119270, Москва, Лужнецкая наб., д. 6,
стр.1, офис 214, ООО «ЭР СИ О»
Тел./факс: (495) 287-98-87
E-mail: info@rco.ru
http://www.rco.ru

## Руководство администратора

# RCO Pattern Extractor – компонент выделения конструкций в тексте

# (в составе библиотеки RCO Fact Extractor 2.9)

Москва, 2011

В содержание данного документа могут быть внесены изменения без предварительного уведомления. Названия организаций, имена и даты, используемые в качестве примеров, являются вымышленными, если не оговорено обратное.

# Содержание

# Введение

Компонент **RCO Pattern Extractor** предназначен для анализа текста и распознавания в нем различных конструкций в соответствии с образцами, заданными на формальном языке. Область применения **RCO Pattern Extractor** в первую очередь включает в себя выделение сложных элементов текста и специальных объектов, чьи названия отличаются особым написанием (наименования юридических лиц, товаров, адресов, номеров и т.п.). В большинстве случаев оно выходит за рамки правил грамматики русского языка, является неочевидным и трудно формализуемым.

Мощный язык описания конструкций текста (CAPE) позволяет оперировать как формальными особенностями написания текста, используя, в частности, язык регулярных выражений, так и всеми грамматическими атрибутами слов – частью речи, родом, числом, падежом и т.д. Компонент использует морфологический анализатор русского языка.

Образцы сложных конструкций-объектов могут строиться иерархически, включать образцы более простых, что позволяет постепенно наращивать мощность системы целевых описаний. Грамматика языка описания образцов обеспечивает как бесконтекстное, так и контекстно-зависимое распознавание конструкций-объектов.

В данном руководстве описаны принципы работы компонента **RCO Pattern Extractor** для выделения в тексте конструкций произвольной природы в соответствии с заданными образцами. Описана грамматика формального языка, используемого для построения шаблонов конструкций.

## Отличия от предыдущих версий

Ниже перечислены отличия данной версии компонента от его предыдущих версий.

### Отличия от версии 2.4

- Введен нечеткий поиск по словарям с опечатками (см. раздел «Словари в модуле выделения объектов»);
- Введены функции для постобработки выделенных из текста значений с целью последующей стандартизации (см. раздел «Использование функций» и «Приложение 3»);
- Введены функции проверки соответствия числовых цепочек заданным формулам для проверки контрольных сумм реквизитов (см. разделы «Использование операторов» и «Приложение 3»);
- Введена возможность определения пользователем собственных функций обработки символьной информации для использования в выражениях стандартного языка описания конструкций (см. раздел «Использование функций»);
- Введены возможность повторного применения правил выделения объектов заданное число раз или пока срабатывают правила (см. раздел «Семантика применения правил»);
- Добавлены новые условия выделения объектов: наилучшее покрытие правилом цепочки лексем, покрытие всех возможных цепочек лексем одним/всеми правилами, ограничение на число лексем в правиле (см. раздел «Семантика применения правил»).

# Основные определения

В данном руководстве используются следующие понятия.

*Лексема* – цепочка символов, представляющая элементарную единицу текста с точки зрения принятой в компоненте схемы обработки. Лексемами могут быть слова, знаки препинания и различные символьные последовательности.

*Объект* – цепочка *лексем* или *объектов*, удовлетворяющая заданным ограничениям и обрабатываемая компонентом как единое целое. Лексемы – это элементарные объекты. В состав объекта могут входить другие объекты.

*Атрибут объекта* – характеристика некоторого свойства *объекта*, к которой применимы операции сравнения и присвоения в рамках принятой в компоненте схемы обработки. Все атрибуты подразделяются на предопределенные атрибуты, автоматически присваиваемые объекту на основании анализа соответствующего текста, и атрибуты, задаваемые пользователем для характеристики требуемых свойств объекта.

*Описание объекта* – набор всех его *атрибутов*, несущих информацию об особенностях написания *объекта*, его грамматических и семантических характеристиках.

*Образец объекта* – логическое выражение на формальном языке, задающее условия, которым должны удовлетворять значения *атрибутов лексем (объектов)*, входящих в состав объекта. Используется для *распознавания* объекта в тексте.

*Распознавание объекта* – совпадение его с *образцом*.

*Выделение объекта* – объединение цепочки *лексем*, соответствующей *распознанному объекту*, в новый объект с присвоением ему заданного *описания*.

*Правило выделения объекта* – пара вида «*образец* → *описание*», записанная на формальном языке и формирующая заданное описание объекта в случае его *выделения* в соответствии с образцом.

*Целевой объект* – конечный *объект, выделяемый* компонентом для задач пользователя в соответствии с заданным *правилом обработки*.

# Принципы работы

Компонент **RCO Pattern Extractor** состоит из трех основных подсистем:

- Модуль фрагментации текста – производит фрагментацию текста на предложения и лексемы с получением формальных и грамматических атрибутов лексем;
- Словарный модуль – производит сравнение цепочек лексем с образцами, заданными в словарях, и выделяет первичные объекты, присваивая им заданные атрибуты;
- Модуль выделения объектов – производит сравнение цепочек лексем и объектов с образцами, заданными правилами на формальном языке, и выделяет соответствующие образцам объекты, присваивая им заданные атрибуты.

## Модуль фрагментации текста

В первую очередь модуль производит фрагментацию текста на предложения и лексемы на основе знаков препинания и тегов HTML, если таковые присутствуют в тексте.

### Критерии конца лексемы

- Структурные теги HTML и парные к ним закрывающие теги;
- Бесконтекстный разделитель;
- Контекстно-зависимый разделитель, после которого следует другой контекстно-зависимый либо бесконтекстный разделитель.

Бесконтекстными разделителями являются символы из следующего набора: **“***()”, “|”, “{}”, “[]”, “<>”, “/”*, а также различные виды кавычек (“"”, 0x84, 0x91, 0x92, 0x93, 0x94, 0xAB, 0xBB) и пробельные символы (пробел, табуляция, символы конца строки).

Контекстно-зависимыми разделителями являются символы из следующего набора: **“***;”, “.”, “,”, “!”, “?”, “\”, “:”, “–“ и другие виды тире (0x96, 0x97, 0xAC, 0xAD).*

Все разделители, удовлетворяющие критериям конца лексемы, выделяются в отдельные лексемы. Исключение составляют пробельные символы и все теги HTML, которые просто удаляются.

Переносы известных слов исправляются.

### Критерии конца предложения

- Структурные теги HTML и парные к ним закрывающие теги;
- Восклицательный и вопросительный знаки, стоящие в конце лексемы или выделенные в отдельную лексему;
- Точка, стоящая в конце лексемы или выделенная в отдельную лексему, после которой следует текст с заглавной буквы.

Дополнительно учитываются общепринятые сокращения (расширяемый список загружается из файлов *shortcuts.txt* и *shortcuts_endsent.txt*), инициалы и т.п., точки, в конце которых анализируются по отдельным правилам при определении конца предложения.

Учитываются правила описания диалогов в тексте и т.п.

# Обработка лексем

После фрагментации текста модуль производит анализ грамматических характеристик лексем и особенностей их написания.

На выходе модуля текст преобразуется в последовательность отдельных предложений, каждое из которых представляет собой цепочку первичных описаний лексем – элементарных объектов, подлежащих последующему анализу. Описание лексемы содержит набор предопределенных атрибутов, таких как: строка лексемы, тип лексемы (известное/неизвестное слово русского языка, латинское слово, специальная конструкция), особенности написания (верхний/нижний регистр), множество грамматических характеристик и т.п. Полное описание множества предопределенных атрибутов и их возможных значений приведено в Приложении 1.

Дальнейшее распознавание и выделение объектов производится по каждому из предложений в отдельности.

# Словарный модуль

Модуль производит сравнение цепочек лексем, заданных своими описаниями, с цепочками лексем из заданных словарей – словарными образцами. Распознанная цепочка выделяется в новый объект-лексему, который получает свое описание и замещает собой распознанную цепочку. Результатом работы модуля вновь является цепочка описаний объектов.

Прежде всего, в словарном модуле выделяются и описываются различные ключевые слова и словосочетания, которые входят в состав сложных целевых объектов и необходимы для их распознавания на следующем этапе – например, слова *«фирма»*, *«ООО»*, *«открытое акционерное общество»* и другие необходимы для распознавания сложных названий организаций. Однако здесь же могут выделяться конкретные целевые объекты, например *«Московский государственный университет им. М.В. Ломоносова»*, *«МГУ»*.

Словарные образцы объектов позволяют задавать три способа распознавания: сопоставление всех лексем в цепочке с учетом морфологии (во всех грамматических формах), сопоставление всех лексем в заданных грамматических формах независимо от регистра, точное сопоставление цепочки как есть.

Режим работы словаря задается параметром `loading` в xml-теге `<dictionary>` файла **config.xml**. Он может принимать три значения: *full*, *compact* и *fast*. В режиме *"full"* поиск слов/словосочетаний выполняется максимально точно, в режиме *"compact"* минимизируется объем занимаемой словарем оперативной памяти, в режиме *"fast"* скорость поиска максимальна.

В результирующее описание выделенного объекта входят:

- Текст строки, соответствующей исходному тексту цепочки;
- Синоним строки, если задан явно;
- Грамматические атрибуты – наследуются от заданного слова в цепочке либо задаются явно в словаре;
- Семантический тип – задается в словаре;
- Прочие атрибуты, описанные в разделе «Присвоение атрибутов».

## Пример 1. Словарь частей наименований организаций

```
Officials:KeyWord // Семантический тип, присваиваемый объектам
ООО   SYN        // Распознавать как есть
Открытое акционерное MAIN общество    MSYN       ООО
```

В последнем случае распознавание производится с учетом морфологии во всех грамматических формах. В выделенном объекте все грамматические атрибуты наследуются от слова «*общество*», перед которым указано ключевое слово *MAIN*, а исходная строка заменяется синонимом «*ООО*».

# Модуль выделения объектов

## Принципы функционирования

Прежде чем перейти к описанию следующего модуля, рассмотрим общие принципы, в соответствии с которыми он производит выделение объектов.

Распознавание нового объекта происходит в результате сравнения с заданным образцом некоторой цепочки, которая состоит из описаний других выделенных объектов или просто лексем. Исходной цепочкой для анализа является последовательность лексем, полученных с выхода модуля фрагментации текста и словарного модуля.

Распознавание осуществляется в ходе последовательного движения слева направо по цепочке описаний, соответствующей предложению. При этом на каждом шаге все подцепочки, находящиеся справа от текущего элемента, сравниваются с образцами объектов на предмет совпадения. Так как фрагменты цепочек, соответствующих распознаваемым объектам, могут перекрываться, операция выделения всех непересекающихся объектов является в принципе неоднозначной.

Ввиду наличия множества альтернатив при выделении объектов для окончательного принятия решения необходимо работать не с отдельной цепочкой, а с целым графом описаний объектов.

При распознавании каждого объекта в соответствии с образцом происходит его выделение и формируется описание нового объекта с заданными атрибутами, которое добавляется в граф описаний в качестве альтернативной цепочки (вместо того, чтобы сразу заменять собой распознанную цепочку).

Выделение может происходить в несколько фаз, когда на каждой последующей фазе выделяются все более сложные объекты, включающие в себя объекты, выделенные на предыдущих фазах.

В общем случае правила каждой фазы работают не на цепочке, а на целом графе описаний объектов, являющемся результатом работы на всех предыдущих этапах. То есть правила каждой фазы могут работать со всеми объектами, выделенными на предыдущих фазах, и добавлять объекты с их описаниями в граф описаний. При этом между вновь созданным объектом и объектами распознанной цепочки устанавливаются отношения подчинения.

Ниже приведен пример графа описаний (см. Рисунок 1). Здесь исходная цепочка описаний лексем с номерами 1-6 соответствует предложению «*В МГУ им. М.В. Ломоносова критикуют...*» и была получена с выхода предобработчика текста (фаза 0).

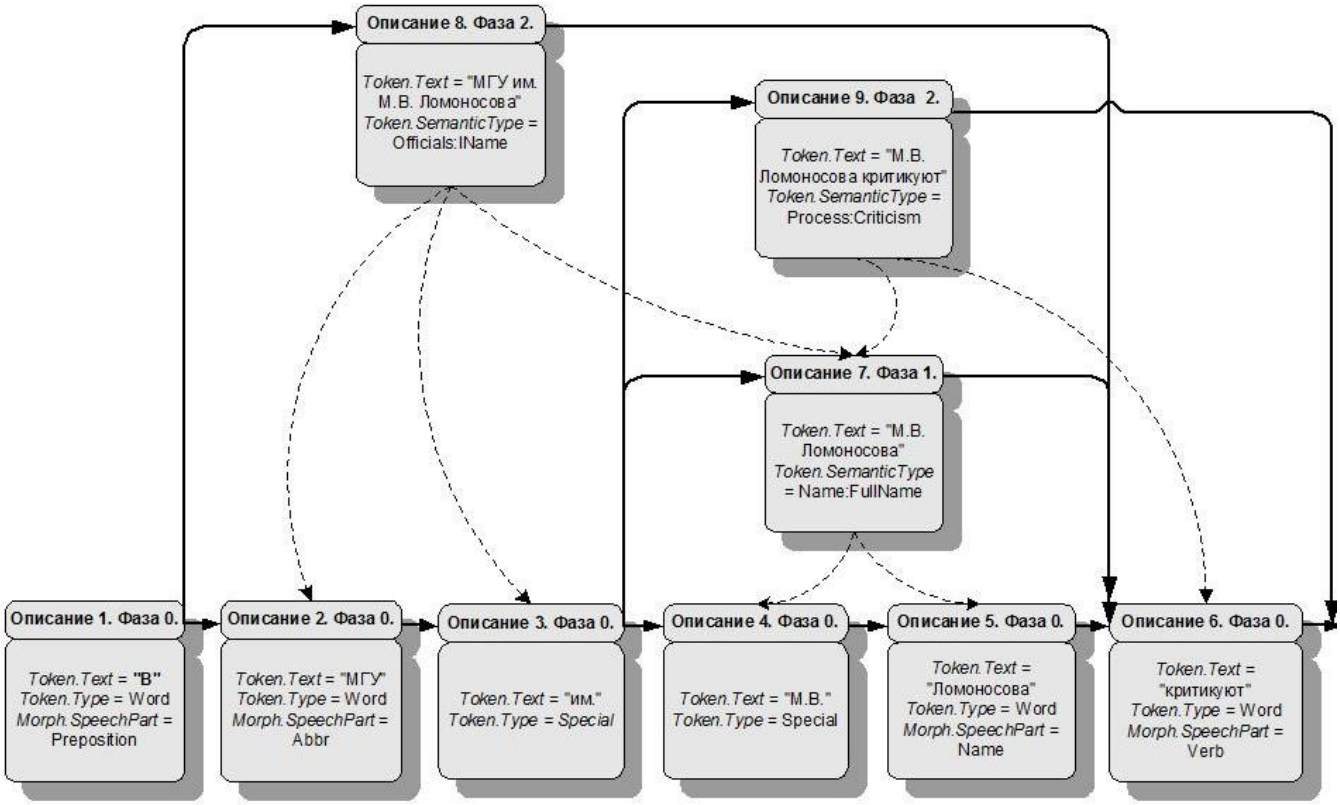



**Рисунок 1. Пример графа описаний.**

На фазе 1 последовательность описаний 4-5 можно выделить в объект *«М.В. Ломоносова»* с описанием 7 (`SemanticType` = *Name:FullName*), добавляющим новую цепочку в граф.

На фазе 2 цепочку описаний 2, 3, 7 можно выделить в объект «*МГУ им. М.В. Ломоносова*» с описанием 8 (`SemanticType` = *Officials:Name*), вносящим еще одну цепочку в граф.

На этой же фазе цепочка описаний 6 и 7 может быть выделена в объект «*М.В. Ломоносова критикуют*» с описанием 9 (`SemanticType` = *Process:Criticism*).

Как видно, любой из путей в этом ориентированном графе **{***1,2,3,4,5,6***}**, **{***1,2,3,7,6***}**, **{***1,2,3,9***}**, **{***1,8,6***}** представляет корректное (с точки зрения использованных правил) покрытие предложения выделенными непересекающимися объектами.

При этом, очевидно, наиболее логичной интерпретацией графа является выбор в нем кратчайшего пути **{***1,8,6***}**, от начала предложения к его концу – цепочки из трех объектов.

Таким образом, окончательно будет принято решение о выделении целевого объекта *«МГУ им. М.В. Ломоносова»*, а цепочка *«М.В. Ломоносова критикуют»*, распознанная в соответствии с образцом для критики физического лица, выделяться в объект не будет.

# Свойства модуля

Модуль осуществляет распознавание и выделение объектов в соответствии с правилами, заданными на формальном языке.

Грамматика языка поддерживает задание последовательности фаз выделения объектов, а каждая фаза состоит из набора правил вида «образец → описание».

Фазы выполняются последовательно, и каждая фаза может обрабатывать цепочки описаний всех объектов, выделенных в результате работы предыдущих фаз.

Левая часть правила состоит из образца, который может содержать логические операции и регулярные выражения над цепочками описаний, например “*”, “?”,”+”.

Правая часть – из выражений, задающих результирующее описание выделенного объекта.

## Пример 2. Правило для выделения географических названий

```
Rule: GeoNameRule
(({Token.SemanticType == "Geo:Adj"})
 (({Token.Text =| "регион"}) | ({Token.Text =| "район"}))
):geo_name
-->:geo_name.Token = { SemanticType="GeoName:GeoName", Text =
   geo_name.Text }
```

Левая часть данного правила устанавливает образец для сравнения атрибутов описаний объектов в распознаваемой цепочке: последовательное упоминание в тексте прилагательного с семантическим типом *Geo:Adj* и любого из заданных слов в любых грамматических формах. При этом предполагается, что семантический тип «географического» прилагательного будет ранее присвоен объекту при выделении на предыдущей фазе или в словарном модуле. В результате будут выделены все объекты типа *«Воронежский регион»* или *«Подольский район»*.

Правая часть правила указывает, что у выделенного объекта атрибуту `Token.SemanticType` будет присвоено значение *GeoName:GeoName,* а в атрибут `Token.Text` будет скопирован текст всей распознанной цепочки.

Для удобства задания правил интерпретатор языка позволяет использовать механизм макросов. При помощи макросов можно описывать фрагменты распознаваемых образцов (или целые образцы) для их последующего использования в различных правилах. Например, введение тройки макросов позволяет задать более сложное правило в компактном виде.

## Пример 3. Правило для выделения географических названий

```
// заданное в словаре "географическое" прилагательное
Macro: GEOADJ
(
 ({Token.SemanticType == "Geo:Adj"})
)
// неизвестное прилагательное с большой буквы
Macro: UNKNOWN_GEOADJ
(
({Token.Type == "UnknownWord", !Token.OrthographicType ==
   "FirstSmall", Morph.SpeechPart == "Adjective"})
)
// ключевое слово
Macro: GEO_KEYPOST
(
 ({Token.Text == "регион"})| ({Token.Text == "район"})
)
// правило
Rule: GeoNameRule
((GEOADJ | UNKNOWN_GEOADJ) GEO_KEYPOST):geo_name
-->:geo_name.Token = { SemanticType = "GeoName:GeoName", Text =
   :geo_name.Token.Text }
```

# Семантика применения правил

Семантика применения правил, установленная в компоненте, позволяет задать общую схему принятия решения о выделении объектов в случае наличия неоднозначностей.

Семантика применения правил задается для каждой фазы в секции `<rule-phase>` файла **config.xml** в виде параметров этого xml-тега и определяет набор исходных описаний, по которым производится выделение объектов.

Параметр `control` определяет порядок обхода цепочек описаний в графе описаний и может принимать следующие значения:

- *appelt* – анализируются все цепочки описаний в графе описаний, полученные в результате выделения объектов на предыдущих фазах;
- *optimal* – анализируется единственная цепочка описаний, соответствующая верхнему пути в графе описаний (т.е., пути, который не включает в себя подчиненных описаний).

Необязательный параметр `accept` определяет порядок выбора цепочек описаний в случае, когда правила распознают пересекающиеся цепочки:

- *normal* (исходное значение) – для каждой анализируемой цепочки осуществляется поиск наиболее длинной подцепочки, распознаваемой в соответствии с образцом какого-либо правила и происходит выделение нового объекта, после чего осуществляется поиск следующей наиболее длинной распознанной подцепочки описаний и выделение следующего объекта и так далее. В итоге при взаимном перекрытии цепочек, распознаваемых правилами, неоднозначность разрешается в пользу первой наиболее длинной цепочки (оптимальный компромисс между скоростью работы и качеством разрешения неоднозначностей).
- *longest* – для каждой анализируемой цепочки осуществляется поиск всех возможных подцепочек, распознаваемых в соответствии с образцами каких-либо правил, после чего выбирается самая длинная подцепочка и удаляются все пересекающиеся с ней подцепочки, далее из оставшихся подцепочек выбирается следующая по длине и удаляются пересекающиеся с ней подцепочки и так далее, пока остаются пересекающиеся подцепочки. В итоге при взаимном перекрытии цепочек, распознаваемых правилами, неоднозначность разрешается в пользу самой длинной цепочки (максимальное качество разрешения неоднозначностей).
- *first* – для каждой анализируемой цепочки выделение нового объекта происходит как только находится подцепочка, распознаваемая в соответствии с образцом какого-либо правила, после чего продолжается поиск, начиная с конца распознанной подцепочки. Данный режим обеспечивает самое быстрое выделение объектов, предполагая при том отсутствие неоднозначностей.
- *every* – для каждой анализируемой цепочки осуществляется поиск всех возможных подцепочек, распознаваемых в соответствии с образцами каких-либо правил, и по всем подцепочкам формируются новые объекты. В итоге в графе описаний сохраняются результаты работы всех подходящих правил вне зависимости друг от друга.
- *all* – эквивалентно режиму *every*, но если имеются несколько перекрывающихся подцепочек, распознаваемых одним и тем же правилом, то выбирается только самая длинная подцепочка, а остальные удаляются. Подцепочки, распознаваемые разными правилами, могут пересекаться как угодно.

Необязательный параметр `iterate` определяет количество повторений выполнения данной фазы. Значение по умолчанию – 1, в этом случае фаза выполняется один раз. В общем случае возможно любое целое число или строка "*cyclic*". В каждом следующем выполнении фазы обрабатываются цепочки описаний всех объектов, выделенных в результате работы предыдущих фаз. Если задано цикличное выполнение фазы ("*cyclic*"), то фаза повторяется, пока удается найти хотя бы одну подцепочку, распознаваемую в соответствии с образцом какого-либо правила данной фазы. Очевидно, что комбинация параметров *cyclic* и *appelt* в одной фазе недопустима, так как может привести к бесконечному циклу.

В зависимости от установленного в секции `<options>` файла **config.xml** значения параметра `cape-extraction` конечным результатом работы модуля, доступным через программный интерфейс, будет являться либо верхняя последовательность описаний (т.е. описаний, не подчиненных никаким другим) в графе описаний (значения *normal* или *direct*), либо полный перечень всех описаний объектов, включая исходные лексемы (значение *full*).

Первый режим является типовым и обычно используется при обработке текста, включающей в себя последующие фазы лингвистического анализа (в частности, синтаксический разбор).

Во втором режиме (`cape-extraction=`*full*) все последующие стадии лингвистического анализа предложения становятся некорректными и автоматически отключаются. Этот режим предназначен для обработки текстов специального вида, например записей в БД, где на выходе модуля может быть использована дополнительная информация для разрешения неоднозначности. В частности, для разбора записей почтовых адресов в БД прикладное приложение может использовать справочник (КЛАДР[1]) для выбора наиболее вероятного варианта разбора.

## Словари в модуле выделения объектов

В начале любой фазы работы модуля выделения объектов можно выполнить в специальном словаре поиск строк, составленных из подцепочек объектов. При составлении строк объекты графа описаний обходятся в прямом направлении (от первого до последнего) в соответствии с установленным порядком обхода, определяемым параметром `control`. Рассматриваются все возможные подцепочки объектов. Ищутся строки, полученные конкатенацией текстовых атрибутов `Token.Text` через пробел. Каждая такая строка сравнивается со всеми терминами словаря по нечеткому критерию сравнения, при этом вычисляется расстояние между строкой и термином по биграммной метрике. Если эта величина меньше максимального расстояния, задаваемого параметром `max-distance` в конфигурационном файле **config.xml**, то над цепочкой объектов, соответствующих строке, выделяется новый объект. В этом новом объекте проставляются те же атрибуты, что перечислены в словарном модуле, за исключением грамматических атрибутов, и дополнительно вводится атрибут `Cape.Relevance`, принимающий значение (`1 - distance / stringsize`), где `distance` – расстояние между строкой и термином, `stringsize` – число символов в исходной строке.

В случае перекрытия цепочек объектов при распознавании выбирается цепочка, содержащая наибольшее количество символов, совпадающих со строкой словаря (если распознавание точное, то выбирается наиболее длинная по количеству текстовых символов цепочка). Остальные пересекающиеся с ней цепочки-кандидаты удаляются. Данный способ выбора эквивалентен режиму *longest*, задаваемому параметром *accept*.

[1] КЛАДР – Классификатор адресов России. Поддерживается ФГУП ГНИВЦ ФНС России. Описание доступно на сайте http://www.gnivc.ru.

# Настройка описаний объектов

Выделяемые объекты могут описываться двумя способами:

- Словарными образцами – в файлах ***.dct**. Список подключаемых словарей указывается в секции `<dictionaries>` файла **config.xml**.
- Правилами выделения объектов – в файлах ***.rul**. Список подключаемых правил с указанием последовательности фаз их применения задается в секции `<cape-rules>` файла **config.xml**.

## Словарные образцы объектов (файлы *.dct)

Каждый файл ***.dct** содержит образцы цепочек лексем, которые могут быть выделены в словарном модуле как объекты.

В случае перекрытия цепочек при распознавании выбирается наиболее длинная из цепочек, указанных в словарях. Так, если в словарях заданы объекты «*федерация»* (как возможная часть юридического лица) и *«Российская федерация*» (как единое административное название), то будет распознан объект «*Российская федерация*».

При задании образцов в файле ***.dct** используются перечисленные ниже обозначения. Интерпретация словарных образцов ключевых слов и имен атрибутов зависит от регистра.

### Ключевые слова

*MSYN* – задает способ сравнения всех лексем в цепочке с учетом морфологии (во всех грамматических формах).

*SYN* – задает способ сравнения всех лексем в цепочке без учета морфологии (в заданной грамматической форме). При этом если первая буква в цепочке – строчная, то все символы в тексте будут распознаваться независимо от регистра, а если первая буква в цепочке – прописная, то первый символ в цепочке в тексте должен быть прописным, а прочие будут распознаваться независимо от регистра.

*CSYN* – эквивалентно *SYN*.

*MAIN* – задает главное слово в цепочке лексем, от которого будут наследоваться грамматические атрибуты, если они не заданы явно. Может стоять перед любым словом в цепочке или отсутствовать. В последнем случае грамматические атрибуты наследуются от первого слова цепочки.

*//* – определяет начало комментария, который следует до конца строки.

### Формат записи

```
<семантическая категория>:<семантическая подкатегория><символ конца
   строки>
(<цепочка текстовых единиц> (MSYN | CSYN | SYN) <синоним> ({<список
   значений грамматических атрибутов>})? <символ конца строки>)*
```

#### Пример 4. Словарь оборотов

```
Idioms:ServiceWords
как бы там ни было,   CSYN {SpeechPart = "Introductory"}
тот или иной    MSYN любой {SpeechPart="Adjective"}
рядом с    CSYN {SpeechPart="Preposition", Case="Instrumental"}
```

### Пример 5. Словарь названий организаций

```
Officials:Name
Московский государственный MAIN университет MSYN
МГУ SYN Московский государственный MAIN университет
МГУ им. М. В. Ломоносова SYN Московский государственный MAIN университет
МГУ имени М. В. Ломоносова SYN Московский государственный MAIN
   университет
Центральный MAIN банк РФ MSYN   ЦБ России
Центральный MAIN банк России MSYN ЦБ России
Центральный MAIN банк Российской Федерации MSYN ЦБ России
```

## Присвоение атрибутов

При выделении объект получает следующие атрибуты:

- `Token.Text` – строка текста, соответствующая распознанной цепочке;
- `Token.Synonym` – строка синонима в случае, когда он задан явно;
- `Token.Type` – значение *Word*;
- `Token.SemanticType` (разбивается на `Token.SemanticCategory`, `Token.SemanticSubcategory`) – задается в первой строке словаря;
- `Атрибуты группы Morph` – значения задаются явно в правой части описания объекта, либо наследуются от заданного слова (*MAIN*), либо наследуются от первого слова в цепочке;
- `Token.OrthographicType`, `Token.HomonimsCount` – наследуются от заданного слова в цепочке (*MAIN*), либо от первого слова.

Полная информация о предопределенных атрибутах и их значениях приведена в Приложении 1.

# Правила выделения объектов (файлы *.rul)

Каждый файл ***.rul** содержит список правил, действующих в пределах одной фазы обработки.

Правила состоят из левой и правой частей, разделенных "-->". Левая часть правила задает образец, в соответствии с которым происходит распознавание объекта. Выражение в правой части задает атрибуты, добавляемые в описание объекта в случае его выделения.

Образец представляет собой логическое выражение на формальном языке, задающее ограничения на атрибуты объектов в цепочке описаний, соответствующей распознаваемому объекту. Результат распознавания определяется проверкой заданных условий на значения атрибутов в цепочке описаний. Для удобства работы образец может быть записан посредством использования ранее определенных макросов.

В левой части правила допускается определения метки для выбранных описаний из цепочки с целью их последующего использования в правой части, например, для наследования грамматических атрибутов. Метки также позволяют включать в выделяемый объект не все объекты цепочки, распознанной в соответствии с образцом, а лишь заданное подмножество для осуществления контекстно-зависимого распознавания.

### Пример 6. Правило для выделения косвенной речи

```
Macro: SPEECH_VERB
(
 ({Token.Text=|"говорить"}) |
 ({Token.Text=|"сказать"}) |
 (
   (({Token.Text=|"отмечать"}) | ({Token.Text=|"отметить"}))
   {Token.Text==","}
   {Token.Text=^"что"}
 )
):speech_verb

Rule: SpeechVerbRule
 SPEECH_VERB
-->
:speech_verb.Token = {SemanticType="Speech:Verb", Rule="SpeechVerb"},
:speech_verb.UserDefined={Something="MyString"}
```

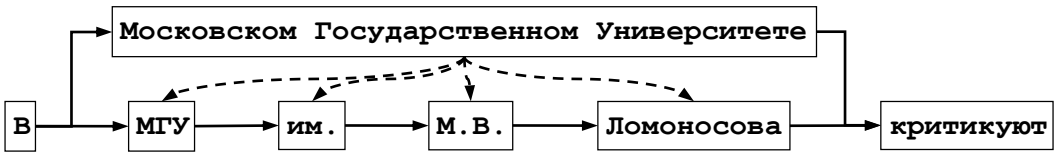
В приведенном выше примере описаны макрос и правило для выделения в тексте некоторых показателей косвенной речи. Макрос **SPEECH_VERB** описывает логическое выражение – образец для распознавания цепочек текста «*говорить*», «*сказать*», «*отмечать, что*» и «*отметить, что*». При этом распознавание всех глаголов производится с учетом морфологии (во всех грамматических формах), распознавание запятой – как есть, а распознавание слова «*что*» – в заданной форме, но без учета регистра, для чего использованы соответствующие операторы сравнения «=|», «==» и «=^». Метка `speech_verb`, указанная после макроса, позволяет обратиться к ней в правой части правила **`SpeechVerbRule`** и объединить все объекты распознанной цепочки в новый выделенный объект.

В правой части правила указывается, какие атрибуты будут присутствовать в описании выделенного объекта, и задаются их значения. Это предопределенный атрибут `Token.SemanticType` и два атрибута, заданные пользователем: `Token.Rule` и `UserDefined.Something`.

Прочие атрибуты, включая предопределенные, не войдут в результирующее описание.

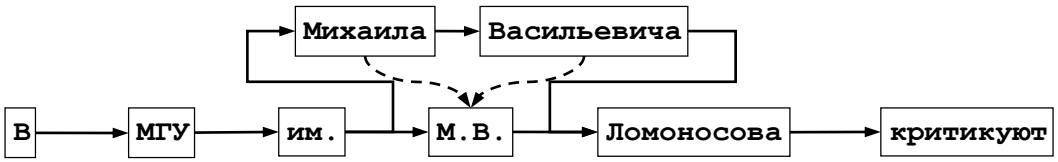
Выражение в левой части правила может содержать описание одного или нескольких объектов. Например, в приведенном правиле для выделения косвенной речи образец соответствует двум цепочкам из одной лексемы и двум цепочкам из трех лексем. Аналогично в правой части правила можно задать не только один объект, но и цепочку распознаваемых объектов. Есть три способа добавления новых объектов в граф описаний:

- Над одним или несколькими объектами, описанными в левой части правила, создается объект, описанный в правой части правила:



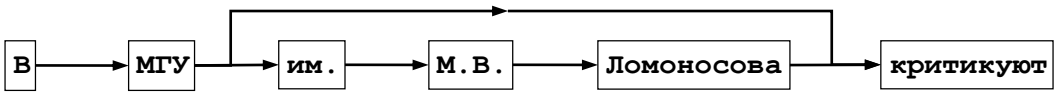

  В этом случае правая часть правила описывается в простом формате, как показано в примере 6 (см. также правило 1 в Приложении 2).

- Над одним или несколькими объектами надстраивается несколько объектов:




  В правой части правил после имени метки в квадратных скобках следует указать порядковый номер объекта в создаваемой цепочке (см. правило 2 в Приложении 2).

- Объекты, описанные в левой части правила, удаляются в новом пути графа:




  В правой части правила после имени метки, объединяющей удаляемые объекты, следует написать в квадратных скобках символ “~” (см. правило 3 в Приложении 2).

## Обращение к атрибутам объектов

Для удобства атрибуты объектов разбиваются на группы с близкой семантикой. Две предопределенные группы `Morph` и `Token` содержат множества атрибутов, описывающих грамматические (такие как `SpeechPart`, `Case`, `Gender`, `Number`) и общие (такие как `Text`, `SemanticType`, `OrthographicType`) характеристики объектов. Полное описание атрибутов приведено в Приложении 1.

Обращение к атрибуту в левой части правила вида «`ИмяГруппы.ИмяАтрибутаВГруппе`» применяется для сравнения значения атрибута с заданным при распознавании, например: {`Token.Length < 5`} или **{**`Morph.SpeechPart == "Adjective"`**}**.

Условие отрицания на значение атрибута объекта задается символом “!”, например: **{**`!Token.OrthographicType == "FirstSmall"`**}** – распознает все лексемы, начинающиеся не с маленькой буквы.

Условие сравнения атрибутов из одного описания может быть задано двумя способами:

- Перечислением атрибутов через запятую внутри одной пары фигурных скобок, например: {`Token.Type == UnknownWord, Token.Length > 1, Token.Length < 6, Token.OrthographicType == "AllCapitals"`} – распознает неизвестную аббревиатуру. В этом случае общий результат сравнения есть результат операции логического «И» над результатами всех сравнений;

- Конкатенированием оператором “|” с обрамлением в круглые скобки, например: ({Token.Text =^ *"им."*}) | ({Token.Text =^ *"имени"*}). В этом случае общий результат сравнения есть результат операции логического «ИЛИ» над результатами всех сравнений. В круглых скобках под оператором “|” может быть задано условие на несколько атрибутов, как указано выше.

Работа с атрибутами группы Morph имеет ряд особенностей, связанных с тем, что атрибуты данной группы являются множественными, т.е. одно описание может иметь несколько значений атрибутов части речи, рода, числа, падежа и т.д. Это связано с наличием омонимии в русском языке – например, словоформа «*туши*» в тексте может соответствовать различным грамматическим формам следующих слов: «*туша*», «*туш*», «*тушь*», «*тушить*». Причем даже для одного слова «*туша*» данная словоформа будет соответствовать трем грамматическим формам: родительного падежа единственного числа, именительного падежа множественного числа и винительного падежа множественного числа. Информацию о наличии омонимичных форм дают два атрибута: Token.HomonimsCount и Morph.InfoCount. Значение первого задает число различных слов, имеющих данную словоформу, а значение второго – общее количество различных грамматических форм, соответствующих данной словоформе.

При сопоставлении атрибута группы Morph все его значения сравниваются с заданным, и распознавание происходит в случае совпадения хотя бы с одним из них. В частности, оба условия {Morph.SpeechPart == *"Verb"*} и {Morph.SpeechPart == *"Noun"*} будут истинны: первое соответствует глаголу «*тушить*», второе – любому из существительных «*туша*», «*туш*», «*тушь*». Если же задано условие на ряд атрибутов группы Morph, то считается, что все совпавшие значения должны соответствовать одной грамматической форме слова. Так, условие {Morph.SpeechPart == *"Verb"*, Number == *"Plural"*} не будет выполнено, и объект не распознается, поскольку результат первого сравнения истинен для слова «*тушить*», а результат второго – для слов «*туша*» и «*туш*».

Условие отрицания значения атрибута “!” в группе Morph имеет противоположную семантику, и распознавание происходит только при отсутствии совпадения заданного значения со всеми значениями атрибута.

Обращение к атрибуту в правой части правила используется для присвоения ему заданного значения, а также для получения его значения с целью присвоения другому атрибуту наследования значения.

В первом случае обращение имеет вид «:ИмяМетки.ИмяГруппы = {ИмяАтрибутаВГруппе = *Значение*}», а во втором – «:ИмяМетки.ИмяГруппы.ИмяАтрибутаВГруппе», где ИмяМетки определяет, к атрибуту какого объекта производится обращение.

Например:

Object1.Token = {SemanticType = *"Speech:Maker"*, Text =*:Object2.Token.Text*}

Как видно, в фигурных скобках может быть перечислено сразу несколько атрибутов одной группы, которым присваиваются или передаются по наследству значения. В примере у объекта Object1 значение атрибута Token.Text наследуется из одноименного атрибута объекта Object2.

Обращение ко множественным атрибутам в правой части правила, например, к атрибутам группы Morph, можно реализовать следующими способами:

1. Чтобы обратиться к конкретному значению атрибута с известным номером, необходимо после имени группы указать его или не указывать ничего для самого первого значения (значения множественных атрибутов нумеруются с 0). Например:

```
{Morph.SpeechPart} или {Morph1.SpeechPart}
```

2. Чтобы обратиться ко всем значениям множественного атрибута необходимо после имени группы поставить знак “?”. Например:

```
{Morph?.SpeechPart}
```

3. Чтобы обратиться к значениям множественного атрибута, удовлетворяющим определенным ограничениям, необходимо после имени группы поставить знак “?”, указав затем в фигурных скобках ограничения. В ограничениях можно обращаться только к атрибутам той же группы с тем же порядковым номером значения: например, к значениям атрибута `BaseForm` тех грамматических форм, для которых выполнено `SpeechPart=="NameF"` и `Case=="Nominative"`, можно так:

```
{Morph?{SpeechPart=="NameF",Case=="Nominative"}.BaseForm}
```

Эти правила применимы и к наследованию атрибутов.

Группа атрибутов допускает групповое наследование – присвоение значений всем своим атрибутам посредством коллективного обращения к ним по имени группы, например: `Object1.Morph` = {*:Object2.Morph*}, `Object1.Token` = {*:Object2.Token*}. Присвоение значений групповым наследованием выполняется раньше, чем наследование значения конкретному атрибуту, а позже всех выполняется присвоение значений атрибутам. Это позволяет реализовать наследование всех атрибутов группы с последующей перезаписью некоторых из них другими значениями, например: `Object1.Token` = {*:Object2.Token,* `Type=`*"Word"*, `Text=`*:Object3.Token.Text*}.

## Пример 7. Правило для выделения «спичмейкеров»

```
Rule: SpeechMakerRule
 (({Morph.IsAnimate=="Animate"}):getmorph |
 ({Token.SemanticType=="Name:FullName"}):getmorph
):speech_maker SPEECH_VERB
-->:speech_maker.Token = { SemanticType = "Speech:Maker",
    Text =: getmorph.Token.Text },
:speech_maker.Morph = {:getmorph.Morph }
```

Приведенное выше правило позволяет выделить всех «производителей» косвенной речи – так называемых «спичмейкеров», обозначенных в тексте одушевленными существительными или именами собственными. При этом предполагается, что имена собственные (могут состоять из нескольких лексем) выделяются другим правилом на предыдущей фазе обработки, и соответствующие объекты получают атрибуты `Token.SemanticType=`*"Name:FullName"*. Для распознавания косвенной речи, которая должна следовать в тексте за упоминанием «спичмейкера», используется макрос **`SPEECH_VERB`**, определенный в Примере 6.

Данный пример иллюстрирует контекстно-зависимое выделение – в качестве объекта выделяется не вся цепочка, распознанная в соответствии с образцом, а лишь ее часть, заданная меткой *speech_maker*.

В выделенном объекте атрибуту `Token.Text` присваивается значение атрибута `Token.Text` объекта, распознанного под меткой *getmorph*, а всем атрибутам группы `Morph` присваиваются значения из описания объекта с меткой *getmorph*.

# Расстановка меток

Как видно из приведенных примеров, метки задаются в логическом выражении в левой части правила (в том числе в макросах) и позволяют выделить требуемую подцепочку объектов внутри всей распознанной цепочки, чтобы обратиться к ней в правой части.

Первичная функция метки – указание цепочки объектов, выделяемой в новый объект при распознавании. По этой метке в правой части правила можно обратиться ко всем атрибутам нового объекта и задать их значения.

Вторичная функция – указание объекта в распознанной цепочке, к атрибутам которого можно обратиться по этой метке в правой части правила и присвоить их значения атрибутам нового выделенного объекта. В этом случае помечаться должен только один объект. Единственное исключение из этого правила – метка может распространяться на цепочку из нескольких объектов, если из них извлекается только значение атрибута `Token.Text`. В этом случае значение атрибута представляет конкатенацию значений `Token.Text` всех объектов, охваченных меткой, через пробел.

В правиле допускается использование нескольких различных меток. Одна и та же метка может выполнять обе функции.

Метки особого вида (завершающиеся символом “#”) могут устанавливаться внутри регулярного выражения, указывая на любую часть цепочки символов, соответствующую единому объекту – раздел «Использование регулярных выражений».

Если в логическом выражении в левой части правила присутствует нескольких альтернативных ветвей, во избежание неоднозначностей, в каждой из них одна и та же метка должна быть проставлена не более одного раза! Метки, выполняющие только первую функцию, могут быть проставлены на цепочку описаний, включая проставление сразу на все ветви, заключенные в скобки. Если под меткой находятся несколько описаний, то использование данной метки для обращения к атрибутам разрешается только для атрибутов `Token.Text`, `Token.Offset`, `Token.Length`. Обращение к атрибутам других типов вызовет ошибку в процессе работы правил. Поэтому следует проявлять известную осторожность при выставлении метки, выполняющей вторую функцию, на часть выражения, содержащую квантификаторы “*” и “+”. Пример 7 иллюстрирует правильную расстановку меток.

Работа с меткой, под которой нет ни одного описания, не приводит к ошибке. Для меток, выполняющих первую функцию, новый объект в правой части правила, задаваемый данной меткой, не создается. Для меток, выполняющих вторую функцию, обращение к атрибутам в правой части правила по этой метке не происходит, поэтому в новом выделенном объекте не появляются атрибуты со значениями, которые надо было получить по этой метке.

Если требуется поставить ограничение на число описаний под меткой, после имени метки следует в квадратных скобках указать тип ограничения (“>”, ”<” или ”=”) и число описаний. Метка не создается, если число описаний под меткой не удовлетворяет заданным ограничениям.

Например:

`(({Token.Text=="."})*):dots[=3]` – выделение троеточий;

`(({Token.Text=~"[0-9]+"})+):numbers[>2]` – выделение нескольких разделенных пробелами чисел;

`(({Token.Text=~"([0-9]+)|([a-z]+)"})*):numbers[<3]` – выделение одной или двух разделенных пробелами букв или чисел.

# Использование квантификаторов “*”, “+” и “?”

Квантификаторы “*”, “+” и “?” представляют собой операторы, которые могут применяться к любой части логического выражения в левой части правила, заключенной в круглые скобки, и позволяют эффективно обрабатывать повторяющиеся конструкции любой длины. Часть выражения, за которой следует символ “?”, распознается ноль или один раз. Часть выражения, за которой следует символ “*”, распознается ноль или более раз. Часть выражения, за которой следует символ “+”, распознается один или более раз.

Например, выражение ({!Token.Text == ""})* позволяет описать повторение любой лексемы в предложении произвольное число раз или вообще ее отсутствие. Следующее выражение позволяет описать образец для распознавания прямой речи, когда после двоеточия в кавычках может следовать любое количество слов, но не менее одного:

```
{Token.Text==":"} {Token.Text=="\""} ({!Token.Text == "\""})+
   {Token.Text== "\""}
```

Семантика сопоставления цепочек описаний с выражением, стоящим под любым из перечисленных операторов, является «жадной», т.е. всегда сопоставляется цепочка максимальной длины, удовлетворяющая условиям выражения (кроме режима *every* – см. раздел «Семантика применения правил»). Например, если в предложении

*... сказал: «Я негативно к этому отношусь», в результате чего ООО «Ромашка» не получила...*
приведенному выше выражению будет сопоставлена цепочка

*: «Я негативно к этому отношусь»*, то выражению

```
{Token.Text==":"} {Token.Text=="\""} ({!Token.Text == "" })+
   {Token.Text== "\""}
```

будет сопоставлена цепочка
*: «Я негативно к этому отношусь», в результате чего ООО «Ромашка»*.

## Пример 8. Правило для выделения интервалов дат

```
Macro: PREP // предлог
(
 ({Morph.SpeechPart=="Preposition"})
)
Macro: REPEAT // лексемы, задающие перечисление
(
 ({Token.Text == ","}) | ({Token.Text =^ "и"}) | ({Token.Text =^
   "или"}) | ({Token.Text == "-"})
)
Rule: DateRule
((PREP)? SINGLE_DATE (REPEAT (PREP)? SINGLE_DATE))*):date_row
-->:date_row.Token = { Type = "Special", SemanticType =
   "Date:DateRow" }
```

Приведенный пример иллюстрирует выделение интервалов дат наподобие «*Дата1, Дата2 или Дата3*», «*Дата1 и с Дата2 по Дата3*», «*Дата1 – Дата2, Дата1 – Дата4*», когда перед каждой датой может стоять предлог, а несколько дат могут следовать друг за другом. При этом предполагается, что образцы отдельных дат описаны макросом **SINGLE_DATE**, который в примере не приводится.

# Использование регулярных выражений

При сравнении строковых значений атрибутов та строка, с которой производится сравнение, может быть задана шаблоном, описанным на языке регулярных выражений (вариант «Perl Compatible»). При сравнении значения атрибута с регулярным выражением следует использовать оператор “=~”.

Регулярное выражение представляет собой одну или более ветвей, разделенных символом “|”. Оно сопоставляется в случае, если сопоставляется одна из его ветвей.

Ветвь состоит из нуля или более частей, следующих друг за другом и сопоставляемых последовательно одна за другой.

За любой частью ветви могут следовать квантификаторы "*", "+" и "?", которые определяют количество допустимых сопоставлений данной ветви (ноль или более раз, один или более раз, ноль или один раз).

Круглыми скобками в ветви выделяется регулярное выражение, диапазон (см. далее), “.” – означает сопоставление с любым символом, “\” с отдельным символом, следующим за ним, или просто отдельный символ – сопоставление с этим символом.

Диапазон задается последовательностью символов в квадратных скобках “[]”. Пара символов, разделенных знаком “-”, означает сравнение с любым символом из заданного диапазона в кодировке Windows-1251. Отдельный символ – сопоставление с ним самим.

В частности, “[0-9]” означает любую цифру, “[А-ЯA-Z]” – любую заглавную букву русского или латинского алфавита, “[АаБбВв]” – любую букву из набора.

Например, для распознавания наименований организаций наподобие *Миноборонпром* или *АВТОПРОМ* и всех их склоняемых форм можно применить следующее сравнение атрибута с использованием регулярного выражения:

```
Token.Text =~ "[А-Я].+[Пп][Рр][Оо][Мм].*"
```

Следующий пример иллюстрирует распознавание всех дат 19-го и 20-го столетий, записанных в формате «*день.месяц.год*». Регулярное выражение содержит несколько ветвей.

```
Token.Text =~ "([1-9]|[0-2][1-9]|3[0-1])\.([1-9]|0[1-9]|1[0-
   2])\.(19[0-9][0-9]|20[0-9][0-9]|[0-9][0-9])"
```

> Сопоставление с регулярным выражением производится только в том случае, если образцу сопоставлена строка целиком!

Метки, завершающиеся символом ‘#’, могут устанавливаться внутри регулярного выражения, указывая на его части, например, для извлечения числа, обозначающего количество минут: {`Token.Text =~ "([1-5]?[0-9]):min#[Мм][Ии][Нн][\.]?"`}. В этом случае текст соответствующей части регулярного выражения становится доступен в правой части правила через обращение вида `ИмяМетки.Token.Text`, например: `time.Cape` = { *Minutes =:min.Token.Text*}.

# Использование фильтров

Чтобы оптимизировать сравнение атрибута `Token.Text` с большим количеством строковых значений, следует использовать так называемые фильтры-словари, включаемые в секции `<cape-filters>` файла **config.xml**. Семантика сравнения строки текста со строками фильтра задается отдельно для каждого фильтра и определяется параметрами в xml-теге `<filter>`. Если ни один параметр не задан, производится точное сравнение. Если задан параметр `backward-search="true"`, тогда строки фильтра сравниваются с окончанием строки текста. Если заданы параметры `min-base-length`, `max-ending-length` или `relevance-ratio`, тогда выполняется «сравнение по основе» или «сравнение по концу» (в зависимости от значения параметра `backward-search`):

- `min-base-length` - задает минимальное допустимое число совпадающих символов от начала строк;
- `max-ending-length` - задает максимальное допустимое число различающихся символов в конце строк;
- `relevance-ratio` - задает минимальную степень соответствия между строкой текста и строкой фильтра (число совпадающих символов, приведенное к длине более длинной строки).

Первая строка словаря определяет имя фильтра, к которому можно обращаться в правилах. Имя фильтра обязано содержать в себе символ ":", например *Date:Month*.

Каждая строка фильтра-словаря содержит одно из возможных строковых значений атрибута `Token.Text`, за которым могут следовать ключевые слова, определяющие его способ сравнения:

- *MSYN* – задает способ сравнения во всех грамматических формах;
- *SYN* – задает точный способ сравнения как есть. Используется по умолчанию.

Пример словаря-фильтра:

```
Date:Month
январь     MSYN
янв.
февраль    MSYN
фев.
```

Для обращения к фильтру в правиле используется оператор "=<", например: `Token.Text` =< *"Date:Month"*.

# Использование функций

При проверке ограничений на атрибуты объектов в левой части правила, а также при присваивании значений атрибутам в правой части правила, можно использовать функции. Эти функции являются внешними относительно формального языка описаний, то есть их имена и поведение полностью определяется приложением, которое вызывает модуль выделения объектов. В Приложении 3 представлен список функций, определенных внутри библиотеки. Кроме того, пользователь может через API библиотеки задать новые функции, а также изменить поведение уже определенных (см. документацию ***Руководство разработчика RCO Fact eXtractor SDK***, функция **`FXCreateContex`**).

Синтаксис вызова функции в левой и правой частях правила одинаковый: имя функции может содержать буквы, цифры, символы "-" и "_", но начинаться должно с буквы или символа "_", после имени функции в скобках через запятую перечисляются ее аргументы. В качестве аргументов можно передавать фиксированные строки или числа, атрибуты объектов, а также результаты вызова других функций:

```
strlen(Token.Text)
substr(Token.Text, 0, 5)
replace( replace(Token.Text, "руб.", "р."), "коп.", "к." )
```

В левой части правила функция записывается слева от оператора сравнения, возвращаемое ею значение сравнивается со значением справа от оператора сравнения:

```
{exists(Token.SentenceStart)==true}
{strlen(Token.Text)>=7, substr(Token.Text,0,6)=^"номер-"}
```

В правой части правила с помощью метки указывается, атрибут какого объекта передается в функцию, а возвращаемое ею значение присваивается атрибуту создаваемого объекта:

```
--> :newObject.Token = {:Object1.Token, Text =
   substr(:Object1.Token.Text, 6, strlen(:Object1.Token.Text)-6)},
    :newObject.Cape = {Date = normalize_date(:Object2.Token.Text)}
```

Алгоритм вызова функций допускает существование разных функций с одинаковым именем. Но в этом случае эти функции должны иметь либо разное число аргументов, либо разные типы аргументов, иначе программа остановится и выдаст ошибку.
Если при вызове функции у объекта отсутствует запрашиваемый атрибут, то соответствующий аргумент функции будет пустой. В коде функции следует предусмотреть обработку такого случая. Результат работы функции может быть пустой, и это не приведет к остановке программы: если функция вызывалась в левой части правила, то в этом случае оператор сравнения всегда выдаст true вне зависимости от его типа. А если функция вызывалась в правой части правила, то не будет никакой модификации атрибута создаваемого объекта. Допустим, в приведенном выше примере метка `:Object2` пустая, тогда в создаваемом объекте `:newObject` будет отсутствовать атрибут типа `Cape.Date`, а если строка атрибута `:Object1.Token.Text` содержит менее 6 символов, то атрибут типа `Token.Text` в новом объекте будет полностью совпадать с атрибутом в объекте `:Object1` (будет выполнено только групповое наследование атрибутов группы `Token`).

## Использование операторов

Аналогично функциям в левой и правой части правил используются операторы – функции с двумя аргументами одинакового типа. При записи оператора вместо имени функции используется специальный символ (“+”, “-”, “*”, “/”), разделяющий аргументы. Допускается последовательное применение операторов, при этом тип оператора не имеет значения, важно лишь, чтобы типы аргументов были одинаковые:

```
{ atoi(symbol(Token.Text,0)) + atoi(symbol(Token.Text,1)) % 11 == 0 }
--> :newobj.Token = {Text = :obj1.Token.Text + " " +
   :obj2.Token.Text}
```

Операторы всегда применяются слева направо без приоритетов. В приведенном примере вначале складываются числа, соответствующие первому и второму символам в строке `Token.Text`, и только затем вычисляется остаток от деления полученной суммы на 11.
Операторы реализованы на базе специальных функций. В приложении 3 приводится полный список возможных операторов, а также соответствующие им функции. В частности, конкатенацию строк можно описать двумя, по сути, эквивалентными способами через операторы и функции:

```
1) :obj1.Token.Text + " " + :obj2.Token.Text + " " + :obj3.Token.Text
2) cape_operator+(:obj1.Token.Text, cape_operator+(" ",
   cape_operator+(:obj2.Token.Text, cape_operator+(" ",
   :obj3.Token.Text))))
```

Цель введения операторов – компактная и наглядная запись наиболее востребованных рекуррентных функций. Пример 9 иллюстрирует возможности операторов и функций при составлении сложных условий на идентификацию номера ИНН.
Как и для функций, поведение каждого оператора можно переопределить. Для этого следует переопределить соответствующую ему функцию через API библиотеки.

### Пример 9. Правило для выделения номеров ИНН организаций

```
Rule: INN_Rule
(
  { Token.Length == 10, Token.Text =~ "[0-9]+",
    2*atoi(symbol(Token.Text,0))
     + (4*atoi(symbol(Token.Text,1)))
     + (10*atoi(symbol(Token.Text,2)))
     + (3*atoi(symbol(Token.Text,3)))
     + (5*atoi(symbol(Token.Text,4)))
     + (9*atoi(symbol(Token.Text,5)))
     + (4*atoi(symbol(Token.Text,6)))
     + (6*atoi(symbol(Token.Text,7)))
     + (8*atoi(symbol(Token.Text,8)))
     % 11 % 10
     - atoi(symbol(Token.Text,9))
     == 0
  }
):inn
--> :inn.Token = { :inn.Token, SemanticType = "Special:INN"}
```

# Формат описания правил

При описании правил в файле ***.rul** используются перечисленные ниже обозначения. При интерпретации правил написания всех ключевых слов, имен атрибутов, названий правил и макросов учитывается регистр буквы.

### Стили комментариев

```
// комментарий
/* комментарий */
;; комментарий
|# комментарий #|
```

### Ключевые слова

*Rule* – задает правило построения описания объекта.

*Macro* – задает макроподстановку для последующего использования в правилах описания объекта.

*Priority* – задает приоритет исполнения правила.

### Операторы

--> – начало правой части правила.

**|** – оператор «или» используется для описания альтернатив в левой части правила.

**,** – оператор «и» используется для задания ограничений на несколько атрибутов описания одновременно.

**=** – присвоение значения атрибуту в правой части правила.

**==** – сравнение значения атрибута как есть. Определено для строк (пишутся в кавычках), целых чисел, чисел с плавающей точкой и двоичных констант *true* и *false* без кавычек.

**=~** – сравнение значения атрибута с регулярным выражением. Определено для строк.

**=|** – сравнение значения атрибута со строкой с учетом морфологии (во всех грамматических формах). Определено для строк.

**=^** – сравнение значения атрибута со строкой без учета регистра. Определено для строк.

**=<** – поиск значения атрибута `Token.Text` в словаре-фильтре с указанным именем. Определено для атрибута `Token.Text`.

**<=** – сравнение значения атрибута по признаку «меньше или равно». Определено для чисел.

**>=** – сравнение значения атрибута по признаку «больше или равно». Определено для чисел.

***** – используется для описания возможности повторения выражений ноль и более число раз в левой части правила. Выражение должно быть взято в круглые скобки.

**+** – используется для описания возможности повторения выражения один и более раз в левой части правила. Выражение должно быть в круглых скобках.

**?** – используется для описания возможности повторения выражения ноль или один раз в левой части правила. Выражение должно быть взято в круглые скобки. В правой части правила используется в конце имени множественного атрибута вместо цифры (см. формат **`Rcondition`**).

**!** – отрицание заданного условия на значение атрибута.

## Формат записи правил (в БНФ)

```
(
 (Macro: <ident>
 (PatternElement | Action))*
 |
 (Rule: <ident> (Priority: <integer>)?
 LeftHandSide --> RightHandSide)+
)+
```

Где:

```
LeftHandSide::=ConstraintGroup
ConstraintGroup::=(PatternElement)+(<bar>(PatternElement)+)*
PatternElement::=(<ident>|BasicPatternElement|ComplexPatternElement)
BasicPatternElement::=((<leftBrace>Constraint(<comma>Constraint)*<rig
   htBrace>)
ComplexPatternElement::=<leftBracket>ConstraintGroup<rightBracket>(<k
   leeneOp>)? (<colon>(<ident>|<integer>))?
Constraint::=(<pling>)?((<ident><period><ident>)|Function|Operator)<e
   quals>AttrVal
Function::=<ident><leftBracket>(FArg(<comma>FArg)*)?<rightBracket>
Operator::=FArg<functionOp>FArg
AttrVal::=(<string>|<ident>|<integer>|<floatingPoint>|<bool>)
FArg::=(<ident><period><ident>)|AttrVal|Function|Operator
RightHandSide::=Action(<comma>Action)*
Action:=AssigmentExpression|<ident>
AssigmentExpression::=(<colon><ident><period><ident><assign><leftBrac
   e>
((<ident>(Rcondition)?<assign>(AttrVal|Function|Operator|
  <colon><ident>(Rcondition)?<period><ident><period><ident>))
   (<comma>)?) | (<colon><ident><period><ident>)
)* <rightBrace>
Rcondition::=<query>(<leftBrace>RConstraint(<comma>RConstraint)*<righ
   tBrace>)?
RConstraint::=(<pling>)?<ident><equals>AttrVal

digit::=[0-9]
digits::={digit}+
letter::=[A-Za-zA-Яа-я]
letters::={letter}+
```

```
letterordigitordash::={letter}|{digit}|"-"|"_"
ident::=("_"|{letter}){letterordigitordash}*
comma::= ","
period::= "."
colon::= ":"
assign::= "="
leftBrace::= "{"
rightBrace::= "}"
leftBracket::= "("
rightBracket::= ")"
integer::= [0-9]+
exponent::= [eE][+-]?{digits}
floatingPoint::={digits}"."{digit}*{exponent}?[fFdD]?|"."{digits}{exp
   onent}?[fFdD]?|{digits}{exponent}[fFdD]?|{digits}{exponent}?[fFdD]
bool::= ([Tt][Rr][Uu][Ee])|([Ff][Aa][Ll][Ss][Ee])
string::= \"([^"]|\\["])*\"
kleeneOp::= "*"|"+"|"?"
bar::=     "|"
equals::= "=="|"=~"|"=^"|"=<"|"=|"|">="|"<="
query::= "?"
functionOp::= "+"|"*"|"/"|"%"
```

# Приложение 1. Список предопределенных атрибутов

Ниже приведен список предопределенных атрибутов, которые могут существовать в соответствующих предопределенных группах (`Token`, `Morph`, `Cape`). У элементарных объектов – лексем – значения атрибутов `Token` и `Morph` проставляются автоматически модулем лексикографического и морфологического анализа текста.

**Внимание!** При самостоятельной настройке описаний объектов в составе продуктов **RCO**, использующих синтаксический анализатор, для корректной обработки выделенных объектов желательно присвоение значений всем атрибутам группы `Morph`, которые присутствуют у той части речи, к которой относится объект. При этом часть речи должна задаваться атрибутом `SpeechPartDetailed` (значение атрибута `SpeechPart` игнорируется). Обычно задание значений всех атрибутов реализуется неявно через их наследование от заданного главного слова в объекте.

## Атрибуты группы Token

`Text` – строка текстовой единицы, соответствующей объекту.

`SemanticCategory` – семантическая категория текстовой единицы, исходное значение – *undefined*.

`SemanticSubcategory` – семантическая подкатегория текстовой единицы, исходное значение – *undefined*.

`SemanticType` – семантический тип текстовой единицы. Является конкатенацией атрибутов `SemanticCategory` и `SemanticSubcategory` через символ ":". Значение по умолчанию *undefined*:*undefined*. Установка значения данного атрибута автоматически приводит к установке соответствующих значений атрибутов `SemanticCategory` и `SemanticSubcategory`, и наоборот. То есть изменения полей `SemanticType`, `SemanticCategory` и `SemanticSubcategory` автоматически синхронизируются.

`Length` – число символов в текстовой единице.

`Offset` – смещение начала текстовой единицы от начала текста.

`Index` – порядковый номер текстовой единицы в предложении (начиная с 0).

`Synonym` – строка синонима, если он задан в словарном модуле.

`SentenceStart` – значение true, если объект стоит в начале предложения.

`SentenceEnd` – значение true, если объект стоит в конце предложения.

`OrthographicType` – орфографический тип текстовой единицы:

*Break* – абстрактный разделитель

*Quote* – кавычка

*QuoteOpen* – открывающая кавычка

*QuoteClose* – закрывающая кавычка

*Colon* – двоеточие

*Semicolon* – точка с запятой

*Dash* – тире

*BracketOpen* – открывающая скобка

*BracketClose* – закрывающая скобка

*HyphenStrong* – дефис

*Comma* – запятая

*AllCapitals* – слово из прописных букв

*FirstCapital* – первая буква слова прописная

*FirstSmall* – первая буква слова строчная

*Number* – число

*Unknown* – тип неизвестен

`Type` – общий тип текстовой единицы:

*Word* – слово русского языка, присутствующее в словаре морфологического анализа

*UnknownWord* – слово русского языка, отсутствующее в словаре морфологического анализа

*LatinicWord* – слово, записанное латинскими буквами

*Special* – специальная конструкция – все прочее

*Delimiter* – разделитель – знак препинания. Тип разделителя – значение атрибута `OrthographicType`

`HomonymsCount` – число омонимов – различных слов, имеющих грамматические формы, совпадающие со словоформой лексемы в тексте.

# Атрибуты группы Morph

`Infocount` – количество различных грамматических описаний. Может быть больше единицы в случае омонимии.

`WordBase` – неизменяемая часть слова – строка.

`BaseForm` – слово в нормальной форме.

`Case` – грамматический падеж:

*Nominative* – Именительный

*Generative* – Родительный

*Dative* – Дательный

*Accusative* – Винительный

*Instrumental* – Творительный

*Prepositional* – Предложный

`Gender` – грамматический род:

*Masculine* – Мужской

*Feminine* – Женский

*Neuter* – Средний

*Common* – Общий (мужской и женский)

`Number` – грамматическое число:

*Plural* – Множественное

*Singular* – Единственное

`IsAnimate` – признак одушевленности:

*Animate* – Одушевленность

*Inanimate* – Неодушевленность

`Person` – грамматическое лицо:

*First* – Первое лицо

*Second* – Второе лицо

*Third* – Третье лицо

`Participle` – тип причастия:

*Active* – Активный залог

*Passive* – Пассивный залог

*Gerund* – Деепричастие

*Unknown* – Не определен

`IsBriefForm` – если прилагательное или причастие стоит в краткой форме, значение равно *true*.

`IsCompareForm` – если прилагательное или причастие стоит в сравнительной форме, значение равно *true*.

`IsInfinitiveForm` – если глагол стоит в форме инфинитива, значение равно *true*.

`IsBackForm` – если глагол или причастие стоит в возвратной форме, значение равно *true*.

`Paradigm` – тип парадигмы словоизменения:

| | |
|---|---|
| *Subject* | – изменяется по типу существительного |
| *Definition* | – изменяется по типу прилагательного |
| *Verb* | – изменяется по типу глагола |

`SpeechPart` – грамматическая часть речи:

| | |
|---|---|
| *Unknown* | – неизвестна. |
| *Introductory* | – вводное слово |
| *Interjection* | – междометье |
| *Predicative* | – предикатив |
| *Preposition* | – предлог |
| *Conjunction* | – союз |
| *Particle* | – частица |
| *Adverb* | – наречие |
| *Comparative* | – обособленная сравнительная степень |
| *Pronoun* | – местоимение |
| *Numeric* | – числительное |
| *Abbr* | – аббревиатура |
| *Name* | – имя собственное |
| *GeoName* | – географическое название |
| *Verb* | – глагол |
| *Adjective* | – прилагательное |
| *Noun* | – существительное |
| *Appelation* | – наименование |

`SpeechPartDetailed` – детализированная часть речи:

*Unknown* – неизвестна

*Introductory* – вводное слово

*Interjection* – междометие

*Predicative* – предикатив

*Preposition1* – предлог, управляющий родительным падежом

*Preposition2* – предлог, управляющий дательным падежом

*Preposition3* – предлог, управляющий винительным падежом

*Preposition4* – предлог, управляющий творительным падежом

*Preposition5* – предлог, управляющий предложным падежом

*ConjunctionCoordinative* – союз сочинительный

*Conjunction* – союз несочинительный

*Particle* – частица

*Adverb* – наречие

*AbbrNoun* – сокращенное существительное

*AbbrAdjective* – сокращенное прилагательное

*AbbrIntroductory* – сокращенное вводное слово

*Comparative* – обособленная сравнительная степень

*PronounPossessive* – притяжательное местоимение

*Pronoun* – местоимение

*PronounAdjective* – местоименное прилагательное

*Numeric* – числительное

*NumericCollective* – собирательное числительное

*NumericOrdinal* – порядковое числительное

*AbbrM* – аббревиатура мужского рода

*AbbrF* – аббревиатура женского рода

*AbbrN* – аббревиатура среднего рода

*NameM* – имя мужского рода

*NameF* – имя среднего рода

*MiddleNameM* – отчество мужского рода

*MiddleNameF* – отчество женского рода

*LastName* – фамилия

*GeoNameM* – географическое название мужского рода

*GeoNameF* – географическое название женского рода

*GeoNameN* – географическое название среднего рода

*GeoNamePlural* – географическое название множественного числа

*VerbImperfect* – переходный глагол несовершенного вида

*VerbIntransitiveImperfect* – непереходный глагол несовершенного вида

*VerbPerfect* – переходный глагол совершенного вида

*VerbIntransitivePerfect* – непереходный глагол совершенного вида

*VerbTwoway* – переходный двувидовой глагол

*VerbIntransitiveTwoway* – непереходный двувидовой глагол

*Adjective* – прилагательное

*NounM* – неодуш. существительное мужского рода

*NounAnimateM* – одуш. существительное мужского рода

*NounAnimateInanimateM* – одуш.-неодуш. существительное мужского рода

*NounF* – неодуш. существительное женского рода

*NounAnimateF* – неодуш. существительное женского рода

*NounAnimateInanimateF* – одуш.-неодуш. существительное женского рода

*NounN* – неодуш. существительное среднего рода

*NounAnimateN* – одуш. существительное среднего рода

*NounAnimateInanimateN* – одуш.-неодуш. существительное среднего рода

*NounPlural* – неодуш. существительное множественного числа

*AppelationM* – наименование мужского рода

*AppelationF* – наименование женского рода

*AppelationN* – наименование среднего рода

*AppelationPlural* – наименование множественного числа

# Атрибуты группы Cape

Атрибуты этой группы с любыми именами могут быть присвоены объекту в правой части правила. В дальнейшем эти атрибуты и их значения могут быть получены через программный интерфейс.

# Приложение 2.
# Примеры правил выделения объектов

## Правило 1. Сворачивание цепочки объектов в один объект

```
Rule: Rule1
(
 {Token.Text=="МГУ"} {Token.Text=="им."} {Token.Text=^"М.В."}
 {Token.Text=^"Ломоносова"}
):univer
-->:univer.Token = {SemanticType = "Organization:Name", Text =
   "Московский Государственный Университет"}
```

## Правило 2. Надстраивание над объектом цепочки объектов

```
Rule: Rule2
(
 ({Token.Text=^"М.В."}):initials {Token.Text=|"Ломоносов"}
)
-->:initials[1].Token = {Text = "Михаил"},
   :initials[2].Token = {Text = "Васильевич"}
```

## Правило 3. Удаление цепочки объектов

```
Rule: Rule3
(
 {Token.Text=="МГУ"}
({Token.Text=="им."}{Token.Text=^"М.В."}{Token.Text=^"Ломоносова"}):d
   el
)
-->:del[~].Token = { :del.Token }
```

# Приложение 3. Встроенные функции и операторы

Ниже (Таблица 1) приведено описание функций, определенных внутри библиотеки. В первом столбце перечислены имена функций, во втором – указаны типы возвращаемых функцией значений и ее аргументы. Тип `int` соответствует целому числу, `double` – вещественному, `string` – строке текстовых символов, а тип `bool` может принимать значения *true* или *false*. Результат работы всех функций, кроме `exists`, будет пустой, если хотя бы один из аргументов пустой (в объекте нет соответствующего атрибута).

Таблица 1. Описание функций, определенных внутри библиотеки.

| **Имя** | **Типы данных** | | **Описание** |
|---|---|---|---|
| | результат | аргументы | |
| **atoi** | `int` | `string` | Переводит в число строку, состоящую из числовых символов (0-9, +, -). Наличие других символов дает пустой результат функции. |
| **concatenate** | `char` | `char` | Соединяет значения атрибутов `Token.Text`. |
| **concatenate** | `char` | `char, char` | Соединяет значения атрибутов `Token.Text`, перемежая их заданным разделителем. |
| **concatenate_attr** | `char` | `char, char, char` | Соединяет значения указанных атрибутов, перемежая их заданным разделителем. |
| **exists** | `bool` | `int` | Проверяет наличие атрибута: возвращает *true*, если аргумент не пустой, иначе – *false*. Так, функция **exists**`(Token.SentenceStart)` вернет *true*, если объект содержит атрибут `Token.SentenceStart`. |
| | | `double` | |
| | | `string` | |
| | | `bool` | |
| **itoa** | `string` | `int` | Переводит число в строку, состоящую из числовых символов (0-9, +, -). Наличие других символов дает пустой результат функции. |
| **negate** | `int` | `int` | Обращает знак числа, переданного функции в качестве аргумента. |
| | `double` | `double` | |
| **NormalForm** | `char` | `char` | Приводит слова описания объекта к нормальной форме. |
| **NormalFormCase** | `Char` | `char` | Приводит слова описания объекта к заданной форме (например, к родительному падежу). |
| **normalize_date** | `string` | `string` | Приводит строку с датой к виду: гггг-мм-дд, где гггг – номер года, мм – месяц, дд – день (может отсутствовать год или день). Умеет разбирать даты в русском и англоязычном формате. |
| **replace** | `String` | `string, string, string` | Принимает на вход строку текста (аргумент 1) и две строки (аргументы 2 и 3), задающие правило замены строк: первой подстроки на вторую. Результат пустой, если хотя бы один из аргументов отсутствует. |
| **symbol** | `string` | `string, int` | Возвращает строку, содержащую один символ из строки, передаваемой в качестве первого аргумента. Номер символа задает второй аргумент (начиная с 0). Результат будет пустым, если номер символа превосходит длину строки. |

| | | | |
|---|---|---|---|
| **substr** | string | string, int, int | Выделяет подстроку из строки – аргумент 1. Второй аргумент задает смещение в символах от начала строки, третий – длину. Результат будет пустым, если длина требуемой подстроки превосходит длину исходной строки. |
| **strlen** | int | string | Функция вычисляет длину строки. |
| **translate** | string | string, string, string | Принимает на вход строку текста (аргумент 1), и две строки символов (аргументы 2 и 3). Производит замену символов строки текста со значениями из первой строки символов, на значения, из второй строки символов. Результат будет пустым при неравенстве длин исходной и результирующей строк символов. |

Таблица 2 содержит описание операторов. Синтаксис оператора: *результат = аргумент1 символ аргумент2*. В первом столбце перечислены используемые символы операторов, во втором – имена функций, соответствующих операторам, в третьем столбце указан тип аргументов (тип результата оператора и типы обоих его аргументов совпадают).

Таблица 2. Описание операторов.

| Символ | Имя функции | Тип | Описание |
|---|---|---|---|
| + | **operator+** | string | Конкатенация двух строк. |
| | | int | Сумма двух целых чисел. |
| * | **operator*** | int | Умножение двух целых чисел. |
| / | **operator/** | int | Целая часть от деления первого числа на второе. |
| % | **operator%** | int | Остаток от деления первого числа на второе. |