно создавать свои собственные функции. Каждая функция принимает определенные аргументы (не обязательно) и возвращает значение. Подпрограмма (**sub**) принимает аргументы, но не возвращает никакого значения.

Пример. Опишем пользовательскую функцию и подпрограмму.

```
Dim MyVar as string
MyVar = “зачет”
’ печатаем значение, возвращаемое функцией MyF
Print “Значение функции MyF: ” & MyF( MyVar)
MySub (MyVar) ’вызываем подпрограмму печатающую значение

Function MyF (s as sting) as string
        MyF =Ucase(s)
End function

Sub mysub (s as string)
        Print “my sub выводит: ” &s
End sub
```

В приведенном выше примере функция **Ucase** преобразует все буквы алфавита к верхнему регистру.

База данных, список ACL, формы, представления, программы-агенты, кнопки, действия и поля – все это **объекты**. В процессе развития среды Lotus Domino / Notes появляются все новые объекты (каждый объект имеет свои собственные свойства и методы). Базы данных Domino часто называют хранилищем объектов. Они являются контейнером для таких объектов как формы, представления и т.д. Каждый из этих объектов содержит в свою очередь другие объекты (понятие вместимости). Например, документ содержит такие объекты, как поля.

На объекты Notes ссылаются следующим образом: объявляют переменную ссылки на объект и затем эту ссылку на объект некоторому экземпляру данного класса.

Классы Domino разделяются на 2 группы: классы клиента (**front-end classes**) и классы сервера (**back-end classes**). Термин «клиент» (**front-end**) относится к чему-либо видимому по интерфейсу пользователя (это можно видеть на экране), а термин «сервер» (**back-end**) – к чему-либо невидимому.

Каждый отдельный класс имеет набор событий, на которые он реагирует. Наиболее распространенные события следующие: **Click**, **Entering**, **Exiting**, **Initialize**, **Terminate**, **QueryOpen**, **PostOpen**, **QuerySave**, **PostSave**. События **Initialize** (происходит, когда объект загружается) и **Terminate** (происходит, когда объект закрывается) есть у всех объектов, но у объектов могут быть и другие программируемые события.

Область действия (видимость) переменных, констант определяется по тем же правилам, как и в большинстве современных языков программирования (см. [7] стр. 531-533).

1.4 Иерархия классов. Клиентские классы

Все классы *lotus domino* разделяются на клиентские и серверные. Иерархия объектов (как и соответствующих им классов) идет сверху вниз – от сеанса к рабочему пространству, от БД к представлению и от документа к полю. Далее объекты подразделяются по типам – база данных, представление и документ.

Переменная (ссылки на объект) объектной ссылки создается в два этапа:

1. Объявление переменной
2. Создание экземпляра объекта

Пример. Dim db **as NotesDatabase**
Set Doc = **New NotesDocument** (db)

Этот оператор создает объект класса **NotesDocument** (новый документ в БД), и ссылкой на него будет переменная *doc*.

Новые объекты *Domino* можно создавать и за один этап, используя операторы формата:

Dim Doc **as New NotesDocument** (**NotesDatabase**)

Подобный оператор создает объектную ссылку типа **NotesDocument,** но не создает сам документ. В этом случае мы работаем с объектом документа, в котором можно создавать поля и помещать в них некоторые значения. Потом можно сохранить этот объект, при этом будет создан реальный документ.

Рассмотрим клиентские классы. Объектами, к которым производится обращение, будут: рабочее пространство, текущая БД и текущий документ. Рассмотрим соответствующие им классы.

Когда с объектами, относящимся к клиентским классам, пользователь работает с помощью интерфейса пользователя, то разработчик может обращаться к таким объектам и вносить изменения, которые пользователь может увидеть сразу же.

Класс **NotesUIWorkspace** – текущее рабочее пространство, отображаемое в текущем окне. Это может быть, или рабочее пространство (рабочий стол Notes), или открытый в настоящее время документ.

В рабочем пространстве можно открыть и представления, но лучше это сделать с помощью класса **NotesUIView**. Если в представлении присутствует кнопка действия, при помощи которой добавляются, удаляются, или изменяются документы, имеющиеся в представлении, то можно работать с сервером. Для того, чтобы сделанные изменения были видны пользователю можно использовать метод **ViewRefresh** класса **NotesUIWorkspace**. Если не применять этот метод, то пользователь не будет знать, что представление изменилось.

Класс **NotesUIDatabase** имеет 2 метода:

1. **OpenView.**
2. **OpenNavigator.**

Объект этого класса служит, прежде всего, для размещения в нем сценариев, присоединенных к событиям БД (например, события *PostOpen*, *QueryClose* и др.). Свойство **Documents** класса **NotesUIDatabase** предоставляет доступ ко всем документам базы данных.

Документ, открытый в данный момент и отображаемый в рабочем пространстве, является объектом класса **NotesUIDocument**. Над ним можно выполнять различные действия: перемещать курсор, вносить текстовую информацию в поле, извлекать содержимое поля, обновлять документ, отправлять его по почте. С помощью свойства **Document** этого класса, можно получить доступ к документу сервера. Метод **Refresh**, существующий только в классе **NotesUIDocument**, обновляет текущий документ, вновь выполняя все формулы входной трансляции, входной проверки и формулы вычисляемых полей. Метод **Reload** обновляет документ клиента, внося в него изменения, которые были выполнены в документе сервера. Метод **Reload** необходим только в том случае, если для свойства **AvtoReload** объекта **NotesUIDocument** установлено *false*. По умолчанию **AvtoReload** = *True*. Если выполняется множество изменений в документе сервера, то устанавливаем **AvtoReload** = *False* и после того, как все изменения в документе будут выполнены, вызываем метод **Reload**.

В агентах, запускаемых на сервере нельзя использовать UI-классы и подключать библиотеки, использующие эти классы.

1.5 Обработка ошибок

Ошибки бывают 2-х типов: ошибки компиляции и ошибки выполнения. Сообщение об ошибке компиляции отображается в нижней части экрана в окне

с раскрывающимся списком *Errors*, пока не будут исправлены все ошибки компиляции, программа не запустится.

Ошибки, которые обнаруживаются только во время выполнения программы, называются *ошибками выполнения*. Они могут быть различного типа. Рассмотрим функции и операторы языка Lotus Script, облегчающие обработку ошибок:

1. Функция **Erl** возвращает номер строки, где произошла ошибка;
2. Функция **Err** возвращает номер текущей ошибки;
3. Оператор **Err** устанавливает номер текущей ошибки;
4. Функция **Error (Error$)** возвращает сообщение о текущей ошибке (заранее определенный номер ошибки);
5. Оператор **Error** генерирует заранее определенный номер ошибки и сообщение о ней;
6. Оператор **On Error** указывает имя подпрограммы обработки ошибок или определяет, как обрабатывать ошибки (или все, или только какого-либо определенного типа);
7. Оператор **Resume** указывает, каким образом будет продолжено выполнение программы после ошибки.

Когда происходит ошибка, текущая процедура проверяется на наличие оператора **On Error**. Если он найден, то он выполняется. Если нет, то проверяется процедура, вызвавшая данную процедуру (если она есть). Если находится оператор **On Error**, то он выполняется, иначе проверяется следующая вызывающая процедура и т.д. Если подпрограмма обработки данной ошибки *не будет найдена*, то на экран выводится сообщение об ошибке и выполнение программы прекращается.

Операторы **Err** и **Error** используются для установки или генерации каких-либо ошибок и часто применяются при тестировании подпрограмм обработки ошибок (особенно собственных).

Оператор **On Error** может содержать номер ошибки. Когда происходит ошибка с этим номером, то управление передается подпрограмме обработки ошибок, которая задана в операторе **On Error**. Но оператор **On Error** может и не содержать номера ошибки, тогда управление передается указанной в операторе подпрограмме обработки ошибок при возникновении любой ошибки не указанной ни в одном другом операторе **On Error**. В программе может быть любое количество операторов **On Error**. Все они обрабатывают ошибки определенных типов, за исключением одного оператора, который обрабатывает все остальные. Если для одного номера ошибки задаются 2 оператора **On Error**, то выполняться будет последний из них.

Существует 5 конструкции операторов, которые могут следовать за оператором **On Error**:

а) **goto label;**
б) **resume next;**
в) **goto 0;**
г) **resume 0;**
д) **resume label.**

Конструкция **goto label** при возникновении ошибки передает управление оператору, следующему за меткой **label**. Конструкция **resume next** передает управление оператору, следующему за оператором, вызвавшим ошибку (продолжить, не обращая внимания на ошибку). Конструкция **Goto 0** указывает, что в текущей процедуре ошибка не должна обрабатываться.

Оператор **resume 0** указывает на то, что оператор, вызвавший ошибку, должен быть выполнен снова. Оператор **resume label** передает управление оператору, следующему за меткой label (resume label используется только в подпрограммах обработки ошибок). Оператор **resume** также восстанавливает значение функций **Err**, **Erl**, **Error**. Когда они сброшены, то это означает, что либо ошибок нет, либо все они обработаны.

Используя функцию **Erl** можно вывести на экран пользователю номер строки оператора, вызвавшего ошибку.

При обработке ошибок часто применяется оператор **Exit sub**, который служит в качестве точки выхода из процедуры, если она завершается нормально, т.е. ошибки отсутствуют.

Ошибки можно задавать не только номером, но и с помощью символических имен констант.

Пример. On Error ErrFileNotFound **Goto** m1

Эти константы хранятся в файлах *LSERR.LSS*, *LSXUIERR.LSS*, *LSXBEERR.LSS*.

1.6 Взаимодействие с пользователем: функции MessageBox, InputBox, метод DialogBox

Функция **MessageBox** выводит на экран информацию для пользователя. Функция **InputBox** обеспечивает ввод пользователем информации в программу. Метод **DialogBox** отображает на экране форму или документ в диалоговом окне.

При использовании этих функций и метода необходимо подключить к программе файл констант *LsConst.lss*, тогда можно будет использовать имена констант в качестве аргументов.

Функцию **MessageBox** можно записать в виде **MsgBox**.

Пример.

RetCode=**MessageBox**(“Вы хотите продолжить?”, MB_YESNOCANCEL).

Переменной RetCode присваивается значение, возвращаемое функцией MessageBox – число, указывающее какую кнопку выбрал пользователь.

Существует и оператор **MessageBox**, который не возвращает ни какого значения.

Пример. MessageBox “ошибка”, MB_OK.

Формат функции MessageBox

MessageBox (message[,[buttons+icon+default+mode][, boxTitle]])

Аргумент *Message* – текстовое сообщение, которое будет отображаться в окне. Каждый из аргументов *Buttons*, *Icon*, *Default*, *Mode* – целые числа, объединение этих чисел в одно определяет, как окно сообщений будет выглядеть и функционировать. Аргумент *Buttons* определяет, какие кнопки будут отображаться в окне (значения можно задавать цифрами или константами (см. в таблице 1.6)). Аргумент *Icon* определяет, какая пиктограмма может отображаться в окне сообщения (см. в таблице 1.7). Аргумент *Default* определяет, какая кнопка будет считаться нажатой по умолчанию, если пользователь нажимает пробел или ввод (см. в таблице 1.8). Аргумент *Mode* определяет, будет ли окно сообщения окном модального приложения (выполнение текущего приложения останавливается до тех пор, пока пользователь не даст ответ в окне сообщения) или окном системного модального приложения (выполнение всех приложений останавливается до тех пор, пока пользователь не даст ответ в окне сообщения), подробнее см. в таблице 1.9. Аргумент *BoxTitle* – строковая переменная, длинной до 128 символов, значение которой отображается в области заголовка окна.

Таблица 1.6 – Возможные значения аргумента Button

Имя константы	**Значение**	**Кнопки**
MB_OK	0	OK
MB_OKCANCEL	1	OK, Cancel
MB_ABORTRETRYCANCEL	2	Abort, Retry, Cancel
MB_YESNOCANCEL	3	Yes, No, Cancel
MB_YESNO	4	Yes, No
MB_RETRYCANCEL	5	Retry,Cancel

Таблица 1.7 – Возможные значения аргумента Icon

Имя константы	**Значение**	**Пиктограмма**
MB_ICONSTOP	16	Знак «стоп»
MB_ICONQUESTION	32	Вопросительный знак
MB_ICONEXCLAMATION	48	Восклицательный знак
MB_ICONINFORMATION	64	Информация

Таблица 1.8 – Возможные значения аргумента Default

Имя константы	**Значение**	**Кнопка, нажимаемая по умолчанию**
MB_DEFBUTTON1	0	Первая кнопка
MB_DEFBUTTON2	256	Вторая кнопка
MB_DEFBUTTON3	512	Третья кнопка

Таблица 1.9 – Возможные значения аргумента Mode

Имя константы	**Значение**	**Режим**
MB_APPLMODAL	0	Приложение
MB_SYSTEMMODAL	4096	Система

Пример. 3 способа кодирования окна сообщения.

1) ret=**MessageBox** (“продолжить?”, 4387, “ошибка!”)
2) ret=**MessageBox** (“продолжить?”, 3+32+256+4096, “ ошибка!”)
3) ret=**MessageBox** (“продолжить?”, MB_YESNOCANCEL+ MB_ICON-QUESTION+ MB_DEFBUTTON2+MB_SYSTEMMODAL, “ ошибка!”)

Функция **MessageBox** возвращает целое число, которое может быть представлено в виде имени константы (см. в таблице 1.10).

Таблица 1.10 – Возможные значения аргумента Button

Значение	**Кнопка**	**Константа**
1	OK	IDOK
2	Cancel	IDCANCEL
3	Abort	IDABORT
4	Retry	IDRETRY
5	Ignore	IDIGNORE
6	Yes	IDYES
7	No	IDNO

Функция **InputBox** позволяет пользователю вводить в окне данные и передает их в программу. Существуют 2 разновидности InputBox:

1) **InputBox,** которая возвращает данные типа *variant*;
2) **InputBox$,** которая возвращает строковые данные.

Формат описания функции InputBox

InputBox[**$**](prompt[,[title][,[Default][,Xpos, Ypos]]]).

Аргумент *Promt* – строка, длинной до 128 символов, определяющая текст в окне ввода. Аргумент *Title* – заголовок окна (до 128 символов). Аргумент *Default* – величина, которая отображается в редактируемом поле окна ввода. Аргументы *Xpos*, *Ypos* – расстояние в пикселях от верхнего левого угла экрана до верхнего угла окна вывода по горизонтали и вертикали соответственно.

Пример. InputBox$ (“введите количество гаражей” , “ число” , 30)

Применяя функцию **InputBox**, необходимо преобразовывать возвращаемые значения (данные типа Variant) в данные требуемого типа.

Метод **DialogBox** позволяет отображать определенную форму в диалоговом окне. Информация, введенная пользователем в форме, передается в поля базового документа формы. Основное отличие метода **DialogBox** от функции **@DialogBox** – то, что с помощью метода можно отображать не только текущий документ, но и любой другой документ в базе данных.

Вопросы для самоконтроля

1. Преимущества языка программирования *Lotus Script*.
2. Что такое суффикс типа?
3. Как можно задавать константы в *Lotus Script*?
4. Что такое литерал?
5. Как объявить переменную в *Lotus Script*?
6. Применение оператора **Deftype**.
7. Как описываются массивы в *Lotus Script*?
8. Особенности работы со списками и данными типа *Variant*.
9. Блочные операторы в языке *Lotus Script*.
10. Какие существуют в *Lotus Script* блочные операторы цикла?
11. Перечислите операторы ветвления и прерывания в *Lotus Script*.
12. Особенности работы с полями в *Lotus Script*.
13. Как описываются функции и процедуры в *Lotus Script*?
14. На какие две большие группы разделяются все классы *Domino*?
15. Объявление объектной ссылки.
16. Перечислите основные клиентские классы.

17. На какие 2 группы можно разделить ошибки в *Lotus Script*?

18. Перечислите основные функции и операторы языка *Lotus Script*, позволяющие обрабатывать ошибки?

19. Особенности использования оператора **On Error**.

20. Зачем используются операторы **Err**, **Error**?

21. Перечислите и кратко охарактеризуйте конструкции операторов, которые могут следовать за оператором **On Error**.

22. Формат описания и область применения функции **MessageBox**.

23. Функция **InputBox**, метод **DialogBox**.

Задание

Напишите на *LotusScript* проверку заполнения обязательных полей на форме с выводом сообщения и передачей фокуса ввода на незаполненное или неправильно заполненное поле. Необходимо использовать обработчики событий *Exiting* поля, *QuerySave* формы. Для вывода сообщений необходимо использовать как функцию **MessageBox**, так и функцию **InputBox**.

Тема 2 РАБОТА С ДОКУМЕНТАМИ-ОТВЕТАМИ

2.1 Подчиненные формы и общие поля

Подчиненные формы (подформы, **subforms**) не используются самостоятельно, а могут быть вставлены в другие формы, в том числе и программно. Одну и ту же подформу можно вставить в несколько форм, т.е. подчиненные формы – повторно используемый ресурс. Процесс создания подчиненных форм ни чем не отличается от процесса создания обычных форм. Подформы создаются в *Shared Code* → *SubForms*. Для включения подчиненной формы в обычную форму необходимо:

1) Открыть основную форму в режиме конструктора и поместить курсор в то место, куда необходимо вставить подчиненную форму;
2) Выбрать команду меню *Create* → *Resource* → *Insert SubForm*;
3) В появившемся окне выбрать требуемую подформу.

Общие поля (*shared fields*) применяются, когда необходимо использовать одно и тоже поле в нескольких формах. **В этих полях общими элементами являются свойства и формулы, но не данные!** Общее поле можно создать, выполнив команду меню *Create* → *Design* → *Shared Field* или сделать существующее поле общим (*Design*→*Share This Field*). Уже созданные общие поля можно вставлять в форму, выполнив команду меню *Create* → *Resource* → *Insert Shared Field*, команду *Insert Shared Field* всплывающего меню или с помощью пиктограммы  панели пиктограмм. После создания общего поля откроется окно его свойств. После установки свойств поля можно написать формулы в определенных событиях, точно также как это делается и в других элементах дизайна. Единственное отличие состоит в том, что панель Design не отображается в интегрированной среде разработки, так как проектируется не форма.

2.2 Работа с идентификаторами документов в языке формул

1) Функция **@DocumentUniqueID**

Область применения: нельзя использовать в формулах навигатора. В формулах полей создает ссылку (*DocLink*) на текущий документ.

Данная функция возвращает так называемый универсальный идентификатор документа (**UNID**) – 32-х символьную комбинацию шестнадцатитиричных цифр, уникально идентифицирующую текущий документ во всех репликах распределенной базы.

Универсальный идентификатор документа можно посмотреть в окне свойств документа на последней закладке в первых двух строках после метки ”ID”, отбросив символы OF в начале и ON в середине.

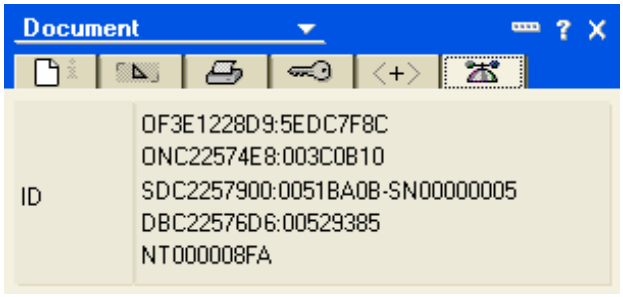

Рисунок 2.1 – Последняя закладка свойств документа

Domino генерирует универсальный идентификатор при создании документа. Первые его 16 символов представляют собой дату-время создания документа (с точность до тиков), а вторые 16 символов – случайное число. Domino проверяет, не оказалось ли в текущей реплике базы документа с таким же универсальным идентификатором, и если это так, то снова случайно генерирует вторые 16 символов. После сохранения документа в базе данных его универсальный идентификатор не изменяется. Исключение составляют случаи, когда документ копируется в базу через буфер обмена или пересылается почтой – перед помещением документа в базу проверяется, не существует ли в ней уже документ с таким же универсальным идентификатором, и если это так, то для добавляемого в базу документа генерируется новый универсальный идентификатор. С помощью *Lotus Script* программист может самостоятельно изменять **UNID** документа, но в этом случае необходимо самостоятельно заботиться об уникальности **UNID**.

Третья и четвертая строки после метки ”ID” содержат информацию, специфичную в каждой реплике распределенной базы [9].

Многие понятия и алгоритмы Domino, прежде всего «дерево» документов-ответов, базируются на универсальных идентификаторах документов. Так документ-ответ всегда содержит предопределенное поле с именем **$Ref**, **в котором хранится универсальный идентификатор документа-родителя для данного документа-ответа**. В поле **$Ref** отображается на 2 символа больше (всего 34): добавляются символы F и N (из OF и ON соответственно).

Пример. Формула столбца представления выводит в этом столбце для некоторого документа его **UNID**.

@Text(@DocumentUniqueID)

2) Функция **@InheritedDocumentUniqueID**

Область применения: нельзя использовать в формулах навигатора. Предназначена для работы с документами, созданными по форме с включенной опцией наследования полей. Если опция не включена, то возвращает такое значение, что и функция **@DocumentUniqueID**. В формулах полей создает ссылку (*DocLink*) на текущий документ.

Данная функция возвращает универсальный идентификатор документа, который был текущим в момент создания документа, в котором используется эта функция.

3) Функция **@NoteID**

Область применения: нельзя использовать в формулах формы и навигатора.

Данная функция возвращает восьмисимвольный идентификатор документа в базе – строку с префиксом NT, например, NT000008FA (последняя строка последней закладки в окне свойств документа, см. рис. 2.1). **@NoteID** уникально идентифицирует документ только в текущей базе данных.

4) Функция **@GetDocField(UNID; имя_поля)**

Область применения: нельзя использовать в формулах отбора, колонок, всплывающих окон и навигатора.

Для текущей базы данных функция возвращает значение поля с именем **имя_поля** из документа с универсальным идентификатором **UNID**. Параметр **имя_поля** – строка, указывается в кавычках. Возвращаемое значение зависит от типа поля: строка или список строк, число или список чисел, дата-время или временной диапазон. Если задан не существующий UNID или поле, то возвращается пустая строка.

Пример. Вычисляемое поле в документе-ответе всегда будет содержать значение поля Subject из главного документа. Когда создается новый документ-ответ, то значение в вычисляемом поле наследуется из главного документа.

@If (**@IsNewDoc**; Subject; **@GetDocField** (**$Ref**; ”Subject”))

5) Функция **@SetDocField(UNID; имя_поля; новое_значение)**

Область применения: нельзя использовать в формулах отбора, колонок, всплывающих окон и навигатора.

Для текущей базы данных функция присваивает новое значение полю с именем **имя_поля** в документе с универсальным идентификатором **UNID**. Па-

раметр **имя_поля** – строка, указывается в кавычках. Параметр **новое_значение** должен соответствовать типу поля.

Функция **@SetDocField** наиболее часто используется в формулах полей, кнопок, агентов.

Пример. Формула кпопки-действия в документе-ответе изменяет значение поля Subject в главном документе:

@SetDocField(**$Ref**; ”Subject”; ”Новое значение”)

2.3 Основные команды для работы с текущими документами в языке формул

1) **@Command**([**Compose**]; ”сервер”: ”база данных”; ”форма”; ”ширина окна”: ”высота окна”)

Область применения: нельзя использовать в формулах диалоговых окон.

Данная команда в базе данных, указанной параметрами *сервер* и *база данных*, создает новый документ по форме *форма* и переключает фокус Domino на него. Если база данных является локальной, то вместо параметра сервер указывается пара пустых кавычек. Если документ необходимо создать в текущей базе данных, то второй параметр также заменяем ””.

Параметр *форма* задает форму (имя или псевдоним). Если этот параметр опущен, и база данных текущая, то будет выведено диалоговое окно для определения формы, по которой должен быть создан документ. Если команда используется в действии представления, у которого определена формула формы или в формуле кнопки панели инструментов, и на экране открыто представление с непустой формулой формы, то новый документ будет открыт по форме из формулы формы данного представления.

При создании документов типа *ответ* или *ответ на ответ*, база данных, в которой планируется создать такой документ, должна быть открыта, и на документе, на который создается ответ, должен стоять световой маркер (курсор). Также для создания ответных документов можно применять команду [**ComposeWithReference**].

Команда может быть использована в Web-приложениях только в формате:

@Command([**Compose**]; ”форма”)

Параметры *ширина окна*: *высота окна* задают соответственно высоту и ширину в дюймах окна, в котором будет создаваться документ. Использовать параметры нет смысла, если выбран MDI (Multiple Document Interface – многооконный интерфейс в рамках главного окна) интерфейс, и окно имеет установ-

ленное свойство – максимальный размер. Если же эти параметры опущены или равны 0, то окно имеет размеры, сохраненные последним пользователем.

Пример. Создадим в текущей базе данных новый документ по форме ”Main_form”:

@Command([Compose];""; "Main_form")

2) **@Command**([**ComposeWithReference**]; ”сервер”: ”база данных”; ”форма”; ”флаги”)

Область применения: база данных должна быть открыта, на документе, для которого создается ответ, должен стоять световой маркер или документ должен быть открыт в режиме чтения/ редактирования.

Создает ответный документ для документа, на котором стоит световой маркер, и переключает фокус на него. Под ответным здесь понимается не обязательно документ, содержащий ссылку на родительский документ в поле **$Ref**, просто создаваемый документ может содержать информацию из документа родителя (что-то вроде наследования).

Параметры *сервер*: *база данных* и *форма* имеют тот же смысл, что и в команде **[Compose]**. Единственное отличие состоит в том, что если в новый документ требуется передать какие-либо данные из родительского документа, то в *форме*, указанной в команде, обязательно должно присутствовать **RT** поле **Body** (данной информации нет в справке [9]). В противном случае (т.е. указан параметр флаги, и у формы отсутствует **RT** поле с именем **Body**) выдается сообщение об ошибке «*Inherit rich text field does not exist*» (RT-поля для наследования не существует) [9]. Но это не всегда так, если в качестве флага использовать «2», то ошибка не возникает (см. пример ниже).

Точно также как и в команде [**Compose**], если [**ComposeWithReference**] используется в действии формы или кнопки панели инструментов при открытом представлении с определенной формулой формы, то новый документ будет открыт по форме из формулы формы данного представления (как это изменить см. [9, стр. 237]).

Параметр *флаги* не является обязательным, он определяет параметры ссылки на родительский документ. Если данный параметр опущен, то новый документ будет создан без ссылки на родительский документ. Тип значения параметра *флаги* – число, поэтому для задания нескольких флагов в качестве параметра указывается сумма их значений. Допустимыми для параметра являются следующие значения:

- **1** – включает в поле **Body** ссылку (**DocLink**) на родительский документ. В Web-приложениях не поддерживается;

- **2** – включает в поле **Body** содержимое родительского документа. **Для полноценной работы** требует наличие флага **1**. В Web-приложениях в действиях формы (но не представления) корректно передается только текстовая информация из родительского документа. Даже если поле **Body** отображается в Web с помощью апплета, то графические изображения из родительского документа передаются некорректно;
- **4** – включает в поле **Body** содержимое родительского документа в виде свертываемого раздела. Требует наличие флагов **1** и **2**. В Web-приложениях не поддерживается;
- **8** – включает в поле **Body** содержимое родительского документа в виде интернет копии (т.е. «кто-то и тогда-то написал» в заголовке, а далее текст из родительского документа, предваряемый символами больше «>». Все остальные объекты будут удалены с соответствующим сообщением). В Web-приложениях не поддерживается. Требует наличие флагов **1** и **2**. Неявно применяет флаг **16**;
- **16** – включает в поле **Body** содержимое родительского документа, удаляя из последнего присоединенные файлы, графические изображения и другие большие объекты, заменяя их на сообщение об удалении, заключенное в квадратные скобки. В Web-приложениях не поддерживается. Требует наличие флагов **1** и **2**;
- **32** – включает перед наследуемым текстом подформу с им*енем $ForwardSep*, если она существует в базе данных (в стандартном почтовом шаблоне **mail6.ntf** такая подформа присутствует). В Web-приложениях не поддерживается. Требует наличие флагов **1** и **2**.

Пример. Кнопка-действие должна создавать ответные документы по форме main_form8, при этом часть полей основной формы будут отображаться в ответной форме.

@Command ([**ComposeWithReference**]; "": ""; "main_form8"; 2)

Вместо значения флага «2» можно использовать и «1», но тогда необходимо создавать поле *Body* (см. выше).

Пример. Откроем в текущей базе данных форму *CRResp*, которая имеет RT-поле с именем **Body**. В поле **Body** скопируем информацию в виде свертываемого раздела из документа, который был текущим на момент выполнения формулы. Это все достигается использованием в действии представления следующей формулы:

@Command ([**ComposeWithReference**]; "": ""; "CRResp"; 1+2+4)

Вопросы для самоконтроля

1. Зачем используются подчиненные формы?
2. Как создать подчиненную форму?
3. Создание и использование общих полей.
4. Какие функции применяются для работы с идентификаторами документов в языке формул?
5. Что такое **UNID**? Как и когда он создается?
6. Где и зачем используется поле **$Ref**?
7. Отличие функций **@DocumentUniqueID** и **@InheritedDocumentUniqueID**.
8. Формат описания функций **@GetDocField** и **@SetDocField**.
9. Перечислите основные команды для работы с текущими документами в языке формул.
10. Зачем используется команда [**Compose**]?
11. Отличия команд [**Compose**] и [**ComposeWithReference**].
12. Перечислите значения, которые может принимать параметр флаги.
13. Как избавиться от ошибки «*Inherit rich text field does not exist*»?

Задания

1. Перенесите область заголовка с формы в подформу. Создайте форму с полем комментария аналогичную по дизайну основной форме и использующую ту же подформу в области заголовка. Данная форма должна использоваться для создания документов-ответов для основных документов и должна наследовать часть основных полей (поля помеченные звездочкой в режиме редактирования) основного документа и отображать их без возможности редактирования.

2. Создайте еще одно представление для отображения основных документов вместе с документами-ответами в иерархическом виде. Представление должно содержать кнопку для создания документов-ответов.

3. Создайте скрытое представление для отображения только документов-ответов и внедрите его на форму основного документа так, чтобы отображался список документов-ответов только для открытого в данный момент документа.

Алгоритм выполнения задания 1:

1. Создать подчиненную форму;
2. Перенести область заголовка в созданную подформу;
3. Создать форму для создания ответных документов (тип *Response*);
4. Для отображения в форме, используемой для создания ответных документов, части основных полей основного документа без возможности редактирования необходимо: создать вычисляемые поля (*Computed When Composed*),

в формулах которых можно указать либо имя поля основной формы или использовать формулу **@GetDocField(@InheritedDocumentUniqueID**; "Имя_поля"). В ответной форме **должна быть включена опция** наследования полей (*Formulas Inherit Values from selected document*).

Алгоритм выполнения задания 2:

1. Создать представление;
2. Вставить в созданное представление столбец, в котором будет отображаться поле комментария из ответной формы;
3. Сделать столбец с комментарием первым по порядку в представлении;
4. В столбце с комментарием необходимо включить опцию *Show Responses Only*, чтобы значения в нем отображались только для документов ответов;
5. В столбце, следующем за созданным столбцом включить опцию показа *twistie*;
6. Создать кнопку-действие, создающую документы-ответы (см. раздел 2.3).

Алгоритм выполнения задания 3:

1. Создать скрытое представление (чтобы представление стало скрытым, его имя необходимо указать в круглых скобках);
2. Создать в скрытом представлении 2 столбца;
3. В первом столбце в свойстве *Column Value* записать формулу: **@Text($Ref)**, т.е. объединяем документы по ссылкам на родительский документ. Первый столбец скрытого представления должен быть **категоризированным,** т.к. требуется отображать документы-ответы только для данного документа. В первом столбце также включить опцию показа *twistie*;
4. Во втором столбце отобразить поле комментария из ответной формы;
5. В свойстве *View Selection* скрытого представления записать формулу: **SELECT Form**="Имя_ответной_формы". Это необходимо для отображения только документов-ответов, созданных по ответной форме;
6. На второй закладке свойств скрытого представления отменить установку флажка *Show Response Documents In Hierarchy*;
7. Внедрить созданное скрытое представление на форму (команда меню *Create→Embedded Element→View*);
8. В свойстве *Show Single Category* внедренного представления записать: **@Text(@DocumentUniqueID)** – т.е. необходимо показывать в представлении только документы с категорией «**UNID** текущего документа на главной форме» в текстовом виде, при этом сама категория не показывается.

Тема 3 ПРОГРАММЫ-АГЕНТЫ

3.1 Работа с программами-агентами

Для создания программы-агента необходимо раскрыть *Shared Code*→ *Agents* в *Domino Designer*, нажать кнопку *New Agent*. Для вновь созданного агента необходимо написать программный код (на языке формул, Lotus Script, Java, Java Script или выбрать простое действие из списка) хотя бы в одном из двух событий: **Initialize**, **Terminate** (последнее используется реже).

Управляющими средствами программ-агентов являются события.

Программы-агенты можно запускать по расписанию в любое время, они могут выполняться как следствие изменения или дополнения документа. Пользователь может запускать программу-агент, она может быть также запущена в результате получения почтового сообщения.

Все действия, отображаемые в меню *Actions* – это программы-агенты, они также используются повсеместно в почтовой базе *Mail* и инструментах календаря.

Если в меню *Design* выбрать программы-агенты, то список всех доступных программ-агентов базы данных будет показан в представлении, столбцы которого описывают различные свойства программ-агентов:

1) **Name/ Comment** – имя программы-агента и сопроводительные документы;

2) **Alias** – псевдоним имени программы-агента;

3) **Trigger** – значения поля *When Should This Agent Run* (когда должна выполняться данная программа-агент);

4) **Owner** (владелец) – *Shared*, если это общая программа-агент или имя конкретного пользователя;

5) **Notes** – если отмечено, то программа-агент запускается в *Notes*;

6) **Web** – если отмечено, то программа-агент запускается в *Web*.

Рассмотри несколько примеров программ-агентов. Программа-агент *Archive* переносит документы в архив почтовой базы данных. Программа-агент *Page Minder* в *Notes Personal Navigator* проверяет наличие обновлений для выбранной Web-страницы, а затем обновляет ее по расписанию.

В типичном приложении, например, в приложении рабочего потока, программы-агенты могут отсылать уведомления о том, что время просмотра истекло. Другая программа-агент может добавить или изменить вновь созданный документ. Дискуссионные базы данных часто содержат программы-агенты, отправляющие сообщения участникам дискуссии после изменения документа или

при добавлении нового документа в базу данных. Более сложные программы-агенты могут управлять перемещением данных из внешних баз данных в базы данных сервера Domino. Для выполнения сложного многоэтапного процесса программы-агенты могут подключать другие программы.

Очень большое количество программ-агентов можно создать для работы исключительно в Web-приложениях. Их можно создать, например, с помощью Java, а затем расширить их возможности для применения в Web-браузерах.

3.2 Создание программ-агентов на языке Lotus Script

Пример. Программа-агент для отправки по почте напоминаний, написанная на языке *LotusScript*. Нижеописанная программа-агент выполняется ежедневно по расписанию в базе данных, содержащей формы *REQ*. В этих формах существуют: поле **CompletionDate**, содержащее дату выполнения задания; поле **Status**, отображающее статус документа; поле **PersonAssigned**, указывающее, кто отвечает за выполнение задания. Программа-агент выполняется каждый день, просматривая все документы в базе данных. Если срок исполнения документа уже прошел, а документ не завершен, то исполнитель получает почтовое напоминание со ссылкой на незавершенный документ.

```
Sub Initialize
  Dim s As New Notes Session
  Dim db As NotesDatabase
  Dim collection As NotesDocumentCollection
  Dim note, memo As NotesDocument
  Dim item As NotesItem
  Dim rtitem As NotesRichTextItem
  Dim cutoff As New NotesDateTime (Today)
  Dim duedate As NotesDateTime
  Set db = s.CurrentDatabase
  Set collection = db.AllDocuments
  Set note = collection.GetFirstDocument
  Do Until note is nothing
    If note.Form(0) = ”REQ” Then
      Set Item = note.GetFirstItem (”CompletionDate”)
      Set duedate = item.DateTimeValue
      If (note.Status(0)=”Open”)and(cutoff.Timedifference(duedate)>0)
      Then
        Set memo = New NotesDocument(db)
        memo.Form=”Memo”
```

```
        memo.SendTo=note.PersonAssigned(0)
        memo.Subject=”Невыполненное задание”
        Set rtitem = New NotesRichTextItem (memo,”Body”)
        Call rtitem.AppendText(”Напоминаем, что у вас есть невыполненное задание!”)
        Call rtitem.AddNewLine(2)
        Call rtitem.AppendText(”Нажмите на этой ссылке, чтобы увидеть незавершенный документ!”)
        Call rtitem.AppendDocLink(note, ”Нажмите для просмотра задания”)
        Call memo.Send(false)
      End If
    End If
    Set note=collection.GetNextDocument(note)
  Loop
End Sub
```

После объявления всех переменных инициализируются объекты *db*, *collection* и *note*. Далее идет цикл **Do…Until** на основе условия: указывает ли ссылка note на какой-либо документ. Сначала внутри цикла проверяется, создан ли документ по форме *REQ*. Если это не так, то переходим к следующему документу. Если документ создан на основе формы *REQ*, то проверяются остальные условия: документ не завершен и срок его исполнения уже истек. Если любое из этих условий не выполняется, то переходим к следующему документу (выполняется оператор, стоящий перед **Loop**). Если же оба условия выполняются, то мы должны отправить сообщение по электронной почте. Для этого создается новый объект *memo* класса *NotesDocument*, и в нем задаем поля *Form*, *SendTo*, *Subject*. В основной части письма будут находиться короткое сообщение, напоминающее пользователю о невыполненном задании и ссылка на незавершенный документ.

В данной программе-агенте для форматирования поля *Body* используются методы *AddNewLine*, *AppendText*, *AppendDocLink* класса *NotesRichTextItem*, позволяющие легко производить различные действия над форматированным текстом. Сначала, используя метод *AppendText*, добавляем в поле *Body* текст «Напоминаем, что у вас есть невыполненное задание!». Затем, используя метод *AddNewLine*, добавляем 2 пустые строки. Далее с помощью метода *AppendText* добавляем текст «Нажмите на этой ссылке, чтобы увидеть незавершенный документ!». И, наконец, применяя метод *AppendDocLink*, отображаем ссылку на незавершенный документ.

У метода *AppendDocLink* существует один обязательный параметр – документ, на который указывает ссылка, и один необязательный параметр, представляющий собой текст, который отображается в строке состояния окна *Notes*, когда курсор устанавливается на ссылке.

После создания объекта memo отправляем его (метод Send, параметр false означает, что при отправке документа к нему не будет присоединена форма), переходим к следующему документу и повторяем весь процесс.

Для действий над форматированным текстом можно применять и другие методы класса *NotesRichTextItem*. Например, метод *AppendRTFile* позволяет присоединять файлы, а метод *EmbedObject* – создавать встроенные объекты, такие как электронные таблицы и презентации.

Метод *Timedifference* возвращает разницу между *cutoff* и *duedate* в секундах (*cutoff–duedate*). Свойство *DateTimeValue* возвращает значение даты-времени в элементе (т.е. фактически в поле). Этот метод применим только к элементам даты-времени. Если элемент другого типа, то возвращается *Nothing*.

Вопросы для самоконтроля

1. Как создать программу-агент?
2. Где можно просмотреть свойства программ-агентов?
3. Перечислите основные свойства программ-агентов.
4. Приведите примеры программ-агентов.
5. Сферы применения программ-агентов.
6. Перечислите основные методы класса *NotesRichTextItem*.
7. Расскажите о методе *TimeDifference* и свойстве *DateTimeValue*.
8. Алгоритм программы-агента (написанной на *Lotus Script*), отправляющей по электронной почте напоминания о невыполненном задании со ссылкой на незавершенный документ.

Задание

Создайте агента, подсчитывающего количество документов-ответов и записывающего эту информацию в основные документы. Поле с количеством ответов должно быть видимо в представлении с общим списком основных документов (создать новое представление), а также на форме без возможности редактирования этого поля.

Запуск агента должен осуществляться через кнопку-действие на представлении с общим списком. В конце работы агент обязан обновить все представления в базе данных.

Алгоритм выполнения задания:

1. Создать программу-агент;
2. В событии *Initialize* программы-агента записать необходимый код на языке *Lotus Script* (использовать принцип двойного перебора документов в базе данных, связь между родительскими и ответными документами по UNID);
3. Создать новое представление с общим списком (можно на основе уже существующего в базе данных);
4. Создать в представлении с общим списком кнопку-действие;
5. В кнопке-действии выбрать пункт *Simple Action(s)*;
6. Нажать кнопку *Add Action*, в открывшемся окне в поле со списком *Action* выбрать *Run Agent;*
7. Выбрать имя вызываемой программы-агента в поле *Select the agent to be run*;
8. Создать на главной форме поле, отображающее количество ответов для данного документа, тип поля – *Computed.* В формуле написать, например, **@Nothing**. Связь количества документов-ответов с полем будет осуществляться по имени поля, например, *RespCount*. Количество ответных документов для текущего документа подсчитывается в программе-агенте и заносится в поле *RespCount*;
9. В представлении, созданном в пункте 3, добавляем столбец, в котором будет отображаться поле, содержащее количество ответов;
10. На главной форме можно скрыть абзац с полем *RespCount*, если документ новый (@IsNewDoc, флажок *Hide paragraph if formula is true* поднят), и абзац с внедренным представлением, если документ новый или количество ответных документов для данного документа равно нулю.

Тема 4 БЕЗОПАСНОСТЬ В LOTUS DOMINO/ NOTES

4.1 Организация безопасности в Notes

Безопасность программных продуктов *Notes* основана на технологии шифрования **RSA Cryptosystem**, в которой используются двойные ключи (**RSA Cryptosystem –** это компания, основанная тремя математиками: Ривестом (Rivest), Шамиром (Shamir) и Эдельманом (Adelman)). Когда с помощью технологии *RSA* создается *id*-файл, то устанавливаются секретный и открытый ключи. Они хранятся в *id*-файле, а открытый ключ также хранится в каталоге *Domino*. Между двумя этими ключами устанавливается математическое соответствие, которое используется в так называемом процессе аутентификации. Но система безопасности – это не только шифрование. Существуют следующие уровни безопасности серверов **Domino**:

1. физический;
2. сервер;
3. база данных (см. раздел 4.2);
4. программа-агент;
5. форма (см. раздел 4.3);
6. представление (см. раздел 4.3);
7. документ;
8. поле.

Физический уровень. Естественно, компьютеры, серверы, прочее оборудование, на котором хранятся и обрабатываются данные, должны быть закрыты в надёжном помещении и доступны только обслуживающему персоналу.

Доступ к серверу в среде Notes подчиняется следующим правилам:

1. сервер может инициировать соединение с другим сервером;
2. клиенты могут запрашивать соединение к серверу;
3. сервера никогда не инициируют соединение с клиентом;
4. клиент не может напрямую соединиться с другим клиентом.

4.2 Установка уровней безопасности в Notes

Ответственность за работу с владельцами и пользователями приложений лежит на разработчике базы данных, именно он определяет права доступа пользователей приложения. *Владелец* – лицо или группа лиц, наделенная полномочиями, позволяющими им изменять дизайн приложений. Владельцы дизайна определяют пути и методы разработки. *Пользователи* – это не только те, кто

вводит данные в базу данных, но и те, кто пользуются информацией из приложения. Существуют также многочисленные типы пользователей с различными возможностями доступа. Типы доступа к приложению могут варьироваться от очень простого до очень сложного и строго контролируемого.

Для базы данных существует 7 уровней доступа. Привилегии доступа предоставляются отдельным лицам или группам в **ACL** (**Access Control List**) базы данных. Уровни безопасности баз данных, используемых в *Domino*, перечислены в таблице 4.1

Таблица 4.1 – Уровни безопасности баз данных Domino

Уровень	**Привилегии доступа**
No Access	Нет;
Depositor	Позволяет создавать и сохранять документы, но не позволяет изменять или читать документы (даже собственные);
Reader	Позволяет читать документы, но не позволяет создавать или изменять их;
Author	Позволяет создавать или изменять свои собственные документы. Может также читать другие документы, удалять свои собственные;
Editor	Может создавать новые документы, редактировать свои и чужие документы. Может также удалять документы;
Designer	Editor + возможность изменять проект базы данных;
Manager	Designer + право изменять ACL. Этим уровнем полномочий обладает также администратор сервера Домино. При создании БД этот уровень получает создавший ее пользователь.

Все перечисленные уровни присваиваются без различия типа пользователя – обычный пользователь, группа пользователей либо сервер. Необходимость настраивать доступ для сервера возникает, если в организации имеется несколько серверов Domino и есть необходимость обмениваться данными между ними. Для репликации данных может быть достаточно дать связанным серверам права уровня «*Editor*». Это позволит обмениваться документами, удалять их и изменять. Устанавливать уровень доступа «*Designer*» нужно только для серверов, с которых планируется менять структуру БД.

Дополнительная степень детализации уровней доступа может быть установлена с помощью *ролей* и специальных полей. Если приложение содержит несколько баз данных, то необходимо продумывать *ACL* для каждой из них.

Уровней доступа бывает достаточно не всегда. Например, автор не может изменить свой собственный документ до тех пор, пока форма не содержит в по-

ле типа авторских данных имя этого автора. Никто на любом уровне *ACL* не может удалить документ из базы данных, если не установлен флажок *Delete Documents*.

Право доступа читателя можно уточнить с помощью поля типа *Reader Names* (имен читателей). Если документ содержит это поле, и в нем нет имени пользователя или группы, в которой пользователь состоит, то данный пользователь не сможет прочитать документ. На самом деле пользователь даже не будет знать о существовании документа.

Точно также в полях имен авторов можно ограничить право создавать и редактировать документы (авторское право), уточнив список пользователей или групп, которые могут создавать и редактировать документы.

4.3 Ограничение использования форм и представлений с помощью ролей

Роли используют, чтобы более точно настроить ограничения на доступ к базе данных. Роли создаются в *ACL* и используются таким же образом, как имена пользователей в формах и представлениях.

Пользователи распределяются по ролям точно так же, как и по группам. Отличительной чертой в этом случае является то, что группе тоже можно назначить роль. На самом деле, роль можно определить для любого субъекта из перечисленных. Но это является и недостатком ролей. Поэтому если определенную роль необходимо назначить только некоторым пользователям из группы, то приходится их имена указывать в *ACL* явно, а не в составе группы.

Роли появились в четвертой версии *Lotus Notes*.

Диалоговое окно *ACL* имеет 4 кнопки (закладки) в левой части окна: **Basic** (основные сведения), **Roles** (роли), **Log** (журнал), **Advanced** (дополнительные сведения).

Чтобы создать новую роль в базе данных необходимо:

1) Открыть окно *ACL* базы данных (*File*→ *Database*→ *Access Control*…);
2) Щелкнуть на закладке *Roles*;
3) Щелкнуть на кнопке *Add*;
4) Ввести имя новой роли и щелкнуть на кнопке *OK*;
5) Щелкнуть на кнопке *OK* для закрытия окна *ACL*.

Роли, заключенные в квадратные скобки, отображаются в окне *Roles* вкладки *Roles* и в нижнем правом окне *Roles* вкладки *Basic*. В базу данных можно добавить до 75 ролей. Имена ролей могут состоять из чисел, букв и про-

белов. Первой в имени роли может идти цифра, пробелы лучше вообще не использовать (рекомендация *IBM*).

После того, как роль добавлена в базу данных, ее можно использовать для ограничения доступа и возможности изменения проекта базы данных. Как формы, так и представления в окне *Properties* имеют вкладку *Security*, в которой можно использовать роли. Она очень полезна, если в базе данных есть набор документов, в которых хранятся значения, обновляемые только небольшой группой пользователей (например, список, используемый в ключевом поле). Можно ограничить право доступа создания к форме, в которой хранятся списки, и право доступа чтения к представлению, которое показывает документы, созданные по этой форме, для группы пользователей. Значения из форм можно получить с помощью команд **@Column** и **@LookUp**, заданных для скрытых представлений.

Чтобы ограничить доступ к форме с помощью роли необходимо выполнить следующее:

1. Открыть форму в режиме проекта;
2. Открыть окно свойств формы *Properties* и щелкнуть на вкладке *Security* (вкладка с ключом);
3. Отменить отметку в поле *All authors and above* (все авторы и выше) в разделе *Who Can Create Documents with This Form* (кто может создавать документы с помощью этой формы) окна *Properties*;
4. Щелкнуть на роли в поле со списком и сохранить форму.

Таким же образом право доступа чтения к документам, созданным с помощью данной формы, можно ограничить, отменив установку флажка *All readers and above* (все читатели и выше) в разделе *Default Read Access for Documents Created with This Form* (стандартное право доступа для чтения документов, созданных по этой форме) окна *Properties*.

Для назначения роли пользователю или группе необходимо на вкладке Basic окна *ACL* отметить группу или отдельного пользователя в *ACL* и щелкнуть на соответствующей роли в поле списка *Roles*. Возле этой роли появится отметка.

В системе безопасности на уровне представлений роли используются точно так же. **Чтобы ограничить доступ к представлению с помощью роли** необходимо:

1. Открыть представление в режиме проекта;
2. Щелкнуть на вкладке *Security* окна *View Properties*;
3. Отменить установку флажка *All readers and above* (все читатели и выше) в разделе *Who may use this view* (может использоваться);

4. Щелкнуть на роли в поле со списком и сохранить представление.

Роли – это не единственное средство ограничения доступа к формам и представлениям. Таким же образом можно разрешить доступ группе или отдельному пользователю. Более того, **использование ролей не отменяет действия ACL.** Если пользователь имеет к базе данных право доступа *Reader*, то он может только читать документы. Даже если для данного пользователя добавить право доступа создания для формы, то он все равно не сможет создавать документы с помощью этой формы.

С помощью ролей можно также ограничить право доступа чтения и редактирования к личным документам. Для этого следует добавить роль в поле *Reader Name* или *Author Name*. **Имя роли следует заключать в двойные кавычки и квадратные скобки**, например, **"[Creator]"**.

4.4 Скрытие кнопок и других объектов в зависимости от роли

Для работы с уровнями доступа используются следующие функции языка формул: **@UserAccess**, **@UserRoles**, **@IsMember**, **@IsNotMember**. Их можно использовать на закладке *Hide When* для написания формулы в окне формулы, с установленным флажком *Hide…if formula is true* (вместо … отображается имя объекта, например, *Action*).

1) функция **@UserRoles**

Область применения: нельзя использовать в формулах отбора, колонок и фоновых агентов. Функция предназначена для использования в базах данных, расположенных на сервере, или для локальных баз данных с установленным флажком "*Enforce a Consistent Access Control List across all Replicas*" (использовать единый *ACL* для всех реплик). Это свойство находится на вкладке *Advanced* окна *ACL*. Для локальных баз данных без этого свойства функция всегда возвращает пустую строку.

Функция **@UserRoles** возвращает текстовый список имен ролей, на которые в ACL назначен текущий пользователь. Имена ролей заключаются в квадратные скобки.

2) функция **@IsMember**

Область применения: без ограничений.

Функцию можно записать в нескольких форматах:

@IsMember (строка; список_строк)
@IsMember (список_строк1; список_строк2)

Возвращает **1(True)**, если строка является элементом списка строк; возвращает **0(False)** в противном случае. Если оба параметра являются списками – возвращает **1(True)**, если все элементы списка_строк1 содержатся в списке_строк2. **Функция учитывает регистр.**

Пример.

@IsMember(”экзамен”; ”зачет”: ”экзамен”: ”курсовая”) – возвращает 1.

@IsMember(”студент”: ”спит”; ”студент”: ”учится”: ”работает”) – возвращает 0.

3) функция **@IsNotMember**

Область применения: без ограничений.

Функцию можно записать в нескольких форматах:

@IsNotMember (строка; список_строк)

@IsNotMember (список_строк1; список_строк2)

Возвращает **1(True)**, если строка не является элементом списка строк; возвращает **0(False)** в противном случае. Если оба параметра являются списками – возвращает **1(True)**, если ни один из элементов списка_строк1 не содержится в списке_строк2. **Функция учитывает регистр.**

Пример.

@IsNotMember(”экзамен”; ”зачет”:”экзамен”:”курсовая”) – возвращает 0.

Замечание. Если оба аргумента являются списками строк, то справедливо: **not @IsMember ≠ @IsNotMember**.

Пример. Удалять документы из представления может только администратор (роль). В окне формулы на закладке *Hide When* для кнопки-действия запишем (флажок *Hide Action If Formula is True* установлен)

@IsNotMember(”[administrator]”;**@UserRoles**)

4) функция **@UserAccess (**сервер: база данных; флаги)

Область применения: нельзя использовать в формулах отбора, колонок, фоновых агентов и всплывающих окон. Для локальных баз данных в варианте синтаксиса без второго параметра и со сброшенным в ACL флажком *Enforce a Consistent Access Control List across all Replicas* всегда возвращает значение 6:1:1:1:1:1:1:1:1. Если пользователь не имеет доступа к базе данных, то он получит сообщение ”*You are not authorized to perform that operation*”.

Функция возвращает числовой список, элементы которого позволяют определить уровень доступа текущего пользователя к указанной базе данных. Пустая строка в качестве параметра *сервер* означает, что база данных расположена локально.

Необязательный параметр *флаги* позволяет определить уровень доступа для конкретного элемента, а не получать весь список доступа, и затем выделять из него требуемое значение. Параметр *флаги* может представлять собой список, состоящий из следующих значений:

1) [**AccessLevel**] – возвращает число от 1 до 6, определяющее уровень доступа пользователя к базе данных. 1 соответствует уровню доступа депозитора, 2 – читателя, 3 – автора, 4 – редактора, 5 – дизайнера, 6 – менеджера;

2) [**CreateDocuments**] – возвращает 1(true), если пользователь может создавать документы, 0(false) в противном случае;

3) [**DeleteDocuments**] – возвращает 1(true), если пользователь может удалять документы, 0(false) в противном случае;

4) [**CreatePersonalAgents**] – возвращает 1(true), если пользователь может создавать личные агенты, 0(false) в противном случае;

5) [**CreatePersonalFoldersAndViews**] – возвращает 1(true), если пользователь может создавать личные представления и папки, 0(false) в противном случае;

6) [**CreateSharedFoldersAndViews**] – возвращает 1(true), если пользователь может создавать общие представления и папки, 0(false) в противном случае;

7) [**CreateLotusScriptJavaAgents**] – возвращает 1(true), если пользователь может создавать агентов на языках *Lotus Script, Java*, 0(false) в противном случае;

8) [**ReadPublicDocuments**] – возвращает 1(true), если пользователь может читать общие документы, 0(false) в противном случае;

9) [**WritePublicDocuments**] – возвращает 1(true), если пользователь может создавать общие документы, 0(false) в противном случае;

10) [**ReplicateOrCopyDocuments**] – возвращает 1(true), если пользователь может реплицировать или копировать документы, 0(false) в противном случае.

Если несколько флагов задано в виде списка, то возвращается список значений, соответствующих указанным флагам. Если не выбран ни один флаг, то возвращаемый список имеет формат: 1):2):3):4):5):6):7):8):9). Значение для 10-ого флага в варианте синтаксиса без второго параметра отсутствует.

Пример. В локальной базе данных для текущего пользователя в *ACL* установлена опция **Editor**, флаг ”*Replicate or copy documents*” опущен, а флаг ” *Enforce a Consistent Access Control List across all Replicas*” поднят. Тогда формула поля

@UserAccess(@DbName;[ReplicateOrCopyDocuments]:[AccessLevel])

возвратит **0:4**.

Вопросы для самоконтроля

1. Что такое **RSA**?
2. Перечислите уровни безопасности серверов *Domino*.
3. Перечислите уровни доступа для базы данных.
4. Зачем используются поля *Reader Names* и *Author Names*?
5. Что такое роль? Как ее создать?
6. Как ограничить доступ к форме с помощью ролей?
7. Как ограничить доступ к представлению с помощью ролей?
8. Как назначить роль пользователю или группе пользователей?
9. Какие функции языка формул используются для работы с уровнями доступа?
10. Функция **@UserRoles**. Особенности ее использования.
11. Функции **@IsMember**, **@IsNotMember**. Особенности их использования.
12. Особенности использования функции **@UserAccess**. Флаги.

Задание

Создайте в учебной базе данных роли «*Administrator*» и «*Editor*». Измените логику базы данных в соответствие со следующим описанием. Пользователь без ролей не должен видеть ни одного действия по созданию, редактированию и удалению документов. Если у пользователя нет роли «Administrator», то он не должен видеть действий по удалению документов и запуску агентов.

Алгоритм выполнения задания:

1. Добавить роли «*Administrator*» и «*Editor*» в *ACL*;
2. Для каждой из кнопок-действий форм и представлений, отвечающих за создание, редактирование, удаление документов и запуск агентов, прописать соответствующие формулы на языке формул (см. раздел 4.4) в окне формул на закладке *Hide When* с установленным флажком *Hide Action if formula is true*;
3. На закладке *Advanced* окна свойств *ACL* установить флажок *Enforce a Consistent Access Control List across all Replicas*;
4. Назначить пользователю (только не себе!) роль «*Administrator*», убедиться в правильности работы базы данных;
5. затем назначить пользователю роль «*Editor*», проверить работу базы данных;
6. затем не назначать пользователю никаких ролей, проверить работу базы данных в этом случае.

ЛИТЕРАТУРА

1 Березина, Н. С. Начальный курс Lotus Notes R5 / Н. С. Березина, Е. Н. Трубникова. – М. : Интертраст, 2002. – 285 с.

2 Ионцев, Н. Н. Почтовая система сервера Lotus Domino R5 и ее конфигурирование / Н. Н. Ионцев. – М. : Интертраст, 2001. – 135 с.

3 Ионцев, Н. Н. Программирование в Lotus Domino R.5.: формулы и функции, язык LotusScript, встроенные классы LotusScript и Java / Н. Н. Ионцев, Е. В. Поляков, О. Г. Таранченко. – М. : Интертраст, 1999. – 456 с.

4 Керн, С. Lotus Notes и Domino 6. Руководство разработчика / С. Керн [и др.]; пер. с англ. – К. : ООО «ТИД «ДС», 2005. – 880 с.

5 Кирклэнд, Р. Domino 5 & 6. Администрирование сервера / Р. Кирклэнд; пер. с англ. – М. : ДМК Пресс, 2003. – 832 с.

6 Кузьменков, Д.С. Lotus Domino/Notes: практическое руководство для студентов математического факультета специальностей 1-31 03 03-02 «Прикладная математика (научно-педагогическая деятельность)», 1-31 03 06-01 «Экономическая кибернетика (математические методы и компьютерное моделирование в экономике)», 1-40 01 01 «Программное обеспечение информационных технологий»: в 2 ч. Ч.1 / Д. С. Кузьменков, М-во образования РБ, Гомельский государственный университет им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2011. –48 с.

7 Линд, Д. Lotes Notes и Domino 5/6. Энциклопедия программиста / Д. Линд, С. Керн; пер. с. англ. – 2-е изд., перераб. и доп. – К. : ООО «ТИД ДС», 2003. – 1024 с.

8 Поляков, Е. В. Изучение новых возможностей IBM Lotus Domino Designer 6 / Е. В. Поляков. – М. : Интертраст, 2002. – 245 с.

9 Поляков, Е. В. Средства разработки приложений в Lotus Domino R5: Domino Designer/ Е. В. Поляков. – М. : Интертраст, 2003. – 467 с.

10 Поляков, Е. В. Язык @-формул в Lotus Domino R6. Справочник разработчика / Е. В. Поляков. – М. : Интертраст, 2004. – 347 с.

11 Поляков, Е. В. Domino Designer R6.5 – интегрированная среда разработки приложений в Lotus Domino: учебное пособие для вузов / Е. В. Поляков. – М. : Интертраст, 2005. – 640 с.

12 IBM Corp. Lotus Domino 6. Administering the Domino System, Volume 1. – IBM Corp., 2002 – 1354 p.

13 IBM Corp. Lotus Domino 6. Administering the Domino System, Volume 2. – IBM Corp., 2002 – 1150 p.

14 IBM Corp. Lotus Domino 6. Installing Domino Servers. – IBM Corp., 2002 – 187 p.

15 Speed, T. Lotus Notes Domino 8: Upgrader's Guide / T. Speed, D. McCarrick, B. Gibson. – Packt Publishing, 2008. – 276 p.

Учебное издание

КУЗЬМЕНКОВ Дмитрий Сергеевич
РОДИОНОВ Андрей Александрович

LOTUS DOMINO/NOTES

ПРАКТИЧЕСКОЕ РУКОВОДСТВО

по выполнению лабораторных работ

для студентов математического факультета специальностей
1-31 03 03-02 «Прикладная математика (научно-педагогическая деятельность)»
1-31 03 06 01 «Экономическая кибернетика (математические методы
и компьютерное моделирование в экономике)»
1-40 01 01 «Программное обеспечение информационных технологий»

В 2-х частях
Часть 2

В авторской редакции

Подписано в печать 31.10.2011 г. (11). Формат издания 60*80 1/16.
Бумага офсетная. Гарнитура «Таймс».
Усл. печ. л. 1,16. Уч.-изд. л. 1,25. Тираж 25 экз.

Отпечатано в учреждении образования
«Гомельский государственный университет
имени Франциска Скорины»,
246019, г. Гомель, ул. Советская, 104.