Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

Д. С. КУЗЬМЕНКОВ, Е. Ю. КУЗЬМЕНКОВА

КОМПЬЮТЕРНЫЕ СИСТЕМЫ УПРАВЛЕНИЯ ДОКУМЕНТООБОРОТОМ И WEB-ТЕХНОЛОГИИ

Практическое руководство

для студентов специальностей
1–31 03 03–02 «Прикладная математика
(научно-педагогическая деятельность)»
1–40 01 01 «Программное обеспечение информационных технологий»

Гомель
ГГУ им. Ф. Скорины
2017

УДК 004.43 (076)
ББК 32.973.5я73
К893

Рецензенты:
кандидат физико-математических наук Т. В. Тихоненко,
кандидат технических наук С. Ф. Маслович

Рекомендовано к изданию научно-методическим советом учреждения образования «Гомельский государственный университет имени Франциска Скорины»

Кузьменков, Д. С.

К893 Компьютерные системы управления документооборотом и web-технологии : практическое руководство / Д. С. Кузьменков, Е. Ю. Кузьменкова ; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель : ГГУ им. Ф. Скорины, 2017. – 46 с.
ISBN 978-985-577-309-3

Практическое руководство предназначено для оказания помощи студентам в овладении практическими приемами и навыками проектирования приложений в среде Lotus Domino/Notes (в одной из сред для создания современных компьютерных систем управления документооборотом), в разработке web-приложений различной сложности.

Адресовано студентам 2 курса специальностей 1–31 03 03–02 «Прикладная математика (научно-педагогическая деятельность)», 1–40 01 01 «Программное обеспечение информационных технологий».

УДК 004.43 (076)
ББК 32.973.5я73

ISBN 978-985-577-309-3

Оглавление

Предисловие

Практическое руководство призвано помочь студентам овладеть основами современных компьютерных технологий и программного обеспечения, проектирования и разработки приложений в среде Lotus Domino/ Notes, приобрести навыки работы с одной из самых современных сред разработки компьютерных систем управления документооборотом, овладеть основами технологии разработки web-приложений XPages.

Издание структурно содержит четыре темы. В первой теме дается описание среды Lotus Domino/Notes, указываются её преимущества и недостатки. Вторая тема посвящена объектно-ориентированному языку программирования Lotus Script. В третьей теме описывается работа с программами-агентами, приводятся типичные примеры программ-агентов, написанных на языке Lotus Script и на языке формул. Последняя тема посвящена технологии разработки web-приложений XPages. В этой теме рассмотрены преимущества технологии XPages, создание приложений в XPages, работа с документами в XPages, использование нескольких страниц, различные элементы управления – строительные блоки XPages, настройка различных свойств XPages. В конце каждой темы есть список вопросов для самоконтроля и задания, направленные на закрепление приведенной в теме информации (кроме первой темы), приведены также алгоритмы выполнения заданий.

Практическое руководство «Компьютерные системы управления документооборотом и web-технологии» направлено на формирование умений и навыков в проектировании приложений в среде Lotus Domino/ Notes, на усвоение современной компьютерной технологии разработки web-приложений различной сложности с использованием технологии XPages.

Среда Lotus Domino/Notes и возможности создания web-приложений в ней изучаются студентами 2 курса специальностей 1–31 03 03–02 «Прикладная математика (научно-педагогическая деятельность)» в рамках дисциплины «Технологии электронного документооборота», 1–40 01 01 «Программное обеспечение информационных технологий» в рамках дисциплины специализации «Компьютерные системы управления документооборотом и web-технологии».

Данное издание может быть использовано преподавателями при проведении практических занятий и студентами в их самостоятельной работе над дисциплиной.

Тема 1. Краткое описание Lotus Domino / Notes

Lotus Domino / Notes – это клиент-серверная система, то есть существует сервер (Domino), который хранит информацию и авторизует пользователей, выполняет серверную логику приложений (запускает запрашиваемые клиентским приложением службы); есть клиенты (Notes), которые выполняют бизнес-логику серверных приложений.

Основная парадигма Lotus – это документарный подход. Если говорить о современных подходах в реляционной технологии, то аналогией является хранение данных в виде XML, который описывает документ в виде набора «поле-значение» и хранится в одном поле таблицы. Lotus работает не с реляционными данными, его основной объект – это документ. Lotus работает с документами совершенно различной структуры (финансовые отчеты, распоряжения, докладные записки и т. д.). Он позволяет эффективно работать с такими данными.

1.2 Слабые стороны Lotus

1. Из-за документарного подхода Lotus не удобен для построения отчетов; на Lotus очень сложно и долго строить многомерный параметризуемый отчет; но есть связка Lotus – RDBMS, где Lotus используется в качестве системы сбора и обработки информации, RDBM-Storage в качестве системы построения отчета.

2. В Lotus практически отсутствует понятие транзакции, то есть если вам необходимо выполнить логическую совокупность действий, как одно целое, и в случае падения операции в середине все откатить, то у вас это стандартными методами не получится. Поэтому на Lotus не рекомендуется строить финансовые системы, которые рассчитаны не на учет, а именно на перемещение средств.

1.1 Достоинства Lotus

1. ***Репликация*** – это синхронизация данных между копиями одного и того же приложения.

Пример. Есть компания, у которой множество филиалов в разных городах. В компании функционирует система, в которую вбиваются и в которой согласуются финансовые проводки. Есть центральный офис, в который эти проводки отовсюду поступают и в котором они либо подтверждаются и отправляются в филиалы на выполнение, либо отклоняются.

Как решается задача территориальной распределённости в Lotus?

В каждом филиале ставится сервер domino, на котором в каждом филиале ставится реплика базы. Пользователи каждого филиала работают с

базой на своем сервере. А сервера по расписанию, например раз в час, обмениваются всеми поступившими изменениями и новыми документами. При этом параметры репликации можно задать таким образом, что множество документов, поступающих с сервера на сервер, будет ограничено – например, в центральный офис будут поступать документы со всех филиалов, а в филиале будут присутствовать только его документы.

При такой схеме в Гомеле бухгалтер вносит планируемую проводку, в состоянии черновик и отправляет ее на рассмотрение, через час она появляется в главном офисе, там он утверждается или отклоняется, и информация об этом изменении еще через час достигает сервера в Гомеле. Можно, и не раз в час, реплицировать обновления, при кластерной репликации обновления распространяются практически мгновенно.

2. ***Накат дизайна***. Дизайн базы – слой бизнес-логики, который хранится в виде специфичных документов. Все обновления дизайна также реплицируются между серверами. Эта парадигма сильно облегчает задачу перехода к новым версиям. База с новой версией просто объявляется источником дизайна, а база со старой версией дизайна – наследником, после чего запускается накат дизайна, и все обновления поступают в работающее приложение.

3. ***Клиент Lotus Notes является облегченной версией сервера***, потому при отсутствии постоянной связи с сервером можно сделать себе локальную реплику базы, работать в ней и отреплицировать их с сервером при появлении связи. Это дает широкие возможности для создания единой информационной системы для компаний, специфика которых подразумевает частые командировки сотрудников.

4. ***Защита информации.*** При регистрации пользователя сервер создает для него id-файл, в котором хранятся публичные и приватные ключи, а также электронная почта этого пользователя. Для обращения к серверу пользователь должен авторизоваться, используя этот id-файл. Все изменения, которые пользователь вносит в документы, подписываются им. Даже администратор не сможет эмулировать, что документ был сохранен от имени конкретного пользователя. Весь трафик между серверами и клиент-сервером шифруется. Локальные реплики баз также могут быть зашифрованы.

5. Lotus тесно интегрирован с ***почтовой системой***. Domino сам является SMTP-сервером, который дополнен рядом возможностей для работы с внутренней корреспонденцией. Любой документ внутри Lotus может быть отправлен по почте, как в виде письма, так и в виде той формы, по которой он был создан.

6. Domino также является ***HTTP-сервером***. Согласно бизнес-логике приложения сервер генерирует HTML для данных, которые на нем

хранятся. Поэтому для большинства приложений возможна реализация как интерфейса в Lotus клиенте, так и через web-браузер;

7. Lotus является ***интегрирующей платформой***, то есть позволяет организовывать обмен данными со множеством различных систем, функционирующих на базе другой платформы. Он поддерживает связку OLE-объектами, что позволяет хранить в Lotus-документах объекты Microsoft Office и пользуются всей их функциональностью.

1.1. Система управления документоориентированой базой данных

С задачей поиска нужных документов, так или иначе, связаны 30 % перемещений сотрудников по офису, в общей сложности этот процесс отнимает у них около одного месяца в год, причем 15 % бумажных документов безвозвратно теряются. На согласование документов уходит 60–70 % рабочего времени. В свете выше указанного, 20–30 % поставленных задач вообще не решаются. Все эти проблемы призвана решить система управления документооборота, организованная с помощью Lotus Notes.

В среде Lotus Notes можно разработать любую базу данных, позволяющую работать с документами. Прежде всего, это нефинансовый документооборот. Большая часть документооборота в Lotus Notes признана эквивалентной статусу бумажных документов.

Если отталкиваться от реальной практики использования, то можно выделить четыре основные сферы применения среды Lotus:

1) организации, которым нужна современная надежная инфраструктура электронной почты, передачи сообщений и коммуникаций;

2) организации, которые используют Lotus в качестве платформы и инфраструктуры для бизнес-приложений, автоматизации деловых процедур, документооборота и т. д.;

3) организации, выбирающие Lotus Domino в качестве уникальной технологии для создания инфраструктуры Web;

4) организации, выбирающие Lotus в качестве интегрирующего программного обеспечения, способного интегрировать информацию и данные практически из произвольных источников информации – реляционных СУБД, систем управления ресурсами предприятий (ERP), сети Internet и т. д.

Вопросы для самоконтроля

1. Что представляет собой среда Lotus?
2. Какие слабые стороны есть у среды Lotus?
3. Перечислите достоинства Lotus Domino/Notes.
4. Что такое репликация?

5. Что такое накат дизайна?
6. Перечислите 4 основные сферы применения Lotus Domino/Notes.
7. Расскажите о защите информации в Lotus Domino/Notes.

Тема 2. Краткий обзор языка программирования Lotus Script

Lotus Script – это объектно-ориентированный язык программирования, совместимый с языком Basic. Применяя 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 (*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.

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

Идентификаторы в Lotus Script ***нечувствительны к регистру***. Синтаксис языка программирования Lotus Script подробно описан в [4, п.1.1].

2.1 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).

2.2 Функции, подпрограммы, объекты, классы и события

В Lotus Script существует ряд предопределенных встроенных функций (например, *Input*, *Month*, *Today*), 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)
'вызываем подпрограмму печатающую значение
MySub (MyVar)

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 (приложение), 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 присваивают 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, что и в большинстве современных языков программирования (см. [5] с. 531–533).

2.3 Иерархия классов. Клиентские классы

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

Переменная объектной ссылки создается в два этапа:

1. Объявление переменной.
2. Создание экземпляра объекта.

Пример. Создадим объектную ссылку.

```
Dim db as NotesDatabase
Set Doc = New NotesDocument(db)
```

Этот оператор создает объект класса *NotesDocument* (новый документ в приложении), 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, относящимися 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 (приложения).

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 содержащие их библиотеки.

2.4 Класс NotesDocument

Свойства этого класса позволяют узнать любую информацию о документе. Например, дату создания, значения, отображаемые в столбцах представления в строке документа (если документ был открыт из представления), встроенные в него объекты, размер документа и т. д. Методы этого класса позволяют выполнять различные действия над любыми элементами документа, делать данный документ ответным, помещать в папку или удалять его из папки, отправлять его по почте или сохранять.

Если создается новый объект *NotesDocument* и в него не добавляются никакие элементы, то документ не будет сохранен, если воспользоваться методом *Save*.

В языке Lotus Script поля называются элементами. Поля создаются в форме для ввода данных. Затем посредством интерфейса пользователя с помощью формы создается документ. Программа-клиент вводит информацию в поля формы, и эта информация сохраняется в документе. Вместе с данными в документе запоминается информация о самом поле. Эти данные и информация относительно поля образуют ***элемент***. С помощью Lotus Script данные могут запоминаться в элементах документа, и эти элементы будут отображаться на экране только в том случае, если в форме, используемой для отображения документа, будет соответствующее поле.

При создании документа с использованием языка Lotus Script и объекта *NotesDocument* в поле *Form* заносится информация о том, какая форма будет использоваться при открытии документа пользователем. Если затем этот документ сохраняется, то не будет выполнена ни одна формула поля в этой форме, т. к. документ создавался на сервере, а форма не была открыта, то связь между данными документа и формулой не будет установлена. Поэтому строка документов в представлении, возможно, не будет отображаться нужным образом. После открытия документа с помощью интерфейса пользователя и его сохранения, формулы полей будут работать. При создании документов на сервере необходимо программно установить значения всех вычисляемых полей или использовать метод *ComputeWithForm*, чтобы вызвать выполнение формул.

В языке Lotus Script при создании документа всегда необходимо устанавливать значения поля *Form*. Информация в него заносится так: **`doc.form`** или **`uidoc.setdocfield.`**

Метод *Send* класса *NotesDocument* используется для отправления документов по электронной почте. Этому методу необходимо передать, по крайней мере, одно значение *Sendto*, с помощью аргумента *attachForm* метода *Send* можно отправить форму вместе с документом.

Метод *Save* класса *NotesDocument* имеет 3 аргумента: *Force*, *CreateResponse*, *MarkRead*. Аргумент *Force* имеет тип *boolean*. Если **`Force =`**

True (значение параметра *CreateResponse* в этом случае не учитывается) и кто-нибудь сохранил документ в то время, когда его модифицировали, то Notes удаляет сохраненный документ, записывая на его место новый вариант документа. Если **Force = False**, то предпринимаемые действия записываются вторым аргументом *CreateResponse* (тип *Boolean*). Если **Force = False**, а **CreateResponse = True**, то текущий документ становится ответом на документ, отредактированный и сохраненный предыдущим пользователем. Если **Force = False**, и **CreatResponse = False**, то сохранение нового варианта документа отменяется. Аргумент *MarkRead* определяет, будет ли сохраненный документ отмечен для чтения текущим пользователем.

2.5 Взаимодействие с пользователем: функции MessageBox, InputBox

Функция ***MessageBox*** выводит на экран информацию для пользователя.

Функция ***InputBox*** обеспечивает ввод пользователем информации в программу.

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. Функция *MessageBox* с двумя параметрами.

```
RetCode=MessageBox("Продолжить?", MB_YESNOCANCEL)
```

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

Существует и оператор *MessageBox*, который в отличие от функции не возвращает никакого значения.

Пример. Оператор *MessageBox* с двумя параметрами

```
MessageBox "ошибка", MB_OK
```

Формат описания 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 2.1)).

Таблица 2.1 – Возможные значения аргумента Buttons

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

Аргумент *Icon* – какая пиктограмма может отображаться в окне сообщения (см. в таблице 2.2). Аргумент *Default* – какая кнопка будет считаться нажатой по умолчанию, если пользователь нажимает пробел или ввод (см. в таблице 2.3). Аргумент *Mode* – 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 2.4. Аргумент *BoxTitle* – строковая переменная, длиной до 128 символов, значение которой отображается в области заголовка окна.

Таблица 2.2 – 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 2.3 – 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 2.4 – Возможные значения аргумента Mode

Имя константы	Значение	Режим
MB_APPLMODAL	0	Приложение
MB_SYSTEMMODAL	4 096	Система

Пример. Три способа кодирования окна сообщения.

```
ret=MessageBox("продолжить?",4387,"ошибка!")
```

```
ret=MessageBox("продолжить?",3+32+256+4096,"ошибка!")
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 2.5).

Таблица 2.5 – Значения, возвращаемые функцией *MessageBox*

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, длиной 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. Функция *InputBox* с тремя параметрами.

```
InputBox$("введите количество гаражей","число",30)
```

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

2.6 Сценарии Lotus Script для форм

Сценарии *Lotus Script* для форм – это функции и подпрограммы, которые совместно используются любыми объектами формы, объявленными в событии (*globals*), или программой обработки событий на уровне формы: *Initialize*, *QueryOpen*, *PostOpen*, *PostRecalc*, *QuerySave*, *QueryModeChange*, *PostModeChange*, *QueryClose*, *Terminate*.

Пример. Определим значения, заданные по умолчанию для полей *name* и *date* (событие *PostOpen*).

```
Sub PostOpen (Source As Notesuidocument)
    Dim s As New NotesSession
    Diw ws As New NotesUIWorkspace
    Dim note As NotesDocument
    Dim theDate As New NotesDateTime(now)
    Set source = ws.CurrentDocument
    Set note=Source.Document
    If Source.IsNewDoc Then
        Note.Name=s.Username
        Set Note.Date=theDate
        Source.reload
    End If
End Sub
```

Все события *Query* могут иметь параметр *continue*, который позволяет определить, следует ли продолжать выполнение текущего события до его завершения. Используя параметр continue, можно прекращать выполнение операции или события, если пользователь, например, ввёл некорректное значение поля.

Пример. С помощью сценария события *QueryOpen* можно создать механизм проверки, с помощью которого при открытии документа в приложении проверяется файл пользователя Notes.ini на наличие в нём определённого значения. Если это значение в файле отсутствует (например, пользователь должен запускать программу-агент или щёлкнуть на кнопке, чтобы произвести необходимые настройки перед работой с приложением), то документ не открывается.

```
Sub QueryOpen (Source As Notesuidocument, Mode As In-
    teger, IsNewDoc As Variant, Continue As Variant)
    Dim s As New NotesSession
    Dim iniVar As Variant
    Inivar=s.GetEnvironmentSting("$DBUserInfo")
    If inivar=" " Then
        messageBox "ВЫ должны нажать кнопку зарегист-
        рироваться для просмотра документов!"
        continue=false
    End If
End Sub
```

Сначала создаём объект класса *NotesSession*, затем с помощью метода *GetInvironmentString* извлекаем значение переменной *DBUserInfo*. Если

оно не установлено или отсутствует, то **continue=false** и на экран выводится сообщение, поясняющее пользователю, почему не был открыт документ. Если же значение установлено, то *continue* остается *true* и документ будет открыт.

Пример. Узнаём, сколько раз пользователь открывал тот или иной документ. Для этого в сценарии события *QueryClose* проверяется, находится ли документ в режиме редактирования. Если да, то значение специального скрытого поля (оно сохраняется вместе с документом), увеличивается на 1. Если документ не редактируется, то происходит переключение в режим редактирования, увеличивается на 1 счетчик того, сколько раз документ читался, затем документ сохраняется, при этом пользователь ничего не замечает.

```
Sub QueryClose (Source As Notesuidocument, Continue
    As Variant)
    Dim ntr As Long
    Dim Note As NotesDocument
    Set note=Source.Document
    If note.IsNewNote Then
        Ntr=0
    Else ntr=clng(note.NumTimesRead(0))
    End If
    ntr=ntr+1
    Call note.ReplaceItemValue("NumTimesRead",
    cstr(ntr))
    Call note.Save(True, True)
End Sub
```

Clng возвращает значение, преобразованное к типу *long*, *Cstr* возвращает значение, преобразованное в строку.

В рассмотренном выше примере действия производятся не над клиентским документом, хотя он является исходным документом в данной программе (используем объект *Source* класса *NotesUIDocument*). В программе происходит переход от клиента к серверу, для чего объектной переменной *note* присваивается значение свойства *Document* источника. Все действия осуществляются на сервере, т. к. это проще и программный код определяет событие *QueryClose* (пользователь находится в режиме закрытия документа), поэтому нет необходимости выполнять какие-либо действия в документе клиента. Далее проверяем, не является ли документ новым, если это так, то счётчик обнуляем, иначе присваиваем ему значение поля *NumTimesRead*, в котором хранится число обращений к данному документу. Так как изменения осуществляются для сервера, то

нет необходимости знать, в каком режиме (редактирования или чтения) находится клиентский документ; так как если документ не редактировался, то сервер сам переключит его в режим редактирования, и пользователь ничего не заметит (см. в постановке примера). Далее увеличим счётчик на 1, независимо от того, это новый документ или нет. Сохраним полученное значение счётчика в поле *NumTimesRead* и сохраним документ сервера.

Пример. Проверим, заполнены ли два обязательных поля на форме. Для этого используем обработчик события *QuerySave*. Если поле не заполнено, то выведем пользователю сообщение и передадим фокус в незаполненное поле.

```
Sub Querysave(Source As Notesuidocument, Continue As
    Variant)
    If Len(Source.FieldGetText("Year_vyh")) = 0 Then
        Messagebox "ГОд выхода на экран не заполнен"
        source.GotoField "Year_vyh"
        continue = False
        Exit Sub
    End If
    If Len(Source.FieldGetText("Zhanr")) = 0 Then
        Messagebox "Жанр не выбран"
        source.GotoField "Zhanr"
        continue = False
        Exit Sub
    End If
End Sub
```

В данном примере **Exit Sub** используется, чтобы не выдавать сразу несколько сообщений, если не заполнены оба поля.

2.7 Сценарии Lotus Script для полей

Пример. Если в сценарии события Entering обнаруживаем, что числовое поле – пустое, то на экране появляется окно ввода. Введённое пользователем значение проверяем на правильность (функция **IsNumeric**). Если в поле имеется некоторое значение, то в сценарии ничего не происходит.

```
Sub Entering (Source As Field)
    Dim ws As New NotesUIWorkspace
    Dim uidoc As NotesUIDocument
    Set uidoc = ws.CurrentDocument
    If uidoc.FieldGetText("SomeField")="" Then
```

```
        Retcode=InputBox$("введите новое значение по-
        ля!")
        If IsNumeric(Retcode) Then
            Call uidoc.FieldSetText("SomeField", Ret-
            code)
        Else
            MsgBox "Введите число"
        Call uidoc.FieldSetText("SomeField", "")
        End If
    End If
End Sub
```

Пример. В сценарии события Exiting проверяем, не превышается ли значение поля 1000. Если это верно, то выводим сообщение, в котором напоминаем, что максимальное значение поля равно 1 000, и очищаем поле.

```
Sub Exiting(Source As Field)
    Dim ws As New NotesUIWorkspace
    Dim uidoc As NotesUIDocument
    Set uidoc=ws.CurrentDocument
    temp= uidoc.FieldGetText("SomeField")
    If temp<> "" Then
        If clng(temp)>1000 Then
            MessageBox "Значение поля не может превы-
            шать 1000!"
            Call uidoc.FieldSetText("SomeField", "")
        End If
    End If
End Sub
```

Пример. Используем обработчик события Exiting поля для проверки правильности введённого в поле значения.

```
Sub Exiting(Source As Field)
    Dim ws As New NotesUIWorkspace
    Dim uidoc As NotesUIDocument
    Set uidoc=ws.CurrentDocument
    fl=1
    Do While fl=1
        If   Isnumeric(uidoc.FieldGetText("Nom_film"))
        Then
            fl=0
        Else
```

```
            retCode=Inputbox$("Введите номер в ката-
            логе (число)",,100,100,30)
            If Isnumeric(retCode) Then
                Call    uidoc.FieldSetText("Nom_film",
                retCode)
                fl=0
            End If
        End If
    Loop
End Sub
```

2.8 Практическое задание

Напишите на *Lotus Script* 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 (как минимум у двух полей), *QuerySave* формы (также обрабатываются два или три поля). Для вывода сообщений необходимо использовать как функцию *MessageBox*, так и функцию *InputBox*.

Алгоритм выполнения задания

1. Написать в обработчике события *QuerySave* программный код, который проверяет, заполнены ли обязательные (помеченные звёздочкой, которая видна только в режиме редактирования) поля на форме (см. пример в п. 2.6). Для вывода сообщений используем функцию *MessageBox*;

2. У поля, обязательного для заполнения, заходим в *Default Value* и записываем там значение по умолчанию. Это необходимо для того, чтобы в случае, если пользователь не зайдёт в поле (т. е. не произойдёт событие *Entering* и *Exiting* соответственно), оно не осталось незаполненным;

3. Написать в обработчике события *Exiting* программный код для проверки правильности введённого в поле значения (см. п. 2.7). Для вывода сообщений используем функцию *InputBox*.

Вопросы для самоконтроля

1. Какие существуют особенности работы с полями в Lotus Script?
2. Как описываются функции и процедуры в Lotus Script?
3. На какие две большие группы разделяются все классы Domino?
4. Как правильно объявить объектную ссылку?
5. Перечислите основные клиентские классы.
6. Перечислите основные методы класса *NotesDocument*.

7. Какие параметры есть у метода Save класса *NotesDocument*?

8. Запишите формат описания и укажите область применения функции *MessageBox*.

9. Запишите формат описания и укажите область применения функции *InputBox*.

10. Какие события используются в сценариях Lotus Script для форм?

11. Каковы особенности использования событий, имена которых начинаются с Query.

12. Какие события наиболее часто используются в сценариях Lotus Script для полей?

13. Приведите примеры использования событий формы и поля для проверки правильности введённого пользователем значения.

Тема 3. Создание программ-агентов

3.1 Работа с программами-агентами

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

Рассмотрим 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 Особенности работы с программами-агентами

Все программы-агенты можно разделить на две основные категории: персональные и общие. Персональным агентом может пользоваться его создатель. При создании и сохранении программы-агента подпись текущего *USERID* сохраняется вместе с программой-агентом, благодаря чему и можно определить, может ли данный пользователь запустить программы-агенты. Настройка прав создания программ-агентов (*Simple Actions*, язык формул, *Lotus Script*, *Java*) осуществляется в *ACL*. Для запуска персональных программ-агентов, хранящихся на сервере, пользователь должен быть внесен в поле *Run Personal Agent* (запуск персональной программы-агента) в разделе *Agent Manager* документа *Server*. Если пользователя нет в этом списке, то он не сможет хранить персональные программы-агенты в общей БД в независимости от настроек *ACL* .

Общей программой-агентом может пользоваться любой, кто имеет к приложению право доступа *Reader*. Для создания общей программы-агента необходимо иметь к приложению право доступа *Designer*.

В документе *Server* также существуют поля *Run Restricted Lotus Script/Java Agents* (запускать ограниченные программы-агенты, написанные на языках Lotus Script/Java) и *Run Unrestricted Lotus Script/Java Agents* (запускать неограниченные программы-агенты Lotus Script/Java). Ограни-

чения на запуск программы-агента может перекладывать администратор, например, он может ограничить время работы программы-агента.

Программы-агенты можно запускать:

1) по расписанию с сервера или с клиента;

2) вручную по горячей ссылке, форме или действию представления, из меню *Action* или из списка *Agent*;

3) на основании действий, таких как создание нового документа, изменение документа, отправка в приложение и т. д.

Для создания программы-агента можно выбрать команду меню *Create→Design→Agent* и нажать на пиктограмму *Create Agent* (с лампочкой).

Если программа-агент должна запускаться пользователем вручную, то она будет ограниченна правами пользователя, указанными в *ACL*.

Первые два флажка (*Store search in search bar menu*, *Store highlights in document*) предназначены для поисковых программ-агентов (их можно хранить и использовать с помощью панели *SearchBar*). В свойствах агента в разделе *Runtime* можно выбрать способ запуска агента: по событию или по расписанию.

Для запуска по событию можно выбрать:

1. *Action menu selection* (вручную из меню *Actions*);
2. *Agent List selection* (вручную списка параметров);
3. *Before new main arrives* (до прибытия новой почты);
4. *After new mail has arrived* (после прибытия новой почты);
5. *After documents are created or modified* (если документы были созданы или изменены);
6. *When documents are pasted* (если документы были вставлены);
7. *When server starts* (когда сервер запускается).

Для запуска по расписанию нужно выбрать:

1. *More the once a day* (чаще, чем один раз в день);
2. *Daily* (ежедневно);
3. *Weekly* (каждую неделю);
4. *Monthly* (каждый месяц);
5. *Never* (никогда).

Щелкнув на кнопке *Schedule* (планирование или расписание), разработчик может настроить время и дату запуска программы-агента (в зависимости от выбранных правее настроек 1–5 в верхней части окна), сервер, на котором запускаются программа-агент (нижняя часть окна). В средней части окна *Agent Schedule* расположены три переключателя: *Start running agent on this day* (запустить агент в указанную дату), *Stop running agent on this day* (остановить агент в указанную дату), *Don't run agent in weekends* (не запускать в выходные). Если в нижней части окна *Agent Schedule* установить флажок *Choose Server when agent is enabled* (выбрать сервер, если

программа-агент активна), то сервер можно на стадии создании программы-агента не выбирать.

Опция *On Schedule Never* (5 пункт) может использоваться для программ-агентов, запускаемых другими программами-агентами.

После того как установлено, когда программы-агенты должны запускаться, необходимо определить, с какими документами они будут работать (поле *Target* в разделе *Runtime*). В поле *Target* возможны следующие значения:

1. *All documents in database*;
2. *All new & modified documents*;
3. *All unread documents in view*;
4. *All documents in view*;
5. *All selected documents*;
6. *None*;
7. *Each incoming mail documents* (все входящие документы);
8. *Newly received mail documents* (новые полученные документы);
9. *Pasted documents* (вставленные документы).

Для первых двух пунктов из *Trigger* доступны первые шесть пунктов, для третьего пункта из *Trigger* – только седьмой пункт, для четвёртого из *Trigger* – только восьмой, для предпоследнего пункта из *Trigger* – только девятый пункт, для последнего пункта – только первый, второй и шестой.

Создание программ-агентов, запускаемых на выделенном наборе документов, позволяет пользователям изменять поля в большом количестве документов.

3.3 Примеры программ-агентов на языке Lotus Script и языке формул

3.3.1 Программа-агент для отправки по почте напоминаний

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 ("Completion-
         Date")
         Set duedate = item.DateTimeValue
         If    (note.Status(0)="Open")    and    (cut-
         off.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 доку-
           мент!")
           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*.

3.3.2 Программа-агент для подсчёта количества документов-ответов

Пример. Программа-агент, подсчитывающая количество документов-ответов и записывающая эту информацию в основные документы, написанная на языке *LotusScript*.

```
Sub Initialize
    Dim session As New NotesSession
    Dim ws As New NotesUIWorkspace
    Dim db As NotesDatabase
    Dim doc,docv As NotesDocument
    Dim view As NotesView
    'Dim docv As NotesDocument
    Dim col As NotesDocumentCollection
    Dim RespCount As Integer
    Set db = session.CurrentDatabase
    Set col=db.AllDocuments 'все документы текущей БД
    Set view= db.GetView( "Films_list" )
    'получаем первый документ в представлении
    Set docv = view.GetFirstDocument
    Do While Not docv Is Nothing
        'Если документ не ответ
        If Not docv.IsResponse Then
            RespCount=0
            'получаем первый документ из коллекции
            Set doc = col.GetFirstDocument
            'цикл по всем документам коллекции
            Do While Not doc Is Nothing
               If doc.IsResponse And docv.UniversalID
               =doc.ParentDocumentUNID Then
                    RespCount=RespCount+1
               End If
               'получаем следующий док-т из коллекции
               Set doc=col.GetNextDocument(doc)
            Loop
        End If
        Call docv.ReplaceItemValue("RespCount",
        Cstr(RespCount))
        Call docv.Save(True,False)
        'получаем следующий док-т в представлении
        Set docv = view.GetNextDocument(docv)
    Loop
```

```
    views=db.Views 'все представления текущей БД
    ' обновляем все представления в текущей БД
    Forall v In views
        Call v.Refresh
    End Forall
End Sub
```

3.3.3 Поисковые программы-агенты

Пример. Программа-агент, осуществляющая поиск информации в приложении, написанная на языке *LotusScript*. Программа-агент запускается в представлении *FilmList* (приложение *filmslibrary*), находит информацию о фильме по его названию и открывает её.

```
Sub Initialize
    Dim ws As New NotesUIWorkspace
    Dim viewb As NotesView
    Dim viewf As NotesUIView
    Dim doc As NotesDocument
    retCode =Inputbox$ ("Введите название фильма
    ",,"Гладиатор",100,30 )
    If Not Isnull(retCode) Then
        'переход от клиента к серверу
        Set viewf =ws.CurrentView
        Set viewb=viewf.VIEW
        Set doc = viewb.GetDocumentByKey(retCode,
        False)
        'выбираем найденный документ в представлении
        viewf.SelectDocument(doc)
        'переходим в режим редактирования документа
        'false – режим чтения документа
        ws.EditDocument(True)
        'копируем найденный документ в папку
        doc.PutInFolder("F1")
    End If
End Sub
```

Обратите внимание, что метод `GetDocumentByKey` ищет по первому отсортированному столбцу, если второй параметр в методе равен *True*, то осуществляется поиск на точное соответствие.

Пример. Программа-агент, осуществляющая поиск информации в приложении, написанная на языке *LotusScript*. Программа-агент запускается в представлении *TestView* (приложение *filmslibrary*), пользователь

вводит номер столбца и поисковое значение, найденный документ будет открыт в режиме редактирования.

```
Sub Initialize
    Dim ws As New NotesUIWorkspace
    Dim viewb As NotesView
    Dim viewf As NotesUIView
    Dim doc As NotesDocument
    Dim i,kolnum As Integer
    Dim column As NotesViewColumn
    kolnum =Cint(Inputbox$ ("Введите номер столбца
    ",,"0",100,30 ))
    retCode =Inputbox$ ("Введите название фильма
    ",,"Гладиатор",100,30 )
    If Not Isnull(retCode) Then
        'переход от клиента к серверу
        Set viewf =ws.CurrentView
        Set viewb=viewf.VIEW
        'отменяем сортировку всех столбцов
        For i=0 To viewb.ColumnCount-1
            Set column = viewb.Columns(i)
            column.IsSorted=False
        Next i
        'Call ws.ViewRefresh
        'выбираем указанный пользователем стобец
        Set column = viewb.Columns( kolnum )
        'включаем в нём сортировку
        column.IsSorted=True
        'перекомпилируем представление
        Call ws.ViewRebuild
        Set doc = viewb.GetDocumentByKey(retCode,
        False)
        'выбираем найденный документ в представлении
        viewf.SelectDocument(doc)
        'переходим в режим редактирования документа
        'false - режим чтения документа
        ws.EditDocument(True)
        'копируем найденный документ в папку
        'doc.PutInFolder("1")
    End If
End Sub
```

Кроме метода **GetDocumentByKey** для поиска информации в представлении часто используется метод **FTSearch**.

Пример. Программа-агент, осуществляющая поиск информации в приложении, написанная на языке формул. Программа-агент запускается в представлении *TestView* (приложение *filmslibrary*) по нажатию на кнопку *Search*.

```
ch:=@DbLookup("":"NoCache";"":"";"TestView";"Адрена-
лин";2;[PartialMatch]);
t:=@Text(ch);
@Prompt([Ok];"Значение второго столбца";t)
```

Последний параметр **[PartialMatch]** означает, что осуществляется «неточный поиск», предпоследний параметр – номер столбца в представлении (начиная с 1), из которого берётся возвращаемое значение. В данном примере метод **@DbLookup** ищет в первом столбце указанного представления (*TestView*) указанное значение (Адреналин) и возвращает соответствующее значение из второго (указанного) столбца. Более подробную информацию о методе *LookUp* можно получить в [3].

Пример. Программа-агент, осуществляющая поиск информации в приложении, написанная на языке формул. Программа-агент запускается в представлении *TestView* (приложение *filmslibrary*) по нажатию на кнопку *Search&Open*. Найденный документ будет открыт в режиме просмотра (чтения).

```
:=@DbLookup("":"NoCache";"":"";"TestView";"Адрена-
лин";2;[PartialMatch]:[ReturnDocumentUniqueID]);
@Command([OpenDocument];0;ch)
```

Добавление к параметру **[PartialMatch]** параметра **[Return-DocumentUniqueID]** приводит к тому, что в переменную **ch** будет помещён UNID найденного документа, что необходимо для его открытия. В команде **([OpenDocument]** второй параметр отвечает за режим открытия документа (0 – режим чтения, 1 – режим редактирования).

3.4 Практические задания

1. Создайте программу-агента, подсчитывающую количество документов-ответов и записывающую 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 программа-агент должна обновить все представления в приложении.

2. Организуйте поиск информации в приложении, используя программы-агенты, написанные на языке формул (два варианта программы-агентов) и языке *Lotus Script*.

Алгоритм выполнения задания 1

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, см. п.п. 3.3.2).
3. Создать новое представление с общим списком (можно на основе представления, уже существующего в приложении).
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 п. 3 алгоритма, 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.

Алгоритм выполнения задания 2

1. Создать программу-агент.
2. В событии *Initialize* программы-агента записать необходимый код на языке *Lotus Script* или на языке формул (необходимо реализовать, как минимум, первый, третий и четвёртый примеры из п.п. 3.3.3).
3. Создать ***новое*** представление с общим списком.
4. Создать в представлении с общим списком кнопку-действие (для каждой программы-агента – своя кнопка).

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*.

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. В каких сферах применяются программы-агенты?
6. Перечислите основные методы класса *NotesRichTextItem*.
7. Расскажите о методе *TimeDifference* и свойстве *DateTimeValue*.
8. Опишите алгоритм программы-агента (написанной на *Lotus Script*), отправляющей по электронной почте напоминания о невыполненном задании со ссылкой на незавершенный документ.
9. Опишите принцип двойного перебора документов при поиске документов-ответов на данный основной документ.
10. Какие методы языка Lotus Script используется для поиска информации в приложении (через представление)?
11. Каковы особенности использования метода *GetDocumentByKey*?
12. Какой метод языка *Lotus Script* используется для выбора документа в представлении?
13. Как из программного кода перевести документ в режим редактирования?
14. Какой метод языка формул используется для поиска информации в приложении?
15. В чём отличие двух поисковых программ-агентов, написанных на языке формул?

Тема 4. Технология разработки web-приложений XPages

4.1 Введение в XPages

Для целей web-разработки в Domino Designer 8.5 был представлен новый элемент дизайна, называемый XPage. Он революционно меняет подход к разработке современных web-приложений в Lotus Domino.

Разработчики могут использовать все возможности HTML, XML, CSS и JavaScript для создания новых и модернизации существующих приложений.

Преимущества XPages:

1) визуальная разработка: перетаскивание элементов, быстрый доступ к свойствам и др;

2) созданы на базе *JSF* (*Java Server Faces*) – компонентная, событийно-ориентированная технология разработки web-приложений от авторов Java;

3) поддержка *AJAX* (*Asynchronous JavaScript and XML* – подход к построению интерактивных пользовательских интерфейсов web-приложений, заключающийся в «фоновом» обмене данными браузера с web-сервером): частичное обновление любого элемента, предиктивный ввод;

4) в комплект входит *Dojo* (свободная модульная библиотека JavaScript. Разработана с целью упростить ускоренную разработку, основанных на JavaScript или *AJAX* приложений и сайтов): доступ к библиотеке компонентов *Dojo* (редактор форматированного текста, селектор даты/ времени и др.);

5) возможность работы непосредственно с XML кодом;

6) любые свойства элементов могут быть вычисляемыми;

7) расширяемая архитектура: возможность использования Java кода и внешних библиотек; Custom Controls (пользовательские элементы управления);

8) расширенный доступ к данным: доступ к более чем одному NSF-файлу из XPage, отображение нескольких документов или представлений на странице.

4.2 Создание простейшего приложения в XPages

Для создания XPage в Domino Designer выберем: *XPages→New XPage*, указываем имя создаваемой страницы, нажимаем кнопку «OK». В результате в центральной части страницы отобразится редактор, а справа – набор элементов управления (если отсутствует, то выполним команду меню *Window→Show Eclipse Views→ Controls*).

Перетащим (или дважды щелкнем) *Computed Field* из вкладки *Core Controls* в центр редактора. В свойстве *Value* вычисляемого поля выберем JavaScript. Программный код можно написать в отображаемом окне, а также можно нажать кнопку *Open Script Dialog*:

```
session.getCommonUserName()
```

Глобальный объект *Session* является объектом Lotus Domino *NotesSession* для текущего пользователя.

Изменим исходный код:

```
Return "Это я –"+ session.getCommonUserName()
```

Сохраним изменения и закроем редактор. Просмотрим полученный результат: *Design→Preview in Web Browsers* и выберем браузер или воспользуемся соответствующей пиктограммой. Для предварительного просмотра Lotus Domino запускает на вашем компьютере минисервер. Для корректного запуска предварительного просмотра XPage необходимо:

1) добавить в *ACL* пользователя *Anonymous* с правом доступа не ниже, чем *author*;

2) разместить вашу базу данных (приложение) в папке с данными …/data/.

URL-адрес просматриваемой страницы имеет вид:

```
http://localhost/mybase.nsf/main.xsp
```

Если приложение расположено на сервере, то вместо *localhost* указывается имя сервера.

Чтобы приложение запускалось в Lotus Notes с XPage, необходимо выбрать *Application Properties*, выбрать вкладку *Launch*, в поле *Launch* выбрать *Open Designated XPage* (*Standard Client*), в поле XPage выбрать требуемую XPage. В Windows 7 и выше для запуска XPage в Lotus Notes следует выбрать всё выше описанное в разделе *Web browser Launch*.

Для открытия XPage из другого элемента дизайна используется формула:

```
@URLOpen("notes:///mybase.nsf/main.xsp?OpenXPage")
```

Если в нижней части редактора выбрать *Source*, то отобразится XML, определяющий вашу XPage.

Обратите внимание: если при запуске XPage в клиенте или в web-браузере возникла ошибка *Error 500*, то это, как правило, ошибка синтаксиса (JavaScript). Сообщение об ошибке можно сделать более информативным (будет показана конкретная ошибка синтаксиса, строка кода и т. д., см. п. 4.6). Ошибка *Error 404* связана с неправильными настройками XPage или web-браузера.

Разместим на XPage ещё одно вычисляемое поле, в свойстве *Value* – JavaScript, запишем следующий программный код:

```
return "Количество документов в БД: " + database.getAllDocuments().getCount().toFixed()
```

То есть получим количество документов в текущей базе данных (приложении) в виде строки (toFixed).

Пример. Получим список названий всех *nsf*-файлов в локальном каталоге Lotus Notes.

```
var list = "";
var dir = session.getDbDirectory(null);
//указывается имя сервера или null(т.е. локально)
var db = dir.getFirstDatabase(1247);
// 1247 – тип БД (nsf, nsg, nsh), 1246 – ntf и.т.д.
while (db != null) {
    if (!list.isEmpty()) list = list + ", ";
    list = list + db.getFileName();
    db = dir.getNextDatabase();
}
return list
```

Файлы с расширениями *nsg* (*Notes Storage Giant*) и *nsh* (*Notes Storage Hugen*) использовались в третьей версии Lotus Domino, когда существовало ограничение на размер файла с расширением *ntf*. Начиная с шестой версии Lotus Domino, файлы с этими расширениями не используются.

4.3 Работа с документами в XPages

4.3.1 Создание документа

Создадим 2 поля – строковое и числовое, для этого создадим таблицу (элемент *Table* в списке *Container Controls*), в первом столбце разместим 2 элемента *Label*, а во втором – 2 *EditBox* (в *Label* запишем названия полей).

Пользователь будет вводить информацию в поля. Свяжем каждое поле с глобальной переменной. Для этого в первом поле выбираем вкладку *Data*, далее – *Advanced*, в выпадающем списке выберем Scoped Variable. В поле *Parametr* выбираем *Request Scope* и указываем имя глобальной переменной, например, *fio*. Объявленная переменная доступна на странице, когда сервер обрабатывает страницу при запросе *URL*. После завершения обработки сервером *URL*-запроса, переменная становится недоступной и память очищается. Существуют другие области видимости: *Application Scope*, *Session Scope*, *View Scope*.

Аналогично поступаем и со вторым полем (*kol*), но на закладке *Data* изменим *Display type*: *String→Number*.

Ниже под полями разместим элемент управления *Button* с надписью *Create Doc*. Перейдем на вкладку *Events*, в событии *Onclick* (выбрано по

умолчанию) выберем *Script Editor* и на закладке *Server* введем код на языке JavaScript (можно использовать кнопку *Open Script Dialog*):

```
var doc = database.createDocument();
//создаем объект NotesDocument
doc.replaceItemValue("fio", requestScope.fio);
doc.replaceItemValue("kol", requestScope.kol);
//заменяем элементы (т.е. поля) значениями из EditBox
doc.save();
//очищаем scoped-переменные, чтобы старые значения не
//показывались в пользовательском интерфейсе
requestScope.subject = null;
requestScope.number = null
```

Сохраняем и просматриваем страницу, вводим значения полей и нажимаем кнопку *Create Doc*. В результате изменится количество документов в БД.

4.3.2 Просмотр документов

Для просмотра документов в XPages можно использовать элемент управления *View*, а можно – элемент управления *Data Table* и JavaScript.

Под кнопкой расположим элемент управления *Data Table*, в свойствах выбираем JavaScript и вводим сценарий, возвращающий коллекцию документа:

```
return database.getAllDocuments()
```

В *Options* в *CollectionName* указываем имя набора документов *rowdoc*. Таблица данных работает с указанным набором документов.

Щелкнем правой кнопкой внутри таблицы и выберем *Insert Column* или *Append Column*. В верхней строке из трех разместим *Label* с текстом *FIO*. В средней строке разместим элемент управления *Computed Field*, в его свойстве *Value* записываем:

```
return rowdoc.getItemValueString("fio")
```

Ввод точки после *rowdoc* не вызывает появления списка методов, так как редактор не знает ничего о типе *rowdoc*. Чтобы появлялась подсказка, запишем:

```
rowdoc:NotesDocument==null;
return rowdoc…
```

Аналогично заполняем второй и третий столбцы. Во втором столбце выведем значение поля *kol*:

```
return rowdoc.getItemValueDouble("kol")
```

В третьем столбе отобразим дату изменения документа (ZE2 – часовой пояс):

```
return rowdoc.getLastModified()
```

4.3.3 Редактирование документов

Добавим еще один столбец в таблицу данных, в который поместим кнопку *Button*. В поле *Label* кнопки записываем «*Edit*», в событии *OnClick* выбираем *Script Editor* и записываем:

```
sessionScope.docUnid = rowdoc.getUniversalID();
requestScope.fio = 'rowdoc.getItemValueString("fio");
requestScope.kol = rowdoc.getItemValueDouble("kol")
```

Для универсального идентификатора необходима переменная типа *sessionScope*, так как он используется в двух запросах. При нажатии пользователем кнопки *Edit URL*-адрес отсылается на сервер Domino (или минисервер). Сервер возвращает обратно страницу с двумя областями редактирования, заполненными значениями из соответствующей строки таблицы данных. Пользователь может изменять эти данные.

Рядом с кнопкой *Create doc* размести кнопку *Replace doc*, в обработчике события *Onclick* (в *Script Editor* выбрано *Server*) которой записываем:

```
var doc = database.getDocumentByUNID(sessionScope.
docUnid);
```

Пять остальных строчек аналогичны последним пяти строчкам кнопки *Create Doc*.

Создадим кнопку *Cancel* для очистки областей редактирования:

```
requestScope.fio = null;
requestScope.kol = null
```

В событии *Onclick* кнопки *Edit* добавим следующую строку:

```
sessionScope.editMode = true
```

В событии *Onclick* кнопок *Replace doc* и *Cancel* добавим строку:

```
sessionScope.editMode = false
```

Значение переменной *EditMode* будет использоваться для настройки видимости кнопок.

Настроим видимость трех кнопок над таблицей. В свойстве *Visible* кнопки *Create doc* нажимаем на ромбик и выбираем *Compute value*… и записываем:

```
sessionScope.editMode != true
```

Для кнопок Replace doc и Cancel:

```
sessionScope.editMode == true
```

Когда пользователь переходит к документу, видна только кнопка *Create doc*. После нажатия кнопки *Edit* кнопка *Create doc* исчезает, и появляются две другие кнопки.

4.3.4 Удаление документов

Добавим еще один столбец в таблицу данных, в его среднюю строку поместим кнопку *Remove*, в обработчике события *Onclick* (в *Script Editor* выбрано *Server*) которой запишем:

```
rowdoc.remove(true)
```

Код, размещенный на стороне сервера, выполняется после того, как браузер или клиентское приложение Lotus Notes отправит страницу на сервер. Можно также использовать JavaScript на стороне клиента. Этот код выполняется в браузере или клиентском приложении Lotus Notes до отправки страницы на сервер. Для одного и того же события можно задать как серверные (*server-side*), так и клиентские сценарии. Клиентский сценарий может предотвратить выполнение операции на стороне сервера, возвратив значение *false*.

Используем клиентский сценарий для подтверждения удаления документа. В событии *Onclick* кнопки *Remove* перейдем на вкладку *Client*, выберем *Script Editor* и введем следующий код:

```
return window.confirm("Вы действительно хотите удалить этот документ?")
```

Когда пользователь нажимает кнопку, клиентский сценарий отображает окно подтверждения. Нажатие кнопки OK возвращает значение *true*, и сценарий на стороне сервера выполняется. Нажатие кнопки *Cancel* возвращает значение *false*, и серверный сценарий не выполняется.

4.4 Использование нескольких страниц

Сделаем копию страницы *main* (например, *main2*) и будем работать с ней. Создадим новую страницу с именем *Create*. Вырежем таблицу с полями редактирования из XPage *main2* и поместим ее на XPage *Create*. Скопируем (не вырезаем!) кнопки *Create doc* и *Cancel* из *main2* и разместим из на XPage *Create*. На XPage *Create* переименуем кнопку *Create doc* в *Submit*. Для обеих кнопок изменим свойство *Visible*, чтобы они были видимы всегда.

В обработчике события *OnClick* кнопки *Submit* вместо 5 и 6 строк (очистка полей) запишем:

```
context.redirectToPage("main2")
```

Глобальный объект *context* в библиотеке *XSP* имеет тип *XSPContext*.

По нажатию на кнопку *Cancel* просто происходит возврат на главную страницу. В обработчике события *OnClick* кнопки *Cancel* можно использовать последнюю строку из обработчика события *OnClick* кнопки *Submit* или выбрать *Simple Actions* и затем *Add Action*. В появившемся окне указываем имя открываемой страницы (по умолчанию в *Action* выбрано *Open Page*) *main2*. Установим тип кнопки как *Cancel* и удалим весь код из обработчика события *OnClick*. В свойствах страницы в *Next Page* (*if update fails*) выберем значение *main2*.

В обработчике события *OnClick* кнопки *Create Doc* страницы *main2* удалим весь код. По нажатию на эту кнопку мы хотим переходить на XPage *Create*. Можно использовать *Simple Action* или следующий *JavaScript* код:

```
context.redirectToPage("create")
```

Создадим XPage страницу с именем *Replace*. Скопируем все содержимое XPage *Create* и поместим в *Replace*. Для этой страницы необходимо связать поля редактирования с переменными типа *sessionScope*, так как данные передаются по нескольким запросам (закладка *Data* свойств поля, раздел *Parametr*).

По нажатию на кнопку *Submit* мы хотим вместо создания документа получить существующий документ при помощи *UNID*, заданного на XPage *main2*. Также необходимо изменить глобальные переменные на переменные типа *sessionScope*. В результате в обработчике *OnClick* запишем:

```
var doc = database.getDocumentByUNID(sessionScope.
docUnid);
doc.replaceItemValue("fio", sessionScope.fio);
doc.replaceItemValue("kol", sessionScope.kol);
doc.save();
sessionScope.fio = null;
sessionScope.kol = null;
context.redirectToPage("main2")
```

На XPage *main2* изменим кнопку *Edit*, связав ее с переменными типа *sessionScope* для установки значений текущего элемента, и открываем XPage *Replace*:

```
sessionScope.docUnid = rowdoc.getUniversalID();
sessionScope.fio = rowdoc.getItemValueString("fio");
```

```
sessionScope.kol = rowdoc.getItemValueDouble("kol");
context.redirectToPage("replace")
```

Теперь у нас 3 страницы: *main2*, *create*, *replace*. Пользователь открывает приложение на странице *main2*. Нажатие кнопки *Create Doc* отправляет его на страницу *Create* с пустыми полями редактирования. Нажатие на *Submit* или *Cancel* возвращает пользователя на страницу *main2*. Нажатие кнопки *Edit* отправляет пользователя на страницу *Replace* с заполненными полями. Нажатие на *Submit* или *Cancel* возвращает пользователя на страницу *main2*.

4.5 XPage View Control. Поиск в БД без использования представления

XPage View Control – это основной строительный блок XPage для отображения табличных данных, который легко и быстро использовать. Строки, столбцы, заголовки и «пейджер» позволяют быстро настроить *View Control*, после того как он создан. Можно внедрить несколько представлений в одну и ту же страницу, даже из различных nsf-файлов. *View Control* автоматически показывает категорию «twisty» для категоризированных представлений.

View Control находится на вкладке *Container Controls*. После добавления его на XPage, необходимо выбрать имя представления из списка доступных (по умолчанию установлено current, т. е. текущее). В появившемся элементе управления можно настроить содержимое столбцов, их заголовки и «пейджер» (элемент навигации по приложению), стиль которого (и количество кнопок соответственно) настраивается на первой закладке свойств.

Для организации поиска на XPage без использования представления необходимо сделать следующее:

1) добавить на XPage элемент управления *EditBox*;

2) на закладке свойств *Data* в разделе *Advanced* в параметре *Use* выбрать *Scoped Variable*, в *Parametr – View Scope*, в *Variable Name* – например, **SearchValue**;

3) добавить на XPage элемент управления *Link Control*, в его свойстве *Label* укажем, например, «Перейти»;

4) используя свойство *URL* или событие *OnClick* в *Link Control*, можно вычислить следующую открываемую XPage (её имя вводим в *EditBox*) или *URL*-адрес. Для этого используем `viewScope.SearchValue`. Вычислим имя (следующей) открываемой XPage, в событии *OnClick* выберем *Add Action*, в открывшемся окне в *Category* выбираем *Basic*, в *Action* –

Open Page. Свойство *Name of page to open* сделаем вычисляемым (правый клик по ромбику). В переключателе *Condition* выберем *Compute Dynamically* и ниже запишем:

```
viewScope.SearchValue+'.xsp'
```

Обратите внимание, что язык JavaScript чувствителен к регистру, в *EditBox* вводим имя XPage без расширения!

Теперь после ввода в *EditBox* имени XPage и нажатия на ссылку «Перейти» будет открыта указанная в *EditBox* XPage.

4.6 Настройка свойств проверки данных для элемента управления. Настройка свойств стилей для XPages

Для задания свойств проверки (валидации) данных необходимо:

1) выбрать в редакторе XPages элемент управления, для которого необходимо установить проверку данных, в его свойствах перейти на вкладку *Validation*;

2) установить флажок *Required Field*, если для данного элемента управления требуется проверка данных (это значение также можно вычислить на основе некоторого условия);

3) в поле *Required Field Error Message* можно написать сообщение об ошибке, которое появится, если пользователь не ввёл необходимые данные в поле (например, «Введите ФИО!»);

4) в полях *Minimum* и *Maximum* можно ввести минимальное и максимальное числовое значение, которое пользователь может ввести в это поле. Эти параметры применяются к следующим типам отображения: *Number*, *String*, *Date/Time*;

5) в поле *Validation Error Message* можно записать сообщение об ошибке, которое выводится, если ввод данных не соответствует параметрам, указанным выше (свойство является вычисляемым), например, «Не попали в заданный интервал!»

Настройка вычисляемых свойств для элемента управления

Вычисляемые свойства элемента управления являются свойствами, которые рассчитываются и отображаются, когда пользователь обращается к XPage (во время её работы), а не в момент запуска XPage. Вычисляемые свойства рассчитываются следующим образом (переключатель *Condition*):

1) если выбрать *Compute Dynamically*, то формулы будут пересчитаны при обновлении страницы;

2) если выбрать *Compute on Page Load*, то формулы вычисляются один раз при загрузке страницы.

Отображение ошибок выполнения

Если что-то пошло не так во время просмотра XPage в web-браузере, то по умолчанию ошибки отображаются очень «размыто» и их почти невозможно отследить.

В XPage-приложениях можно настроить способ вывода ошибки: в навигаторе приложений переходим к *Application Configuration*, выбираем *XSP Properties*, на первой вкладке General установим флажок *Display XPage runtime error page*. Сохраняем и закрываем страницу свойств приложения. Теперь при возникновении ошибки появится более подробная информация о ней.

Для настройки свойств стилей для XPages необходимо сделать следующее:

1) выбрать вкладку *Style* в свойствах XPage. Любые стили для шрифтов, полей и фона (определённые в XPages) будут иметь приоритет перед любыми стилями из CSS;

2) чтобы применить стиль из конкретной CSS (который не входит в список недавно использованных стилей), нужно выполнить такие действия:

а) нажать *Add style sheet to page…*

б) для создания новой CSS необходимо выбрать *Use style sheet from this application* и нажать New. Далее указываем имя новой CSS, нажимаем на кнопке «ОК», выбираем новую таблицу стилей из списка, снова нажимаем «ОК». Затем необходимо написать код CSS в редакторе. Сохраняем полученный файл;

в) для того чтобы импортировать CSS, необходимо выбрать *Use style sheet from this application* и нажать *Import*;

г) для использования внешних CSS необходимо выбрать *Link to a Style sheet from a URL* и указать путь к CSS;

е) после того как новая CSS появится в категории *Style sheets on the page*, необходимо выбрать класс из CSS. После выбора класса он появится в поле *Class*;

3) можно также выбрать ранее использованные классы из CSS (категория *Recently used styles*);

4) можно включить или выключить опцию *Disable runtime-applied default styles* для активации или деактивации runtime-исполняемых стилей, заданных по умолчанию;

5) можно использовать вкладки Style вкладок Font, Background, Margins;

6) в обязательном порядке сохраним сделанные изменения.

4.7 Практические задания

1. Создать в учебной базе данных Xpage страницу. В верхней части страницы располагаются вычисляемые поля для отображения имени текущего пользователя и количества документов в БД. XPage должна создавать документы в БД (но ***поля берутся не из примера***, а как минимум, три основных поля из вашей БД), иметь возможность редактирования и удаления документов из БД. В *Data Table Control* кроме трёх основных полей документа (были помечены звёздочкой, которая видима только в режиме редактирования), должна отображаться дата создания документа, и присутствовать кнопки для редактирования и удаления документов. Нужно соответствующим образом настроить видимость всех кнопок, организовать проверку заполнения поля данными, как минимум, в двух полях.

2. Организовать такую же работу с документами через XPage, как описано выше, но с использованием нескольких страниц. Опять же ***работаем со своими полями***, а не с полями из рассмотренного выше примера (см. п. 4.4).

3. Создать простейшую XPage страницу для работы с представлениями, используя *View Control*. Организовать поиск на XPage без использования представлений. В элемент управления *EditBox* вводим имя XPage, которая будет открыта через *Link Control*.

Алгоритм выполнения задания 1

1. Создаём XPage страницу, например, с именем main (см. п. 4.2). На ней размещаем вычисляемые поля, кнопки, настраиваем их свойства (см. п.п. 4.3.1).

2. На XPage помещаем элемент управления *Data Table,* настраиваем его свойства, размещаем в нём необходимые элементы управления (кнопки, *EditBox*), настраиваем их свойства (см. п.п. 4.3.2). Редактирование и удаление документов осуществляется согласно п.п. 4.3.3 и 4.3.4.

3. Видимость кнопок настраивается в свойстве *Visible*, предварительно необходимо сделать его вычисляемым (см. п.п. 4.3.3).

4. Выбираем поле, обязательное для заполнения и организуем проверку заполнения поля данными (вкладка Validation свойств элемента управления, см. п. 4.6).

Алгоритм выполнения задания 2

1. Создаём копию XPage страницы *main*, назовём её, например, main2. Порядок работы со страницей *main2* рассмотрен в п. 4.4.

2. Создаём еще две XPage страницы: *Create* – для создания новых документов и *Replace* – для редактирования существующих документов. По-

рядок размещения кнопок и других элементов дизайна на новых страницах приведён в п. 4.4.

3. Изменим программный код кнопок *Cancel*, *Submit* (см. п. 4.4).

4. Запускаем XPage *main2*. По Нажатию кнопки *Create Doc* на страницу *Create* с пустыми полями редактирования. Нажатие на *Submit* или *Cancel* возвращает пользователя на страницу *main2*. Нажатие кнопки *Edit* отправляет пользователя на страницу *Replace* с заполненными полями. Нажатие на *Submit* или *Cancel* возвращает пользователя на страницу *main2*.

Алгоритм выполнения задания 3

1. Создаём новую XPage. С вкладки *Container Controls* разместим на XPage элемент управления *View*. Выбираем требуемое представление, настраиваем строки, столбцы и др. (см. п. 4.5).

2. Для организации поиска на XPage без использования представления создадим новую XPage. С вкладки *Core Controls* разместим на XPage элемент управления *Link*. Далее следуем алгоритму, приведённому в п. 4.5.

Вопросы для самоконтроля

1. В какой версии Domino Designer впервые появились XPages?
2. Перечислите основные преимущества XPages.
3. Что такое AJAX?
4. Что такое Dojo?
5. Что означает расширяемая архитектура XPages?
6. Порядок создания простейшего приложения в XPages.
7. Что необходимо сделать для корректного запуска предварительного просмотра XPage?
8. Какая формула используется для открытия XPage из другого элемента дизайна?
9. Чем ошибка Error 500 отличается от Error 404?
10. Каков порядок создания документа в XPage?
11. Как осуществить просмотр документов в XPage?
12. Как происходит редактирование документов в XPage?
13. Как правильно удалить документы в XPage (серверные и клиентские сценарии)?
14. Как необходимо изменить работу приложения при использовании трёх XPage страниц вместо одной?
15. Для чего используется XPage *View Control*?
16. Какие преимущества имеются у XPage *View Control*?

17. Какой элемент управления используется для организации поиска в приложении без использования представления?

18. Чувствителен ли язык программирования JavaScript к регистру?

19. Как правильно настроить свойства проверки данных для элемента управления?

20. Как правильно настроить вычисляемые свойства для элемента управления?

21. Как сделать так, чтобы отображение ошибок выполнения стало более информативным?

22. Порядок настройки свойств стилей для XPages.

Литература

1 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.

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

3 Кузьменков, Д.С. Lotus Domino/Notes: практическое руководство: 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.

4 Кузьменков, Д. С. Lotus Domino/Notes: практическое руководство: 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.

5 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.

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

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

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

9 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.

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

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

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

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

14 Donnelly, M. Mastering XPages / M. Donnelly, M. Wallace, T. McGuckin. – IBM Corp., 2014. – 750 p.

15 XPages без сложностей – URL [Электронный ресурс] : http://www.ibm.com/developer-works/ru/library/ls-domdes-xpages/index.html. – Загл. с экрана.

Производственно-практическое издание

Кузьменков Дмитрий Сергеевич,
Кузьменкова Екатерина Юрьевна

КОМПЬЮТЕРНЫЕ СИСТЕМЫ УПРАВЛЕНИЯ ДОКУМЕНТООБОРОТОМ И WEB-ТЕХНОЛОГИИ

Практическое руководство

Редактор *В. И. Шкредова*
Корректор *В. В. Калугина*

Подписано в печать 01.06.2017. Формат 60×84 1/16.
Бумага офсетная. Ризография. Усл. печ. л. 2,8.
Уч.-изд. л. 3,1. Тираж 25. Заказ 521.

Издатель и полиграфическое исполнение:
учреждение образования
«Гомельский государственный университет
имени Франциска Скорины».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 3/1452 от 17.04.2017.
Специальное разрешение (лицензия) № 02330 / 450 от 18.12.2013.
Ул. Советская, 104, 246019, Гомель.