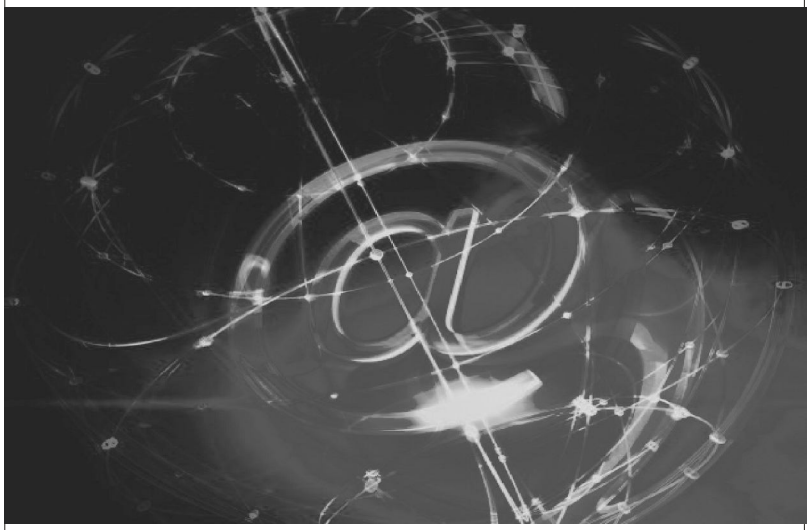
InterTrust

Е.В.Поляков

Язык @-формул в Lotus/Domino R 6
Справочник разработчика



тел./факс: (095) 956-7928
http://www.intertrust.ru
E-mail: intertrust@inttrust.ru

IBSN 5-7419-0075-5

УДК 004.45
ББК 22.18
П 16

Поляков Е.В.
«Язык @-формул в Lotus/Domino R 6. Справочник разработчика»

Настоящий справочник посвящен языку программирования @-формул, используемому при создании приложений под управлением Lotus Domino/Notes версии R 6 - программного продукта компании IBM Company.

В книге детально рассматривается синтаксис языка @-формул, его лексические элементы, последовательность выполнения формул и механизмы обработки ошибок. Наиболее часто встречающимся в разработках @-функциям и @-командам с примерами их использования и областью определения посвящены две самые обширные главы настоящего издания. В разделе приложений приводится полный алфавитный список @-функций и @-команд, контекст их применения, а так же имеющихся ограничения их использования на различных платформах.

Изложение материала сопровождается обширным набором иллюстраций и примеров, наглядно демонстрирующих варианты применения конкретных @-функций и @-команд.

Справочник предназначен разработчикам приложений для Lotus Notes, уже знакомым с основным средством разработки Domino Designer. Книга содержит исчерпывающие сведения о языке @-формул Lotus/Domino версии R 6, и может быть полезна как начинающим разработчикам, так и специалистам со стажем.

ООО «Светотон»
109341, г. Москва, ул. Верхние Поля, д.18
E-mail: svton@from.ru

Подписано в печать 29.11.2006 г. Формат 60x90/16
Печать офсетная. Бумага офсетная №1.
Усл. печ. л. 24,25 Тираж 500 экз. Заказ

Отпечатано в ФГУП «Производственно-издательский комбинат ВИНИТИ», 140010, г. Люберцы Московской обл., Октябрьский пр-т, 403

Предисловие автора

Книга, которую Вы держите в руках, посвящена вопросам программирования на языке @-формул. Данный язык широко используется при создании приложений под управлением Lotus Domino/Notes версии R 6 - программного продукта компании IBM Company.

Книга состоит 3 частей и двух сегментов приложений. В первой части (главы 1- 5) рассматривается синтаксис языка @-формул, его лексические элементы, последовательность выполнения формул и механизмы обработки ошибок. Во второй части (глава 6) приведены наиболее употребляемые в разработках @-функции с примерами их использования. Третья часть содержит часто используемые @-команды, так же с наглядными примерами. Сегмент приложений содержит полные алфавитные списки @-функций, @-команд, контекст их применения, а так же имеющихся ограничения их использования на различных платформах.

Примеры, используемые в настоящей книге, тестировались на версии Lotus/Domino R 6.0.1 под управлением Windows-2000 (в некоторых случаях использовался Linux Red Hat 8). Тестировании Web-приложений выполнялось в браузере MS Internet Explorer 5.0.

Автор отдает себе отчет в том, что книга не лишена недостатков, и имеет к себе претензии по стилю изложения и оформления материала. Ряд ее разделов хотелось бы улучшить, дополнить и снабдить более интересными примерами. Однако в условиях необычайно быстрого развития продукта лучше скорее дать в руки читателей информацию в том виде, как она есть сейчас, чем отложить выпуск книги еще на один-два месяца.

Автор будут искренне рад Вашим замечаниям и предложениям по содержанию книги и любой информации технического характера по затронутым в книге вопросам. Направляйте их по e-mail: EPolyakov@inttrust.ru или «оставляйте» на WWW-сервере нашей компании http://www.intertrust.ru.

Автор выражают искреннюю благодарность и признательность всем, кто оказывал прямую или косвенную помощь в работе над книгой. Особые благодарности ведущим сотрудникам компании InterTrust Co.: Ионцеву Н. Н., Панову В. А., Ардабьеву И. Н., Иванову Д. Ю., Кадникову А. О. – без помощи которых, книга выглядела бы другой. Специальные благодарности: Бреусу И. Б., Поляковой Г. Д., Полякову В. Г. и Перлич Н. А. за существенную помощь, оказанную при написании настоящей книги, а также сыну Роману и дочери Лане за терпение и сочувствие во время работы над книгой.

С уважением, Е. Поляков

1 Введение

Формулы или **@-формулы** состоят из одного или более предложений - операторов (statements), которые обычно выполняются по порядку (более подробно это будет рассмотрено в пункте "4.4"). Префикс @- происходит от частого использования в таких формулах вызовов @-функций. В приложениях Domino формулы широко используются в следующих целях:

- отбор документов при селективных репликациях;
- отбор документов для включения в вид;
- вычисление значений для показа в документах, видах и папках;
- определение видимости абзаца формы, абзаца в RichText-поле документа, объекта в области размещения, акции в форме или виде;
- вычисление, преобразование и проверка значений полей документов;
- добавление и удаление полей в документах;
- определение формы, используемой для открытия документа в виде;
- определение вычисляемых субформ в форме;
- выполнение последовательности действий при запуске агента, клике по кнопке панели инструментов, акции в виде, активного участка в навигаторе, акции, кнопки или активного участка в форме.

В Notes версий 3.x @-формулы были практически единственным средством для вычислений и автоматизации обработки информации. В 4-й версии появились, по крайней мере, еще два способа: один более простой, но ограниченный - простые действия (simple actions), другой более сложный, но богатый по своим возможностям - объектно-ориентированный язык программирования LotusScript. Пятая версия Domino в дополнение к языку LotusScript дала разработчикам возможности использования объектно-ориентированного языка Java, который в последнее время стал фактически стандартом для создания Internet-приложений.

В R 6 язык @-формул значительно обновился и стал более похож на процедурный язык (появились операторы цикла, разрешено повторное присваивание значений временным переменным, операторы присваивания допускают вложенность, и т. д.). Кроме этого появились новые элементы дизайна, поддерживающие язык @-формул (например, формула поля Input Enabled [разрешить/запретить ввод в поле]).

Обратите внимание, что новые или модифицированные в версии R 6 @-функции и @-команды помечены значком перед заголовками.

Java и LotusScript обеспечивают множество операций, невозможных для @-формул. Например, модификацию списка управления доступом к базе (ACL), но часто использование @-формул бывает проще и эффективнее, а во многих местах @-формулы остались единственным вариантом. Например, в кнопках панели инструментов, при селективных репликациях, для определения видимости объектов, в колонках, критериях отбора и формулах формы видов, в формулах полей.

2 Синтаксис формул

Формула состоит из одного или более операторов (statements), разделяемых символом точка с запятой («**;**»). Синтаксис формул должен подчиняться следующим правилам:

- формула должна начинаться с ключевого слова, вызова @-функции, переменной или константы;
- формула должна заканчиваться переменной, константой или @-функцией;
- аргументы @-функций отделяются друг от друга с помощью символа «**;**»;
- список аргументов функций заключается в скобки. Если аргументы не требуются, скобки можно опустить;
- допускается размещать произвольное количество пробелов и переводов строки (в том числе ни одного) между операторами формул и выполнения операций;
- за ключевым словом должен следовать хотя бы один пробел;
- регистр (прописные или строчные буквы) не имеет значения, кроме как внутри текстовых констант. Ключевые слова автоматически преобразуются в верхний регистр при сохранении формулы.

3 Лексические элементы формул

Каждый оператор формулы может состоять из:

- переменных;
- констант;
- операторов выполнения операций (operators), не путать с операторами формул (statements);
- встроенных @-функций;
- ключевых слов.

3.1 Переменные

Переменные бывают двух типов: поля и временные переменные (также называемые временными полями).

3.1.1 Поля

Поля - наиболее часто используемые переменные в формулах. Формула имеет доступ к полям обрабатываемого документа. Для ссылки на значение поля нужно вставить в формулу название поля. При этом важно не забывать о типе данных поля. Большинству функций требуются определенные типы аргументов, а операторам - определенные типы операндов. Для определения типа и имени поля можно обратиться к дизайну конкретной БД.

При использовании полей в качестве переменных в формулах, наиболее часто встречаются ошибки для полей типа RichText и полей с множественными значениями (multi-values field).

RichText-поля, как правило, не могут использоваться как переменные. Тем не менее, в формуле агента можно присвоить значение такому полю с помощью ключевого слова FIELD, а функция @DbLookup может получить содержимое RichText-поля из другого документа.

Функцию @IsAvailable(FieldName) можно использовать для установления факта наличия в документе поля (в том числе RichText) с таким именем.

Для multi-values-полей следует помнить, что только достаточно ограниченное число @-функций и операторов работают со списками. Для определения является ли значение поля списком можно воспользоваться функцией @Elements(FieldName). Если возвращаемое значение больше единицы, это говорит о том, что значение поля действительно список.

Формулы могут модифицировать значение полей в документе только в том случае, если эти действия не противоречат списку управления доступом к текущей БД.

Имена полей в формулах могут использоваться различными путями:

- для определения значений по умолчанию. При создании нового документа, если в форме установлено "наследование" полей, возможны ссылки на поля в документе, который был открыт или выделен в момент создания. Эта возможность широко используется в Domino при разработке форм для документов-ответов, при создании новых документов "по образцу";
- в выражениях для получения новых значений с помощью операторов. Например, формула в колонке вида: FirstName + " " + LastName, где FirstName и LastName - поля в документах, содержащие фамилию и имя;
- для задания аргументов функций. Здесь возможны два принципиально различных варианта:
  - название поля используется для ссылки на значение, содержащееся в поле, например, @Text(BirthDate);
  - название поля используется как текстовая строка, например формула возвращает содержимое поля **Status** в документе, найденном по ключу "Virus Check":

```
@DbLookup( "" : "NoCache"; "RESEARCH" : "SMITH\\PROJECTS.NSF";
          "In Progress"; "Virus Check"; "Status" )
```

- для задания значения поля:

```
FIELD FieldName := Value;
(например, FIELD Status := "На контроле";)
```

или

```
@SetField( "FieldName"; Value );
(например, @SetField( "Status"; "На контроле" );
```

Оператор с ключевым словом FIELD присваивает значение полю, если оно уже существует в документе, в противном случае создается новое поле с таким именем. Его тип данных определяется типом присваиваемого значения. Таким же образом можно и удалить поле из документа:

```
FIELD FieldName := @DeleteField;
```

Поля, значения которых не определены, имеют значение пустой строки (""). Это правило справедливо для полей любых типов, включая и нетекстовые поля. В связи с этим возникает достаточно много ошибок несовместимости типов данных. Для решения этой проблемы рекомендуется для нетекстовых полей определять формулу значения поля по умолчанию.

3.1.2 Предопределенные поля

В документах Domino могут присутствовать поля, содержащие определенную служебную информацию. Например, $UpdatedBy, $Revisions, $Ref, $Conflict, Form и т.д. Не все из этих полей (и не во всех контекстах) могут быть использованы в качестве переменных @-формул (для каждого конкретного случая необходимо обращаться к документации). Так, например, в акции формы формула @Prompt([Ok]; "First Editor"; @Subset($UpdatedBy; 1)) действительно выдаст окно с именем первого автора документа, однако формула @SetField("$UpdatedBy"; "Ivan I Ivanov") не изменит значения поля $UpdatedBy. Следует заметить, что сразу после выполнения данной формулы, если посмотреть значение поля $UpdatedBy (например, через окно свойств), то его значение действительно будет "Ivan I Ivanov", но по выходу из режима редактирования (даже с сохранением) Lotus сам восстановит значение указанного поля.

Наиболее часто из предопределенных полей в @-формулах используется поле Form. Данное поле содержит имя формы, по которой текущий документ должен отображаться/модифицироваться по умолчанию. Поле Form может использоваться в @-формулах так же как и любое другое не предопределенное поле.

3.1.3 Временные переменные

Временные переменные доступны только в пределах формулы, в которой они определены (от определения до конца формулы) и в течение одного выполнения этой формулы. Создание и инициализация временной переменной аналогично инициализации поля, но без ключевого слова FIELD:

```
VariableName := Value;
```

Тип данных временной переменной соответствует типу значения в правой части оператора присваивания. Этот тип может быть Text, Time, Number или Boolean. Последний - логический - возвращается некоторыми @-функциями

и может принимать значения True (Истина - эквивалентно числу 1) или False (Ложь - эквивалентно числу 0).

В R 6 снято ограничении на то, что в пределах одной формулы переменная с одним и тем же именем не должна более одного раза встречаться в левой части оператора присваивания. Для изменения значение временной переменной в процессе выполнения формулы можно использовать как оператор присваивания, так и функцию @Set:

```
VariableName := 1;

. . .

VariableName := 2;
```

так и :

```
VariableName := 1;

. . .

@Set( VariableName; 2 );
```

3.2 Константы

В формулах можно использовать константы трех типов: текстовые, числовые и типа дата-время. Список значений также может использоваться в качестве константы.

Текстовые константы в формулах заключаются в двойные кавычки: **"Text constant"**. В R 6 текстовые константы можно заключать в фигурные скобки. **{Text constant}**.

Чтобы многократно повторить один и тот же символ используйте функцию @Repeat.

Обратный слеш «\» внутри текстовой константы служит Esc-символом. Таким образом, если текст в константе, заключенной в двойные кавычки, должен содержать символы «"» или «\» перед ними нужно вставлять добавочный символ «\». В некоторых случаях после сохранения формулы с фигурными скобками, выступающими в качестве ограничителей текстовой константы, последние преобразуются к «\"».

Числовые константы могут состоять из символов цифр и специальных символов, неразделенных пробелами. При этом должны соблюдаться следующие правила:

- первый символ «+» или «-» определяет знак числа, соответственно положительное значение, или отрицательное;
- целые числа состоят только из символов цифр, неразделенных пробелами;

- нецелые числа могут быть представлены в обычной и математической нотации. Для обычной нотации десятичная точка может находиться перед, после или внутри последовательности символов цифр, неразделенных пробелами. Для математической нотации – число может содержать суффикс из символа «E» и следующих за им чисел, определяющих показатель степени со знаком или без.

  Например, 123, -123, 1.23, -0.123, 1.23E-12.

Дата-время (Time-date) константы должны быть заключены в квадратные скобки. Например, **[5:30]**, **[30.3.93]** или **[30.3.93 5:30]**. Формат представления даты и времени (порядок следования, разделители и т.д.) должен соответствовать стандартам, установленным в операционной системе. При использование сокращенного формата даты для года, например, **[dd.mm.yy]**, если **yy** < 50 - считается, что это 20**yy** год, в противном случае 19**yy** год. Используйте полный формат даты для года, если это необходимо.

Результат вычитания двух значений типа дата-время есть целое число, представляющее разницу во времени между этими значениями в секундах. Например, формула:

```
@Prompt( [OK]; "Проба"; @Text( [30.3.93 5:30]- [30.3.93 5:29] ));
```

выдаст окно со значением 60.

3.3 Операторы выполнения операций

В таблице, приведенной ниже, перечислены все допустимые в @-формулах операторы в порядке убывания их приоритета. Горизонтальные линии отделяют группы операторов с разным приоритетом. Если в одном выражении встречаются операторы одного приоритета, они выполняются слева направо. Изменить порядок выполнения можно с помощью круглых скобок.

Оператор	Название, описание	Приоритет	Если операнды - списки
:=	Присваивание. В R 6 допустимы вложенные операторы присваивания	-	
[]	Выделение элемента списка. Новый оператор в R 6	1	
:	Объединение списков	2	
-, +	Изменение знака	3	
*	Умножение	4	Попарно
**	Умножение		Все со всеми
/	Деление		Попарно
*/	Деление		Все со всеми
+	Сложение, конкатенация	5	Попарно
*+	Сложение, конкатенация		Все со всеми
-	Вычитание		Попарно
*-	Вычитание		Все со всеми
=	Равно	6	Попарно
*=	Равно		Все со всеми
<>, !=, =!, ><	Не равно		Попарно
*<>	Не равно		Все со всеми
<	Меньше		Попарно
*<	Меньше		Все со всеми
>	Больше		Попарно
*>	Больше		Все со всеми
<=	Меньше или равно		Попарно
*<=	Меньше или равно		Все со всеми
>=	Больше или равно		Попарно
*>=	Больше или равно		Все со всеми
!	Логическое ОТРИЦАНИЕ	7	
&	Логическое И		
|	Логическое ИЛИ		

В R 6 стали допустимы вложенные операторы присваивания. Например,

```
Categories := @UpperCase((Country := "Египет") + "\\" + Town := "Хургада");
@Prompt([Ok]; "Categories"; Categories); - выдаст ЕГИПЕТ\ХУРГАДА
@Prompt([Ok]; "Country"; Country); - выдаст Египет
@Prompt([Ok]; "Town"; Town); - выдаст Хургада
```

Если бы в первом операторе формулы не взять в скобки (Country := "Египет"), то значение переменной Country было бы Египет\Хургада. Более наглядно последовательность выполнения вложенных операторов видна из следующего примера:

```
A := 2;

B := 3;

C := 4;

D := 5;

E := @Text( A * B := C + D );

@Prompt([Ok]; "A"; @Text(A)); - выдаст 2

@Prompt([Ok]; "B"; @Text(B)); - выдаст 9, т.к. 4+5 = 9

@Prompt([Ok]; "C"; @Text(C)); - выдаст 4

@Prompt([Ok]; "D"; @Text(D)); - выдаст 5

@Prompt([Ok]; "E"; E); - выдаст 18, т.к. 2*9 = 18
```

В левой части от оператора присваивания как префикс перед переменной типа поля (но не временной переменной) могут стоять ключевые слова DEFAULT, ENVIRONMENT, или FIELD. Для вложенных операторов присваивания допустимо только ключевое слово FIELD.

```
FIELD Categories := @UpperCase(FIELD Country := "Египет" + "\\" + "Хургада");
```

Оператор двоеточие «**:**» используется для объединения двух значений одного типа данных в список. Каждый из операндов сам может быть списком. Результат содержит все элементы первого операнда, затем все элементы второго. При многократном использовании в одном выражении можно объединить в список несколько значений:

```
"Moscow" : "London" : "New York" : "Tokyo"
```

В R 6 появился новый оператор квадратные скобки «**[]**», который позволяет выделить N-й элемент списка по его индексу:

```
my_list := "Moscow" : "London" : "New York" : "Tokyo";

@Prompt([Ok]; "Информация"; my_list[2]) – выдаст London
```

В качестве индекса может выступать константа, переменная или выражение, имеющие числовое значение (десятичные числа округляются до ближайшего целого). Первый элемент списка имеет индекс – 1. При выходе значения индекса за границы слева и справа, возвращается сообщение об ошибке «Array index out of bounds» («Индекс массива выходит за границы»).

Оператор выделения элемента списка может применяться для любых типов данных, которые могут принимать значение списков (текст, числа, дата/время), даже если значение скалярно (т.е. не список). Для типов данных, которые не могут принимать значение списков (Rich text поля), допустимо использование только индекса 1. Возвращаемое значение в этом случае равно текущему уже сохраненному значению Rich text поля без изменений.

```
Rem "поле my_date имеет тип Data/Time содержит значение 26.04.1964, и не допускает принятие множественных значений типа список «Allow multiple values»";

@Prompt([Ok]; "Информация"; @Text(my_date[1])); - выдаст 12.04.1964
```

Оператор выделения элемента списка не может быть использован в левой части оператора присваивания. Таким образом нельзя изменить значение списка присваиванием нового значения оператору выделения элемента списка (т.е. VariableName[1] := "New Value" – приведет к состоянию ошибки). В правой части оператора присваивания оператор выделения элемента списка допустим.

```
my_list := "Moscow" : "London" : "New York" : "Tokyo";

my_list := my_list[3] : "Volgograd" : my_list[1];

@Prompt([Ok]; "Информация"; @Implode(my_list; "-")) – выдаст New York-Volgograd-Moscow
```

В документации сказано, что оператор выделения элемента списка, который следует за оператором объединения списков, должен быть заключен в круглые скобки. Но все работает и без этого, и не совсем понятно зачем это нужно, ведь оператор выделения элемента списка имеет более высокий приоритет, чем оператор объединения списков.

Операции (и операторы) со **списками** возможны двух типов:

Попарно (Pair-wise) - первый с первым, второй со вторым и т.д. Если один из списков короче, вместо недостающих элементов используется последний.

Все со всеми (Permuted) - каждый с каждым - перебираются все комбинации в следующем порядке: первый элемент первого списка со всеми элементами второго, затем второй элемент первого списка со всеми второго и т.д. Обозначение: перед обычным знаком операции вставляется символ «*».

При операциях сравнения списков в обоих случаях для получения положительного результата всей операции достаточно положительного результата в одной паре.

Пример. Сравнение двух списков на равенство и неравенство может привести к одинаковому результату!

```
"A":"B"  = "A":"C"  - Истина, т.к. "A" = "A"

"A":"B" != "A":"C"  - тоже Истина, т.к. "B" != "C"
```

3.4 Порядок выполнения вычислений

В выражении все входящие в него значения должны быть одного типа. Порядок вычислений соответствует общепринятому в математике. Сначала вычисляются выражения в скобках (при наличии вложенных скобок в первую очередь выполняются вычисления в самых внутренних скобках), затем выполняются операции с высшим приоритетом. При равенстве приоритетов вычисления выполняются слева направо.

Пример. В результате вычисления переменной **a** присвоится значение 20.

```
a := (12 + (24-3)*2)/3 + 6/3;
```

Необходимо помнить, что оператор объединения списка имеет один из наиболее высоких приоритететов, поэтому элементы списка, которые являются выражениями, необходимо брать в скобки.

Пример. a := 1:2:3:4 + 1:2:-3:4; В результате вычисления переменной **a** присвоится значение 2:4:0:0. Дело в том, что элементы 3 и 4 сначала были объединены в список, а затем к этому списку применяется операция изменения знака. Соответственно для выражения 1:2:3:4 + 1:2:(-3):4 значение будет 2:4:0:8.

3.5 Ключевые слова

DEFAULT *имя_поля* := *значение*

Область применения: без ограничений.

На время выполнения формулы присваивает полю с именем ***имя_поля*** соответствующее ***значение*** по умолчанию. Данная конструкция устанавливает, что если в документе не задано поле с именем ***имя_поля***, то до конца формулы или до присвоения этому полю другого значения будет считаться, что оно содержит соответствующее ***значение***. Если же поле с именем ***имя_поля*** в документе присутствует, то конструкция не оказывает никакого воздействия.

ENVIRONMENT *переменная_окружения* := *"значение"*

Область применения: нельзя использовать в формулах отбора, колонок, видимости объекта и навигатора. Ограниченно применима в формулах всплывающих окон. В Web-приложениях выдает пользовательские Web-переменные окружения.

Присваивает ***значение*** текстовой строки ***переменной_окружения*** (среды). Данные переменные сохраняются либо в файле NOTES.INI (для операционных систем Windows, OS/2 и UNIX), или в Notes Preferences файле (для Macintosh). Действие ключевого слова аналогично выполнению встроенной @-функции @SetEnvironment.

Для возврата значений таких переменных используется функция @Environment:

```
FIELD FieldName := @Environment( "EnvVariable" )
```

В Web-приложениях, используя предопределенные имена полей, с помощью функций работы с переменными окружения можно получить пользовательские Web-переменные окружения. В данном случае используется механизм CGI (Common Gateway Interface).

FIELD *имя_поля* := *значение*

Область применения: нельзя использовать в формулах отбора, колонок, видимости объекта, заголовка окна, формы и навигатора.

Присваивает ***значение*** полю с именем ***имя_поля***. Данное поле сохраняется в документе (в отличие от временных переменных). Может быть использовано как для создания новых, так и для изменения существующих полей. При создании новых полей разработчик должен сам следить за тем, чтобы ***имя_поля*** было уникально в пределах текущего документа.

В R 6 язык снято ограничение, запрещающее использование ключевого слова **FIELD** в качестве параметров других функций (например, @Do или @If). Соответственно допускается и вложенное использование **FIELD**. Для предыдущих версий (включая R5) это ограничение действительно.

Пример. В R 6 полю a1 и временной переменной с присвоится значение “123”, а полю a2 - “12345”.

```
Field a2 := @Do( c:= Field a1 := "123"; c + "45");
```

В документации на данное ключевое слово приведено два замечания. Первое - рекомендует внутри других функций использовать вместо ключевого слова FIELD функцию @SetField. Во втором говорится, что в определенных случаях формула должна возвращать некое значения (например, формула

кнопки). Поэтому для предупреждения возникновения ошибки типа "No Main or Selection expression in formula" (нет главного выражения или оператора отбора в формуле) рекомендуется добавить после оператора с ключевым словом FIELD новый оператор возвращающий значение (например, "" или Select @All).

Строгое следование данным замечаниям действительно на 100% избавит от описываемых ошибок. Однако с другой стороны, в R 6 снято ограничение, запрещающее использование ключевого слова **FIELD** в качестве параметров других функций, и кроме этого ключевое слово **FIELD** теперь возвращает значение. Помимо этого при сохранении формул, требующих возвращаемого значения (например, формула кнопки), Domino обычно добавляет в конец формулы функцию @True. Данное правило выполняется не всегда, например, если определить обработчик события onLoad формы на языке @-формул в виде «FIELD a2 := "45";», то в результате как раз и получим сообщение об ошибке "No Main or Selection expression in formula".

REM ["*комментарий*"]
REM [{ *комментарий* }]

Область применения: без ограничений.

Позволяет включить в формулу одну или несколько строк ***комментариев***. В R 6 комментарии стало можно заключать в фигурные скобки. Это достаточно удобно при отладке @-формул, когда надо закомментировать какой-то фрагмент текста, а внутри него уже встречаются двойные кавычки. Символ «\» (обратный слеш) используется внутри ***комментария*** как Esc-символ, аналогично текстовым константам.

```
REM "Это строка \"комментария\"!";

REM {и это строка \\комментария};
```

SELECT *формула*

Область применения: разрешается использовать только в формулах отбора и агентах.

Определяет критерий для отбора документов, которые будут обрабатываться. Используется в формулах отбора видов, формулах селективной репликации, формулах агентов. Обрабатываются только те документы, для которых ***формула***, возвращает значение "Истина". Ключевое слово SELECT должно быть самым первым оператором в формуле.

Для агентов критерий отбора может так же определяться в его окне свойств (например, **All unread documents in view** – все непрочитанные документы в виде), а также в объекте **Document Selection** из объектно-событийной панели агента. Тем не менее, даже в этом случае в теле агента можно использовать ключевое слово SELECT. Критерий отбора документов для

агента будет тогда определяться путем объединения перечисленных условий. При селективной репликации наблюдается похожая картина с использованием ключевого слова SELECT.

Пример 1. Выбор всех документов:

```
SELECT @All
```

Пример 2. Отбор документов в вид или при репликациях. Выбираются все документы, созданные по форме "Описание фирмы" и все их "потомки" - ответные документы всех уровней иерархии.

```
SELECT Form = "Описание фирмы" | @AllDescendants;
```

При использовании в формуле операции сравнения со значением поля, которого в документе нет, такая формула всегда возвращает значение 0 (FALSE). Для решения проблемы можно воспользоваться функциями @IsAvailable или @IsUnavailable.

Ключевое слово всегда должно быть стоять первым в операторе. При запоминании формулы ключевые слова всегда преобразуются к верхнему регистру. В операторах с ключевыми словами DEFAULT, ENVIRONMENT и FIELD значения ***имя_поля*** и ***переменная_окружения*** не должны быть текстовыми константами.

3.6 Функции

@-функции - это способ обращения из формул к встроенному в Domino набору процедур, выполняющих специализированные вычисления или производящих определенные действия в интерфейсе Notes.

Формат вызова функций:

```
@FunctionName( Arg1; Arg2; ...; ArgN );
```

Список аргументов заключается в скобки. Если аргументы не требуются, скобки опускаются. Друг от друга аргументы отделяются точкой с запятой. В большинстве случаев они должны иметь определенный тип данных.

Некоторым функциям, например, @Command, @PostedCommand, @DocMark, @GetPortsList, @PickList, @MailSend, @Name, @Prompt требуются аргументы - ключевые слова. Они заключаются в квадратные скобки.

Примеры.

```
@Prompt( [OK]; Title; Subject );

@Name( [CN]; AUTHOR );

@Command( [EditClear] );
```

Большинство @-функций в результате выполнения возвращают значение определенного типа. Иногда тип результата зависит от типов аргументов и еще каких-либо условий.

Пример. @Prompt([OkCancelList]; Title; Subject; Default; List) по кнопке OK возвращает выбранную строку текстового списка. Если ничего не выбрано, то возвращает истину (число 1), если нажата кнопка OK, или ложь (число 0), если нажат Cancel.

@-Команды - специальные функции, вызывающие немедленный отклик в интерфейсе пользователя. С помощью @-команд можно выполнить практически все стандартные команды меню. Кроме того, имеется ряд специализированных команд. Область применения @-команд - "Run once"-агенты, кнопки, акции, гиперобъекты и кнопки панели инструментов. Заметьте, что некоторые из команд действуют не во всех указанных контекстах.

Нельзя использовать @-команды в контексте, где отсутствует диалог с пользователем (например, агенты по расписанию, фоновые агенты, формулы полей и т.д.).

В Domino 4-й версии и выше есть две @-функции для выполнения команд:

```
@Command( [ИмяКоманды]; параметр1; ...; параметрN )
@PostedCommand( [ИмяКоманды]; параметр1; ...; параметрN )
```

Их первый аргумент - ключевое слово, определяющее имя команды. От него зависит состав остальных параметров.

Различие между @Command и @PostedCommand заключается в порядке их выполнения в формуле (более подробно рассмотрено в пункте "4.4").

Пример. Формула кнопки панели инструментов открывает БД на любом доступном сервере по ее ID реплики. Если база открыта успешно, формула выполняется дальше - создается новый документ и т.д.

```
REM "Если базу открыть не удалось, - Return";
@If( @Command( [FileOpenDBRepID]; "C3256310:002881A3" ); "";
     @Return( "" ));
@Command( [Compose]; ""; "Main Topic" );
@Command( [EditGotoField]; "Subject" );
@Command( [EditInsertText]; "New subject" );
@Command( [EditGotoField]; "Body" )
```


Примечание 1: когда имя БД (или другое имя файла) включается в команду в качестве параметра, пути доступа к файлам должны быть указаны в соответствии с требованиями операционной системы рабочей станции или сервера, на котором хранится БД. Если БД хранится на компьютере, работающем в Windows или OS/2, используйте формат записи:

```
"Сервер" : "Каталог\\Database.NSF", где Database.NSF - имя файла БД.
```

Примечание 2: при включении в формулу в качестве параметра имени вида или формы, не указывайте символ подчеркивание в имени, даже если он в параметре явно присутствует. Для вида "_Туры_По дате" при использовании его в формуле применяется следующий синтаксис:

```
"Туры\\По дате"
```

3.6.1 Побочные эффекты

Некоторые функции при выполнении вызывают «побочные эффекты». Основное предназначение @-функции произвести некоторые вычисления и возвратить определенное значение. Если функция помимо этого выполняет еще какие-либо действия, например, выводит диалоговое окно, то говорят что это функция с "побочным эффектом". Ниже приводится таблица с такими функциями.

@-функция	Побочный эффект
@Command, @PostedCommand	Чаще всего выполняется команда меню Domino
@DbColumn, @DbCommand, @DbLookup	Доступ в текущую или другую базу для поиска и получения данных
@DDEInitiate, @DDEExecute, @DDEPoke, @DDETerminate	DDE - диалог с другим приложением.
@MailSend	Текущий документ отправляется почтой Domino или создается и отправляется почтовое сообщение.
@Prompt, @PickList @DialogBox	Открывается диалоговое окно, результат возвращается функцией только по окончании диалога.

Если установлена переменная окружения NoExternalApps = 1, то не будет работать ни одна формула, содержащая функции из этого списка. Пользователь не увидит сообщения об ошибке, формула просто не будет выполняться.

3.6.2 Ограничения применимости @-функций и команд, налагаемые списком управления выполнением (ECL)

Список управления выполнением (ECL) влияет на работу ряда @-функций. Ниже приводится таблица, где по горизонтали перечисляются эти функции, а по вертикали указаны флаги доступа ECL. Если на пересечение соответствующей строки и столбца стоит символ "+", это говорит о том, что данная функция при отсутствии соответствующего флага выполняться не будет.

Функция	**Доступ к текущей базе данных** (Access to current database)	**Доступ к переменным окружения** (Access to environment variables)	**Доступ к базам данных сторонних фирм** (Access to non-Notes databases)	**Доступ к внешним программам** (Access to external programs)	**Возможность отправки почты** (Ability to send mail)	**Доступ к ECL рабочей станции** (Access to Workstation Security ECL)	**Доступ на чтение к другим базам данных** (Ability to read other database databases)	**Доступ на модификацию к другим базам данных** (Ability to modify other database)
@DbColumn							+	
@DbColumn (ODBC)			+					
@DbCommand			+					
@DbLookup							+	
@DbLookup (ODBC)			+					
@DDEExecute				+				
@DDEInitiate				+				
@DDEPoke				+				
@DDETerminate				+				
@DeleteDocument	+							
@DeleteField	+							
@EditECL						+		
@EditUserECL						+		
ENVIRONMENT		+						
@Environment		+						
@GetProfileField	+							
@MailSend					+			
@RefreshECL						+		
@SetDocField	+							
@SetEnvironment		+						
@SetProfileField	+							
@Unavailable	+							
@UpdateFormula Context	+						+	+
@URLGetHeader	+							
@URLOpen	+							

Аналогичные ограничения существуют и для @-команд:

Команда	**Доступ к файловой системе** (Access to the file system)	**Доступ к текущей базе данных** (Access to current database)	**Доступ к базам данных сторонних фирм** (Access to non-Notes databases)	**Доступ к коду внешних программ** (Access to external code)	**Доступ к внешним программам** (Access to external programs)	**Возможность отправки почты** (Ability to send mail)	**Возможность экспорта данных** (Ability to export data)
AdminSendMailTrace						+	
AgentEnableDisable				+			
AgentRun				+			
AgentSetServerName		+					
AgentTestRun				+			
AttachmentDetachAll	+						
AttachmentLaunch	+						
Clear		+					
DesignRefresh		+					
EditClear		+					
EditCopy		+					
EditCut		+					
EditDetach	+						
EditInsertFileAttachment	+						
EditInsertObject	+						
EditOpenLink		+					
EditPaste		+					
EditPasteSpecial		+					

EditUntruncate		+					
EmptyTrach		+					
ExchangeUnreadMarks		+					
Execute					+		
FileDatabaseCompact		+					
FileExport	+						+
FileImport	+						
FileOpenDatabase		+					
FileOpenDbRepID		+					
FilePrint		+					
FileSave		+					
FileSaveNewVersion		+					
Folder		+					
FolderDocuments		+					
FolderMove		+					
FolderRename		+					
MailForward						+	
MailForwardAsAttachment						+	
MailRequestCrossCert						+	
MailRequestNewName						+	
MailRequestNewPublicKey						+	
MailSend						+	
MailSendCertificateRequest						+	
MailSendEncryptionKey						+	
MailSendPublicKey						+	
ObjectOpen					+		
OpenDocument		+					

RemoveFromFolder		+					
ReplicatorReplicateHigh		+					
ReplicatorReplicateNext		+					
ReplicatorReplicateSelected		+					
ReplicatorReplicateWithServer		+					
ReplicatorSendMail		+					
ReplicatorSendReceiveMail		+					
ReplicatorStart		+					
RunAgent		+		+			
RunScheduledAgents		+		+			
SetCurrentLocation		+					
ToolsCategorize		+					
ToolsReplicate		+					
ToolsRunBackgroundMacros		+		+			
ToolsRunMacro		+		+			
ToolsScanUnreadSelected		+					
ViewRefreshUnread		+					

3.6.3 Ограничения применимости @-функций и команд для Web-приложений

В Web-приложениях следующие @-функции не работают, или результат их применения отличается от обычного при использовании в клиенте Notes.

Функция	Комментарии
@Certificate	
@DbCommand	В Web работает только с первым параметром "Domino"
@DDEExecute @DDEInitiate @DDEPoke @DDETerminate	
@DocMark @DeleteDocument @HardDeleteDocument @DocLock	
@DocChildren @DocDescendants @DocLevel @DocNumber @DocParentNumber @DocSiblings	Работают только в формулах столбцов.
@IsCategory	В документации сказано, что в Web не работает, однако исходя из личного опыта – функция работает в колонках видов.
@IsExpandable @Responses	
@DialogBox @PickList @Prompt @IsModalHelp	
@GetPortsList	
@GetFocusTable	
@FontList	
@Environment @SetEnvironment ENVIRONMENT	Для получения информации о пользовательских Web-переменных окружения необходимо использовать запросы к Common Gateway Interface (CGI) с предопределенными именами полей.
@MailSend	В Web не работают флаги [Encrypt] и [Sign].

@Domain @MailDbName @MailEncryptSavedPreference @MailEncryptSendPreference @MailSavePreference @MailSignPreference	
@IsAgentEnabled	
@IsDocBeingMailed	
@Unique @URLGetHeader @URLHistory	В документации сказано, что @Unique не поддерживается в Web-приложениях. Однако из опыта функция работает как в случае генерации случайных текстовых строк, так и при удалении из списка повторяющихся значений (правда, возможно не во всех контекстах)
@UserPrivileges	
@UpdateFormulaContext	
@Platform	Возвращает только платформу для сервера. Для того, чтобы различать пользователей Notes и Web используйте функцию @ClientType.

Аналогичные ограничения существуют и для @-команд. Большинство из них нельзя применять в Web-приложениях, т.к. @-команды ориентированы на интерфейс рабочей станции Lotus Notes. Исключение составляют команды:

- [CalendarFormat]
- [CalendarGoTo]
- [Clear]
- [CloseWindow]
- [Compose]
- [EditClear]

- [EditDocument]
- [EmptyTrash]
- [FileCloseWindow]
- [FileOpenDatabase]
- [FileSave]
- [Folder]
- [FolderDocuments]
- [MoveToTrash]
- [NavigateNext]
- [NavigateNextMain]
- [NaviagtePrev]
- [NavigatePrevMain]
- [NavNext]
- [NavNextMain]
- [NavPrev]
- [NavPrevMain]
- [OpenDocument]
- [OpenFrameset]
- [OpenHelpDocument]
- [OpenNavigator]
- [OpenPage]
- [OpenView]

- [RefreshFrame]
- [RemoveFromFolder]
- [RunAgent]
- [SwitchView]
- [ToolsRunMacro]
- [ViewChange]
- [ViewCollapse]
- [ViewCollapseAll]
- [ViewExpand]
- [ViewExpandAll]
- [ViewRefreshFields]
- [ViewShowSearchBar].

Некоторые из перечисленных выше команд (например, [ViewExpand] или [Folder]) могут использоваться при программировании под Web только в контексте апплетов видов.

4 Как выполняются формулы

4.1 Порядок выполнения

Domino выполняет действия в формулах слева направо, сверху вниз, завершая каждый оператор, перед тем как перейти к следующему, за исключением любого использования функции @PostedCommand и функции @Command с определенными значениями аргументов. Эти функции всегда выполняются после выполнения всех остальных операторов (более подробно рассмотрено в пункте "4.4").

```
(1)  Statement1;
(2)  Statement2;
(3)  Statement3;
(4)  Statement4;
(5)  Statement5;
```

Вы можете прервать процесс выполнения, используя функцию @Return. В R 6 язык формул стал поддерживать операторы циклов (@For, @While, @DoWhile и @Transform), которые позволяют выполнить несколько раз определенную последовательность операторов. Так же можно "пропустить" во время выполнения какие-либо операторы с помощью функции @If.

Пишите формулы в том порядке, в котором они должны выполняться. Если оператор должен работать только при выполнении некоторого условия, используйте функцию **@If** для проверки этого условия:

```
@If( Condition_1; TRUE_Statement_1;
   ...; ...;
      Condition_N; TRUE_Statement_N;
      FALSE_Statement );
```

Чтобы заставить Domino выполнить несколько операторов при заданном условии, вставьте функцию **@Do** внутрь **@If**:

```
@If( Condition;
     @Do( Statement_1; ...; Statement_N );
     FALSE_Statement );
```

4.2 Формулы, возвращающие значения

Большинство типов @-формул, если классифицировать их по месту применения (типу объекта, в котором они определяются), в результате выполнения должны возвращать значение. Значение может быть возвращено либо последним оператором формулы, либо функцией @Return, которая возвращает свой аргумент. В любом случае последним оператором в такой формуле должен быть оператор, возвращающий значение. Это может быть переменная, константа, @-функция или выражение, составленное с помощью операторов из перечисленных элементов. Операторы с ключевыми словами (кроме некоторых случаев с SELECT и FIELD) не возвращают значений, поэтому не могут стоять последними в таких формулах. Следующие типы формул должны возвращать результат:

- **Replication formula** (формула селективной репликации). Результат - истина или ложь в критерии SELECT для каждого документа базы;
- **Form formula** (формула формы). Результат - имя формы для открытия документа;
- **Selection formula** (формула отбора документов в виде). Результат - истина или ложь в критерии SELECT для каждого документа базы;
- **Column formula** (формула колонки). Результат должен быть пригоден для преобразования в текст;
- **Hide action formula** (формула скрытия акции). Результат - истина или ложь;
- **Formula pop-up** (формула "всплывающего" окна). Результат - текст;
- **Window title formula** (формула заголовка окна). Результат должен быть пригоден для преобразования в текст или число, за исключением случая, когда формула состоит из одного поля, любого типа;
- **Section access formula** (формула доступа к секции). Результат - имя или список имен;
- **Insert subform formula** (формула вычисляемой субформы). Результат - имя субформы (текст);
- **Section title formula** (формула заголовка секции). Результат - текст или число, за исключением формул, состоящих из одного поля любого типа;

- **Hidden paragraph formula** (формула видимости объекта). Результат - истина или ложь;
- **Default value formula** (значение поля по умолчанию). Результат должен быть пригоден для сохранения в данном поле;
- **Input translation formula** (формула преобразования значения поля). Результат должен быть пригоден для сохранения в данном поле;
- **Input validation formula** (формула проверки значения поля). Результат - истина или ложь;
- **Input enabled formula** (формула разрешения ввода значения в поле). Результат - истина или ложь;
- **Computed field formula** (формула вычисляемого поля). Результат должен быть пригоден для сохранения в данном поле;
- **Keyword field formula** (формула ключевых слов). Результат – значение или список значений, пригодный для сохранения в данном поле;
- **Image resource formula** (формула загружаемого изображения). Результат - имя изображения из дизайна ресурсов;
- **Computed text formula** (формула вычисляемой надписи). Результат - текст.

Приведенный выше список не полностью описывает все типы формул, которые должны возвращать результат. В качестве примера можно привести формулу скрытия столбца в виде или формулу строки аутлайна. К сожалению, в документации отсутствует структурированное описание данных контекстов применения формул.

4.3 Формулы, выполняющие последовательность действий

Перечисленные ниже типы формул могут не возвращать значения в результате выполнения, поскольку его обычно некуда возвращать. Такие формулы только выполняют заданную последовательность операторов. Будем называть такие @-формулы @-программами:

- **Agent formula** (формулы агентов) - выполняются при запуске соответствующего агента. Формула выполняется для каждого документа, удовлетворяющего критерию отбора документов агента и внутренним условиям алгоритма агента;

- **Action formula** (формула акций) - выполняются при нажатии на соответствующую кнопку-акцию в виде или форме;
- **Button formula** (формула кнопки) допустима для использования в навигаторе, форме или RichText-поле документа;
- **Toolbar button formula** (формулы кнопки панели инструментов) - выполняются однократно, по нажатию на соответствующую кнопку. До версии R6 эти кнопки назывались **SmartIcon buttons** (кнопки активных пиктограмм);
- **Action hotspot formula** (формула гиперобъекта) применяется в навигаторах, формах и RichText-полях.

4.4 Выполнение формул, содержащих вызовы @-команд

Функция @**PostedCommand** используется в 4-й и выше версиях Domino как замена для @Command 3-х с целью обеспечения совместимости, поскольку @Command 4-й и выше версии выполняется иначе. Формула с @Command, написанная в 4-й и выше версии не может быть выполнена в 3-й.

При выполнении указанных в формуле действий Domino всегда выполняет функции @PostedCommand (как @Command в 3-й версии) в последнюю очередь. Если в формуле имеется несколько вызовов функции @PostedCommand, они обрабатываются в порядке появления, но после всех остальных операторов и @-функций. Даже если для изменения порядка действий используется @Do, функции @PostedCommand выполняются в последнюю очередь.

Например, предположим, что приведенная ниже формула содержит ряд выражений, заключенных в скобки внутри функции @Do:

```
FIELD X :=
    @If( условие;
        @Do( @PostedCommand1; @Prompt; @PostedCommand2 );
        @Error )
```


При вычислении, если условие истинно, указанные в @Do действия будут выполняться в следующем порядке:

```
(1) @Prompt
(2) @PostedCommand1
(3) @PostedCommand2
```

@Do выполняет все, не относящиеся к @PostedCommand действия в порядке слева направо; а после их завершения выполняет все @PostedCommand в том же порядке.

Предположим, вы пишете формулу, которая должна выполнять @PostedCommand, проверку @If, а затем несколько других действий, за которыми следует еще одна @PostedCommand. Domino выполнит действия в такой последовательности: проверка условия и соответствующее действие, затем прочие "обычные" действия, затем обе @PostedCommand.

Программа

```
(4) @PostedCommand( [имя_команды]; параметр );
(1) @If( Условие; True_действие; False_действие );
(2) FIELD X := "Текст";
(3) FIELD Y := "Следующий текст";
(5) @PostedCommand( [имя_команды] )
```

Чтобы "заставить" Domino выполнить первую @PostedCommand перед проверкой @If и следующими за ней действиями, можно создать формулу-подпрограмму (ее нужно определить в "Run once" - агенте) и запускать ее с помощью дополнительной @PostedCommand:

Программа (в акции, кнопке, кнопке панели инструментов, гиперобъекте, Run once - агенте)

```
(1) @PostedCommand( [имя_команды]; параметр );
(2) @PostedCommand( [ToolsRunMacro]; "(Subprogram)" )
```

(Подпрограмма) (Run once - агент, запускаемый из списка агентов, с именем **Subprogram**)

```
(3) @if( Условие;  True_действие;  False_действие )
(4) FIELD X := "Текст";
(5) FIELD Y := "Следующий текст";
(6) @PostedCommand( [имя_команды] )
```

Вызовы функций @**Command** происходят при выполнении формулы в том порядке, в котором встречаются (вместе с другими операторами и @-функциями). Однако имеются исключения из этого правила. Перечисленные ниже в таблице типы @Command выполняются аналогично @PostedCommand, то есть после всех остальных действий. В R 6 добавлен целый ряд однотипных команд, которые выполняют те же действия, но немедленно, а не после других команд:

Команда, выполняющаяся после всех остальных @-команд	**Новый аналог в R 6, выполняющийся немедленно**
[EditClear]	[Clear]
[EditProfile]	[EditProfileDocument]
[FileCloseWindow]	[CloseWindow]
[FileDatabaseDelete]	[DatabaseDelete]
[FileExit]	[ExitNotes]
[Folder]	[FolderDocuments]
[NavigateNext]	[NavNext]
[NavigateNextMain]	[NavNextMain]
[NavigateNextSelected]	[NavNextSelected]
[NavigateNextUnread]	[NavNextUnread]
[NavigatePrev]	[NavPrev]
[NavigatePrevMain]	[NavPrevMain]
[NavigatePrevSelected]	[NavPrevSelected]
[NavigatePrevUnread]	[NavPrevUnread]
[ReloadWindow]	[RefreshWindow]
[ToolsRunBackgroundMacros]	[RunSheduledAgents]
[ToolsRunMacro]	[RunAgent]
[ViewChange]	[SwitchView]
[ViewSwitchForm]	[SwitchForm]

5 Обработка ошибок в формулах

Синтаксические ошибки "отлавливаются" Domino при нажатии кнопки  , требующей принять изменения в формуле, либо при закрытии окна редактирования формулы. Сообщение об ошибке выводится либо в диалоговом окне (Рис. 5.1), либо в нижнем колонтитуле окна с формулой (Рис. 5.2). Затем в формуле выделяется фрагмент, вызвавший ошибку. Вот некоторые примеры:

- No main or selection expression in formula (в формуле нет "главного" выражения) - случается, если последний оператор в формуле, которая по своему назначению должна возвращать значение (подробности в разделе "4.2"), не может этого делать (например, оператор с ключевым словом). Самое простое "формальное" решение такой проблемы - добавить в формулу в качестве последнего оператора константу "" (пустую строку).

Пример. В формуле поля Field1 (см. ниже), значение которого не важно, должно быть изменено значение поля Field2. В результате выполнения формулы поле Field2 получит значение NewValue, а поле Field1 - "".

```
FIELD Field2 := NewValue;

"";
```

- An operator or semicolon was expected but non was encountered - здесь должен быть оператор (арифметический, логический и т. п.) или точка с запятой.

- @If must have an odd number of arguments - функция @If должна иметь нечетное число аргументов.

- Unknown [KeyWord] for @Function... - неизвестное ключевое слово в качестве аргумента функции.

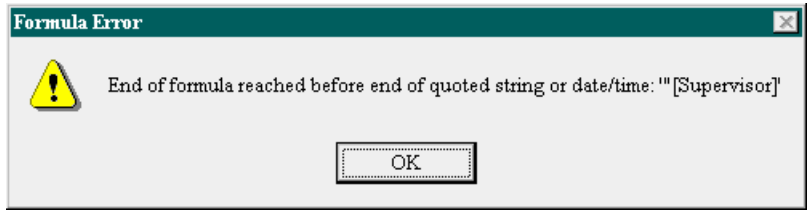


Рис. 5.1 Сообщение о синтаксической ошибке: конец формулы достигнут ранее конца текстовой константы или константы типа дата/время

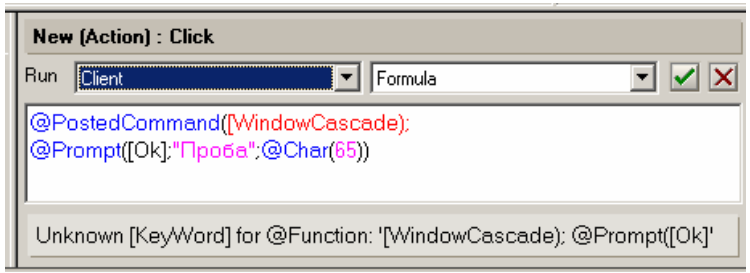


Рис. 5.2 Сообщение о синтаксической ошибке: неизвестное ключевое слово для @-функции: '[WindowCascade);@Prompt([OK]'

Ошибки во время выполнения формулы (Run-time) могут быть поделены на три группы:

- Неожиданные ошибки проектирования, которые не должны встречаться пользователю. Например, если вызов функции содержит недостаточное число параметров, во время выполнения будет выведено сообщение: "Insufficient arguments for @function".

- Ошибки проектирования, которые не отслеживает Domino, также не должны влиять на работу пользователя. Например, если вы пытаетесь показать с помощью @Prompt значение числового типа, @Prompt работает, но показывает пустое значение.

- Ожидаемые ошибки, которые могут быть вызваны пользователем или какими-либо "внешними" условиями, например, пользователь ввел в диалоговом окне имя несуществующей базы. Такие ошибки нельзя предотвратить совсем, но их можно предвидеть и включить в формулы соответствующие проверки.

Для отладки Run-time ошибок в Domino не предусмотрен соответствующий механизм, аналогичный отладчику LotusScript. Предлагаемые пути решения:

- для организации точек останова и вывода результирующих значений использовать функцию @Prompt, которую после успешной отладки удалить из тела формулы. Следует помнить, что @Prompt применим не везде (например, в формуле отбора документов). В этом случае предлагается записать отлаживаемую формулу в элемент дизайна, в котором функция @Prompt работает (например, кнопку-акцию или кнопку панели инструментов). После отладки - перенеси формулу в нужный вам объект дизайна;

- в качестве альтернативы функции @Prompt, можно создать в форме поля, куда будут записываться промежуточные результаты выполнения формулы. После отладки удалить эти поля.

Следующие функции помогают обрабатывать ошибки периода выполнения:

- **@IsError(value)** - возвращает 1 (True), если значение value, которое содержится в поле или переменной, или получается в результате вычисления выражения, содержит ошибку;

- **@IfError(выражение1, выражение2)** – возвращает значение выражения1, если выражение1 не содержит ошибки, и значение выражения2 или пустую строку, если в при вычислении выражения1 произошла ошибка;

- **@Error** - генерирует ошибку, например, чтобы сделать "ошибочным" значение поля;

- **@Failure(message)** - показывает сообщение в окне при проверке значения поля (используется только в формулах проверки редактируемых полей);

- **@Success** - всегда возвращает значение 1 (True) (используется только в формулах проверки редактируемых полей);

- **@Return(value)** - прерывает выполнение формулы и возвращает значение value, например, в случае положительного результата проверки на какую-либо ошибку.

Domino генерирует состояние ошибки в случае, когда пользователь вводит в поле значение, не соответствующего предусмотренного разработчиком типа данных. Например, в числовое поле вводится нечисловое значение. Ситуация ошибки генерируется при попытке сохранения такого документа. При этом выдается сообщение "Cannot convert text to a number" (не могу преобразовать текст в число). Разработчик может изменить текст сообщения с помощью функции @Failure, использую ее в формуле проверки значения поля.

Для формул, отличных от формул полей (например, формул агентов или кнопок), для проверки содержимого полей не требуется возникновения ситуации сохранения документа. В этом случае соответствующие проверки и возможная генерация ошибки происходит немедленно, по факту запуска данной формулы.