Изменения 4.12

Функция с параметрами ToString(bool, …) у Referent и его производных класов переименована в ToStringEx, функция ToString() без параметров осталась без изменений.

Для Javascript раньше конвертер кода ошибочно генерировал имена методов в стиле Python с подчёркиванием в качестве разделителя, в версии 4.12 это исправлено (например, теперь вместо process\_on\_server будет processOnServer).

Изменения 4.8

До версии 4.8 для C# при сравнении экземпляров MorphClass и MorphCase использовалось переопределение оператора ==. Сейчас это заменено на функцию Equals, а == теперь обычное сравнение экземпляров. В принципе, прямое сравнение редко встречается (судя по моему коду) - в основном используются функции типа .IsNone, .IsNominative и пр. Но тем не менее...

Переход на версию 4.0

Основные изменения касаются версии C#: теперь нет больше dll-сборок, а SDK распространяется в виде исходных кодов, как и для других языков программирования. В связи с этим namespace “EP” теперь “Pullenti”, а структура файлов и классов идентична структуре на других языках.

Сгенерирована документация в виде набора html-страниц в стиле MSDN. В связи с этим пришлось сделать небольшие переименования, касающиеся всех языков:

|  |  |
| --- | --- |
| *Было* | *Стало* |
| Namespace EP (для C#) | Namespace Pullenti |
| Класс Pullenti.Ner.Sdk | Pullenti.Sdk |
| Внутренний класс Referent.EqualTyp | ReferentsEqualTyp |
| TerminCollection.TryAttachStr() | TerminCollection.FindTerminsByString() |
| Morphology | MorphologyService |
| Explanatory | DerivateService |
| TextToken.GetLemma() | TextToken.Lemma |
| CanBeEqualAttrs | CanBeEqualAttr |

Полностью удалены из SDK анализаторы BusinessAnalyzer и SentimentAnalyzer в связи с невостребованностью и недоделанностью.

Также убрано Repository – то, что планировалось как надстройка над СУБД для хранения сущностей. Если вдруг кто использовал – дайте знать (вернём).

Изменения версии 3.23

В функциях GetNormalCaseText у Token и производных от неё второй параметр (необходимость приведения к единственному числу) с bool заменено на элемент MorphNumber, чтобы можно было приводить ко множественному числу при необходимости.

Класс DateExToken для работы с относительными датами упразднён (сделан приватным), теперь весь функционал находится в основных классах DateReferent и DateRangeReferent. В них добавлен флажок bool IsRelative, а также добавлены функции, которые вычисляют реальную дату или диапазон относительно заданного момента времени (например, текущего момента). Для DateReferent:

/// <summary>

/// Вычислить дату-время (одну)

/// </summary>

/// <param name="now">текущая дата (для относительных дат)</param>

/// <param name="tense">время (-1 - прошлое, 0 - любое, 1 - будущее) - испрользуется

/// при неоднозначных случаях</param>

/// <returns>дата-время или null</returns>

public DateTime? CalculateDate(DateTime now, int tense = 0)

/// <summary>

/// Вычислить диапазон дат (если не диапазон, то from = to)

/// </summary>

/// <param name="now">текущая дата-время</param>

/// <param name="from">результирующее начало диапазона</param>

/// <param name="to">результирующий конец диапазона</param>

/// <param name="tense">время (-1 - прошлое, 0 - любое, 1 - будущее) - испрользуется

/// при неоднозначных случаях

/// Например, 7 сентября, а сейчас лето, то какой это год? При true - этот, при false - предыдущий</param>

/// </param>

/// <returns>признак корректности</returns>

public bool CalculateDateRange(DateTime now, out DateTime from, out DateTime to, int tense = 0)

У DateRangeReferent фукнция CalculateDateRange с таким же определением.

Изменения версии 3.15

В связи с поддержкой PHP и javascript пришлось ещё кое-что подправить.

|  |  |  |
| --- | --- | --- |
| *Класс* | *Метод* | *Новое имя* |
| Referent | GetValue(string) | GetSlotValue |
| NumberHelper | TryParseFloatNumber | TryParseRealNumber и возвращает NumberToken вместо NumberExToken |
| NumberExToken | TryAttachPostfixOnly | TryParsePostfixOnly в NumberHelper |
| NumberExToken | TryParseNumberWithPostfix | TryParseNumberWithPostfix в NumberHelper |

Класс NumberToken существенно переделан: вместо long Value теперь string Value { get; set; }, причём число может быть очень большим, а также быть с дробной частью (разделитель – всегда точка). Это сделано, чтобы уйти от ограничений на разрядность (например, в Javascript возможны только 32-х разрядное целое, а далее оно становится double с возможной потерей точности). Добавилось int? IntValue { get; set; }, возвращающий null, если число не умещается в 32 разряда и не целочисленное. Добавился double RealValue { get; set; }, если строку Value не удаётся преобразовать в double, то вернёт NaN. Планируется реализовать все арифметические операции в рамках NumberToken для любых размеров Value (чуть позже).

Соответственно из NumberExToken убрано свойство RealValue – оно переместилось в NumberToken.

Изменения версии 3.7

В версию 3.6 пришлось ввести некоторые переименования. Это связано с поддержкой реализации на Python, который не позволяет иметь одинаковые имена для разных методов класса (сигнатура не содержит типов параметров), а также запрещает идентификаторы на кириллице.

|  |  |  |
| --- | --- | --- |
| *Класс* | *Метод* | *Новое имя* |
| Token | IsChar(string) | IsCharOf |
| MiscHelper | GetTextValue(MetaToken, …) | GetTextValueOfMetaToken |
| ExtOntology | Add(…, Referent) | AddReferent |
| Processor | DeserializeReferent(XmlNode…) | DeserializeReferentFromXml |
| Processor | Process(AnalysysResult) | ProcessNext |
|  |  |  |
|  |  |  |

Все структуры переделаны в классы (MorphClass, MorphCase, MorphLang, ExplanWordAttr), соответственно у виртуальной функции GetNormalCaseText из Token и его производных классов первый параметр из MorphClass? mc = null переделан в MorphClass mc = null.

Перечисление NumberToken.SpellingType вынесено на верхний уровень и стало NumberSpellingType, перечисление NumberExToken.ExTyps стало NumberExType. У перечисления BlkTypes все члены переведены в латиницу.

Класс из Explanatory: ExplanWord полностью переделан, теперь он объединён вместе с полями DerivatWord и используется вместо него в деривативных группах DerivatGroup.

**Версия для .NET Core переведена в версию 2.0.**

Pullenti NER: переход на версию 3.0

C версии 3.0 Pullenti NER имеет реализацию на Java, для чего написан специальный конвертор кода C# => Java. В связи с этим пришлось внести в базовую часть ряд изменений, которые описываются ниже. В основном они касаются введения пространств имён и отказом от плагинной инициализации анализаторов.

# Пространства имён

Во 2-й версии все классы располагались в пространствах (namespace) EP (базовая часть, специфические сущности), EP.Semantix (токены и пр. лингвистические возможности) и EP.Text (морфология).

В 3-й версии пространства такие:

* EP.Morph – морфология
* EP.Ner – базовые классы движка (ProcessorService, Processor, Referent, SourceOfAnalisys, Token, Slot, Analyzer …)
* EP.Ner.Core – разные лингвистические обработчики, хелперы и пр. (NounGroupHelper, BracketHelper, NumberHelper, TerminCollection …)

Остальные подмножества EP.Ner относятся к конкретным анализаторам. Обычно публичная часть содержит выделяемые сущности (наследники от Referent) и выделяющий их анализатор (наследник от Analyzer)

* EP.Ner.Address – работа с адресами (AddressReferent, StreetReferent, AddressAnalyzer);
* EP.Ner.Bank – банковские реквизиты;
* EP.Ner.Booklink – ссылки на внешнюю литературу;
* EP.Ner.Business – выделение бизнес-объектов (*в стадии разработки, не востребовано*);
* EP.Ner.Date – даты и диапазоны;
* EP.Ner.Decree – ссылки на нормативно-правовые акты;
* EP.Ner.Definition – выделение определений и тезисов;
* EP.Ner.Denomination – непонятные буквенно-числовые комбинации типа «А-120», «1С»;
* EP.Ner.Geo – географические объекты (страны, районы, населённые пункты);
* EP.Ner.Instrument – определение структуры нормативно-правовых актов;
* EP.Ner.Mail – обработка писем (email);
* EP.Ner.Money – денежные суммы;
* EP.Ner.Named – разные мелкие именованные сущности типа морей, озёр, памятников, планет и т.п., которые имеют имена и тип (*в стадии разработки, не востребовано*);
* EP.Ner.Org – организации и подразделения;
* EP.Ner.Person – персоны и их атрибуты (должности, профессии, звания, регалии и пр.);
* EP.Ner.Phone – телефонные номера;
* EP.Ner.Semantic – семантический анализ (*в стадии разработки, не востребовано*);
* EP.Ner.Sentiment – эмоциональный анализ (*в зачаточном состоянии, не востребовано*);
* EP.Ner.Titlepage – анализ титульной информации (название, авторы …);
* EP.Ner.Transport – транспортные средства;
* EP.Ner.Uri – всё, что относится к модели СХЕМА:ЛОКАЦИЯ. Например, гиперссылки, email, скайп, ББК и УДК, ИНН, ОКАТО …

Всё, что находится в пространствах имён, заканчивающихся «.Internal», предназначено исключительно для внутреннего использования.

# Инициализация

В старой версии была возможность плагинной инициализации, когда движок сам определял в рабочей директории DLL-сборки, содержащие анализаторы, и динамически подгружал их. Данные способ не всегда работал (например, для Web-приложений), и второй способ предусматривал явную инициализацию сборки путём включения в проект ссылки на неё и явной инициализацией вызовом функции. Сборка могла содержать множество анализаторов, но функция инициализации сама их все инициализировала.

В новой версии инициализация каждого анализатора происходит явно путём вызова его статического метода Initialize(). Если анализатор не инициализирован, то движок его вообще «не видит». Но сначала нужно вызвать функцию Initialize у ProcessorService, указав ему языки, которые будут загружены в память.

EP.Ner.ProcessorService.Initialize(MorphLang.RU | MorphLang.EN);

EP.Ner.Money.MoneyAnalyzer.Initialize();

EP.Ner.Uri.UriAnalyzer.Initialize();

EP.Ner.Phone.PhoneAnalyzer.Initialize();

EP.Ner.Date.DateAnalyzer.Initialize();

EP.Ner.Bank.BankAnalyzer.Initialize();

EP.Ner.Geo.GeoAnalyzer.Initialize();

EP.Ner.Address.AddressAnalyzer.Initialize();

EP.Ner.Org.OrganizationAnalyzer.Initialize();

EP.Ner.Person.PersonAnalyzer.Initialize();

EP.Ner.Mail.MailAnalyzer.Initialize();

EP.Ner.Transport.TransportAnalyzer.Initialize();

EP.Ner.Decree.DecreeAnalyzer.Initialize();

EP.Ner.Titlepage.TitlePageAnalyzer.Initialize();

EP.Ner.Booklink.BookLinkAnalyzer.Initialize();

EP.Ner.Named.NamedEntityAnalyzer.Initialize();

Порядок вызова у анализаторов не важен, так как движок сам располагает анализаторы в нужной последовательности, учитывая их зависимости от результатов работы других анализаторов.

# Создание процессора

Раньше процессор создавался через new Processor(…), теперь экземпляр создаётся путём вызова соответствующего статического метода ProcessorService, причём если сервис не инициализирован, то будет генерироваться исключение.

/// <summary>

/// Создать процессор со стандартным списком анализаторов (у которых свойство

IsSpecific = false)

/// </summary>

/// <returns>экземпляр процессора</returns>

static public Processor CreateProcessor()

/// <summary>

/// Создать экземпляр процессора с пустым списком анализаторов

/// </summary>

/// <returns></returns>

static public Processor CreateEmptyProcessor()

/// <summary>

/// Создать процессор с набором стандартных и указанных параметром специфических

/// анализаторов.

/// </summary>

/// <param name="specAnalyzerNames">можно несколько,

разделённые запятой или точкой с запятой.

/// Если список пустой, то эквивалентно CreateProcessor()</param>

/// <returns></returns>

static public Processor CreateSpecificProcessor(string specAnalyzerNames)

# Разные мелочи

Сборка EP.Analyzer.Date удалена, весь функционал перенесён в сборку EP.Analyzer.Misc.

Сборки EP.Analyzer.Transport и EP.Analyzer.Business удалены, их функционал перенесён в новую сборку EP.Analyzer.Util.

Анализатор DenominationAnalyzer стал специализированным (IsSpecific = true), то есть он по умолчанию не попадает в список анализаторов при создании обычного процессора.

Статическое поле Version у ProcessorService стало типа string вместо Version.

Всё остальное осталось без изменений.