Соглашения о кодировании Ars Magna

версия 1.0 альфа

12 марта 2014 г.

Настоящий документ регламентирует соглашения о кодировании проекта Ars Magna для исходных текстов на языке C#. Приведенные в нем рекомендации призваны помочь разработчикам решить проблему выбора между возможными вариантами кодирования, унифицировать исходные тексты и тем самым облегчить их восприятие. Мы надеемся, что код, соответствующий Соглашению, будет более надежным и переносимым.

# Обозначения, принятые в документе

В некоторых примерах кода с целью лучшей иллюстрации значащие пробелы заменяются на символ ∙ (точка посреди строки).

# Физическое размещение исходных файлов

## Один тип — один файл

Каждый тип (класс, структура, перечисление, интерфейс) помещается в отдельный файл. Имя файла должно совпадать с именем типа. Таким образом, для имен файлов исходных текстов применяется те же правила именования, что и для классов (см. пункт ).

Исключения:

* вложенные типы;
* partial-классы, в которых, например, физически разделены части, написанные вручную и сгенерированные инструментально.

Для файлов исходных текстов применяется расширение .cs (в нижнем регистре).

## Дерево каталогов

Группа связанных по смыслу файлов помещается в отдельный подкаталог.

Для каждого уровня пространства имен создается подкаталог. Например, для AM.Marc.Schema создаются подкаталог AM/Marc/Schema.

Но не каждый подкаталог означает введение пространства имён. Например, в отдельный каталог могут быть вынесены классы отчётов или близкие по смыслу бизнес-сущности.

# Общее оформление исходных текстов

## Интернационализация

Исходные тексты хранятся в кодировке UTF-8 (кодовая страниц 65001). Обязательно наличие сигнатуры Unicode (EF BB BF). В качестве перевода строки используется пара символов CR/LF (0x000D/0x000A).

## Краткое описание

Файл исходного текста должен начинаться с комментария следующего вида:

/\* FileName.cs -- Very short description   
 \*/

Краткое описание должно быть на английском языке и занимать не более одной строки. Если необходимо, пояснения могут быть вынесены в отдельный блок комментариев ниже.

Пример:

/\* MarcRecordCollection.cs -- Container for MarcRecord’s  
 \*/

# Форматирование исходных текстов

## Табуляция и отступы

Для создания отступов используются символы пробела. Принят отступ 4 знакоместа. Замена на символы табуляции недопустима.

Логический отступ (означающий вложенную конструкцию) формируется добавлением 4 пробелов.

## Пробелы

Пробелы ставятся:

* между операторами;
* между операторами и операндами;
* после запятой, точки с запятой;
* перед и после скобок (круглых и квадратных). Исключения: 1) пустые пары скобок () и [], 2) квадратные скобки в объявлениях массивов, например object[];
* перед и после двоеточия, исключение составляют метки (в т.ч. в операторе switch).

Пробелы не ставятся:

* перед запятой, точкой.

Примеры:

z∙=∙x∙+∙y;↵

for∙(∙int∙i∙=∙0;∙i∙<∙list.Length;∙i++∙)↵  
{↵  
∙∙∙∙list∙[∙i∙].Dump∙();↵  
}

## Перенос строк

Желательно избегать строк длиной более 80-символов, для длинных строк использовать перенос.

Строку можно переносить в следующих позициях:

* после запятой;
* перед оператором.

Перенесенная строка по отношению к своему началу должна иметь дополнительный отступ в два пробельных символа. Пример:

LongMethodCall  
∙∙(  
∙∙∙∙argument1,↵  
∙∙∙∙argument2,↵  
∙∙∙∙argument3,↵  
∙∙∙∙argument4↵  
∙∙);

Оператор (такой как «сложение», «логическое или» и т.д.) не оставляется последним на строке, а переносится на следующую. Пример:

result∙=∙"Line begins:∙" ∙+∙var1↵  
∙∙+∙"∙middle∙of∙the∙line∙"↵  
∙∙+∙var2∙+∙"∙(line∙ends).";

Для длинных вызовов сложных методов можно использовать следующий метод переноса строк:

ComplexMethodCall↵  
∙∙(↵  
∙∙∙∙firstArgument,∙//∙some∙comments↵  
∙∙∙∙"Second argument",∙//∙another∙comment↵  
∙∙∙∙thirdArgument↵  
∙∙);

При переносе предпочтительнее не разбивать низкоуровневые конструкции, лучше делить строку по границам блоков б***о***льшего уровня (например, скобок).

**/\*∙bad style∙\*/**↵  
result∙=∙(∙a∙+∙b∙)∙/∙(∙c∙–∙d↵  
∙∙+∙e∙)∙–∙g∙\*∙h; ↵

**/\* better \*/**↵  
result∙=∙(∙a∙+∙b∙)∙/∙(∙c∙–∙d∙+∙e∙)↵  
∙∙-∙g∙\*∙h;

## Другое форматирование

Открывающая и закрывающая фигурные скобки должны располагаться на отдельных строках. Закрывающая фигурная скобка должна располагаться строго под соответствующей открывающей скобкой.

Код, заключенный между парой фигурных скобок, должен иметь отступ в один символ табуляции.

Желательно, чтобы на одной строке присутствовало не более одного оператора.

Между логически завершенными блоками кода допускается наличие одной пустой строки.

### «Табличное» форматирование

Где возможно, следует использовать "табличное" форматирование, подчеркивающее логический "ритм" в коде:

newColor.Red∙∙∙=∙brightColor.Red∙∙∙-∙10;↵  
newColor.Green∙=∙brightColor.Green∙–∙10;↵  
newColor.Blue∙∙=∙brightColor.Blue∙∙-∙10;↵

# Оформление и именование отдельных элементов языка C#

## Регионы

Весь код должен заключаться в регионы. Названия регионов должны быть на английском языке.

Типичные названия регионов:

* Initialization;
* GUI events handing;
* Override inherited members;

Директива #region отделяется от окружающего кода пустыми строками.

## Директива using

Порядок следования пространств имен в using должен быть следующим:

* .NET Framework;
* сторонние сборки;
* собственные библиотеки;
* пространства имен проекта.

Группы директив using отделяются друг от друга пустой строкой. Внутри групп элементы упорядочиваются в алфавитном порядке.

Директива using должна располагаться вне пространства имен.

## Комментарии

Все комментарии в тексте программ должны быть только на английском языке.

Для пространных объяснения каких-либо особенностей кода предпочтительнее использовать многострочный комментарий следующего вида:

/\*  
 \* This code implements some algorithm,  
 \* see Some Book, pp. 100-110  
 \*/

Для комментирования "в одну строчку" используется следующий стиль:

bool hasProblems; // If something goes wrong  
  
ComplexMethodCall  
 (  
 firstArgument, // some comments  
 "Second argument", // another comment  
 thirdArgument  
 );

Комментарии // часто применяются для «закомментирования» (временного исключения) участков кода (см. горячие клавиши среды разработки). Однако для больших кусков кода (более одного метода или более 10 строк) предпочтительнее использовать препроцессор:

x = 1.0; // Initialize variables  
 y = 2.0;  
 // z = x.op\_Add ( y ); // Strange code, isn’t

z = x + y;  
┼ #if NOTDEFINED  
│ // this code is too buggy  
│ result = Math.Atan2 ( z, x + y ); // excepton will be raised  
┼ #endif

Обратите внимание, что //, «закомментирующие» код, помещаются с начала строки, в отличие от обычного однострочного комментария.

Также применяются однострочные комментарии для аннотации кода (они автоматически помещаются окно задач среды разработки) следующего вида:

// TODO check for locked records

// TODO what if file does not exist?

Кроме аннотации TODO допускается применение аннотаций HACK и UNDONE.

В стабильную версию Ars Magna не должны попадать большие куски закомментированного кода – перед коммитом в транк их желательно удалять.

### Комментарии для самодокументирования

Комментарии для самодокументирования также должны быть только на английском языке.

Заполнение следующих тегов самодокументирования обязательно: <summary>, <param>, <returns>.

/// <summary>  
/// Inserts given text repeatly.  
/// </summary>  
/// <param name="str">Text to insert.</param>  
/// <param name="ntimes">How many times.</param>  
/// <returns>Same object.</returns>  
public MarkupText Repeat ( string str, int ntimes )  
{  
 StringBuilder builder = new StringBuilder ();

for ( ; ntimes > 0; ntimes-- )  
 {  
 builder.Append ( str );  
 }

if ( builder.Length > 0 )

{  
 SubPieces.Add ( builder.ToString () );  
 }

return this;  
}

Для упрощения создания самодокументирующих комментариев рекомендуется использовать GhostDoc (<http://submain.com/download/ghostdoc/>).

## Общие правила именования для идентификаторов

Индикаторы должны содержать следующие символы:

* Латинские буквы (прописные и строчные);
* Арабские цифры;
* Знак подчеркивания.

Другие символы в идентификаторах недопустимы. В частности, нельзя называть типы, методы и переменные по-русски (несмотря на то, что C# позволяет это). Нельзя использовать символ подчеркивания в публичных идентификаторах (чтобы не вызывать проблем у пользователей VisualBasic.Net и других ущербных языков).

В проекте Ars Magna не применяется венгерская нотация для префиксов, означающих тип параметра, поля или переменной.

По возможности следует избегать аббревиатур или сокращений в качестве частей имен идентификаторов. Например, вместо GetWin предпочтительнее использовать GetWindow. Если все же используются аббревиатуры, то аббревиатуры длиной не более двух символов следует капитализировать (например, AM.UI), а более длинные — давать в нотации Pascal или Camel (например, RtfFormatProvider).

Следует избегать использования идентификаторов, конфликтующих с ключевыми словами языков C#, VB.NET, VC++.NET, J# и JScript.Net. По крайней мере, такие идентификаторы не должны быть видны "наружу".

Примерный список ключевых слов:

AddHandler, AddressOf, Alias, And, Ansi, As, Assembly, Auto, Base, Boolean, ByRef, Byte, ByVal, Call, Case, Catch, CBool, CByte, CChar, CDate, CDec, CDbl, Char, CInt, Class, CLng, CObj, Const, CShort, CSng, CStr, CType, Date, Decimal, Declare, Default, Delegate, Dim, Do, Double, Each, Else, ElseIf, End, Enum, Erase, Error, eval, Event, Exit, extends, ExternalSource, False, Finalize, Finally, Float, For, Friend, Function, Get, GetType, Goto, Handles, If, Implements, Imports, In, Inherits, instanceof, Integer, Interface, Is, Let, Lib, Like, Long, Loop, Me, Mod, Module, MustInherit, MustOverride, MyBase, MyClass, Namespace, New, Next, Not, Nothing, NotInheritable, NotOverridable, Object, On, Option, Optional, Or, Overloads, Overridable, Overrides, package, ParamArray, Preserve, Private, Property, Protected, Public, RaiseEvent, ReadOnly, ReDim, Region, REM, RemoveHandler, Resume, Return, Select, Set, Shadows, Shared, Short, Single, Static, Step, Stop, String, Structure, Sub, SyncLock, Then, Throw, To, True, Try, TypeOf, Unicode, Until, Val, volatile, When, While, With, WithEvents, WriteOnly, Xor

## Пространства имен

Для пространств имен используется нотация Pascal.

Все типы должны быть заключены в пространства имен. Не должно быть типов, не заключенных в пространство имен.

К имени пространства имен, содержащего типы, которые обеспечивают функциональность времени разработки для основного пространства имен, добавляется суффикс .Design. Например, AM.Library.Loans.Design.

Вложенные пространства имен должны иметь зависимость от типов содержащего их пространства имен. Например, типы в AM.Library.Loans.Design должны зависеть от AM.Library.Loans.

Предпочтительнее имена пространств имен во множественном числе, если это семантически оправданно. Например, лучше использовать AM.Library.Loans, чем AM.Library.Loan.

## Шаблон объявления типов

Пример (обратите внимание на использование пробелов вокруг двоеточия):

class Loan   
 : LibrarySubdivision  
 , IInspectable  
{  
  
#region Class fields  
  
 private string name;  
  
 public string Name  
 {  
 get  
 {  
 return name;  
 }  
 }  
  
#endregion  
  
#region Initialization  
  
 public Loan ( string name ) : base ()  
 {  
 this.name = name;  
 }  
  
#endregion  
  
#region IInspectable members  
  
 private bool IInspectable.Inspect ()  
 {  
 return true;  
 }  
  
#endregion  
  
}

## Классы

В качестве имен классов используются существительные или именные группы. Применяется нотация Pascal.

Нельзя использовать префиксы типа, например, называть класс CBook (правильно — просто Book).

Следует избегать использования имен классов, которые дублируют общепринятые пространства имен .NET Framework. Например, не используйте в качестве имен классов следующие имена: System, Collections, Forms или UI.

Нельзя использовать одинаковые имена для класса и пространства имен.

## Структуры

Правила именования для структур такие же, как и для классов (см. пункт ).

## Перечисления

Правила именования для перечислений такие же, как и для классов (см. пункт ).

Нельзя использовать суффикс Enum в имени перечислимого типа.

Если перечисление является битовым полем (т.е. имеет атрибут FlagsAttribute), его имя должно быть по возможности во множественном числе, иначе — в единственном.

## Интерфейсы

Правила именования интерфейсов такие же, как и для классов (см. пункт ), но имя всегда предваряется заглавной латинской буквой I.

Дополнительно в качестве имен интерфейсов можно использовать прилагательные, описывающие поведение класса, реализующего интерфейс. Например: IPersistable или IFormatable.

## Члены типов

Порядок следования членов типа должен быть следующим:

* вложенные типы;
* константы;
* события;
* поля класса;
* свойства класса;
* публичные методы;
* защищенные (protected) и внутренние (internal) методы;
* приватные (private) методы.

Внутри групп свойства, методы и поля упорядочиваются по алфавиту. Для полей, тесно связанных со свойством или методом допускается размещение в непосредственной близости.

Акцессор private всегда указывается явно. Не допускаются члены с неуказанным акцессором.

Между членами типа оставляется пустая строка. Для связанного набора полей или для свойства и соответствующего ему поля пустую строку можно опускать.

Конструкторы экземпляров и перегруженные методы рекомендуется располагать последовательно в порядке нарастания сложности:

public class SampleClass  
{  
 public SampleClass () : this ( “abc”, 1 )   
 {  
 }  
  
 public SampleClass ( string firstPar ) : this ( firstPar, 1 )  
 {  
 }  
  
 public SampleClass ( string firstPar, int secondPar )  
 {  
 }  
}

## Свойства классов/структур/интерфейсов

В качестве имен свойств используются существительные или именные группы.

Для публичных свойств применяется нотация Pascal, для остальных — Camel.

Применяется нотация Pascal. Не используется венгерская нотация для префиксов.

Имя приватного свойства (с акцессором private) должно начинаться с символа подчеркивания.

Допускается не разделять пустой строкой публичное свойство и связанное с ним приватное поле. Пример:

private Color \_backColor;  
  
public Color BackColor  
{  
 get  
 {  
 return backColor;  
 }  
}

## Поля классов/структур

В качестве имен полей используются существительные или именные группы.

Для публичных (и защищенных) полей применяется нотация Pascal, для остальных — Camel.

Имя приватного поля (с акцессором private) должно начинаться с символа подчеркивания.

## Методы классов/структур/интерфейсов

### Собственно методы

В качестве имен методов используются глаголы или глагольные группы.

Применяется нотация Pascal.

Имя приватного метода (с акцессором private) должно начинаться с символа подчеркивания).

Примеры:

void RemoveAll ();  
  
char[] ReadChars ( int howMany );  
  
void Invoke ();

### Параметры методов

В качестве имен параметров используются существительные или именные группы.

Применяется нотация Camel. Не используется венгерская нотация для префиксов.

Не используются зарезервированные параметры.

Имена параметров должны быть достаточно описательными, чтобы по имени параметра и его типа в большинстве случаев можно было догадаться о его назначении. Имена должны описывать значения параметров, а не их тип.

### Локальные переменные

В качестве имен локальных переменных используются существительные или именные группы.

Применяется нотация Camel.

В качестве целочисленного счетчика в циклах рекомендуется использовать переменные с именами i, j, k, l, m, n.

На одной строке должно быть не более одного объявления локальной переменной. Пример:

int level; // indentation level  
int tableSize; // size of table (in bytes)

Не допускается размещать в одной строке два и более объявления переменных. Локальные переменные отделяются от остального кода метода одной пустой строкой (см. пример ниже).

Объявлять переменные желательно как можно ближе к месту их использования, при объявлении желательно инициализировать их:

int hoursWorked = thisMonth.Hours + previousMonth.Hours;  
  
if ( hoursWorked < minimumAllowed )  
{  
 manager.WarnAboutHours ( worker, hoursWorked );  
}

## Открытость/закрытость для наследования

Класс не предназначен явно для наследования, если:

* в нём нет защищённых (protected) членов;
* в нём нет виртуальных (virtual) методов/свойств;
* в нём нет абстрактных (abstract) методов/свойств.

Подобные классы следует снабжать модификатором sealed, чтобы подчеркнуть для прикладного программиста, что данный класс не проектировался как наследуемый.

# Производные типы

## Атрибуты

### Классы атрибутов

Имя класса атрибута всегда должно содержать суффикс Attribute. Класс атрибута должен быть наследником System.Attribute. Остальные требования такие же, как и у обычных классов (см. пункт ).

И наоборот, если имя класса содержит суффикс Attribute, этот класс должен быть наследником System.Attribute.

Класс атрибута обязательно должен быть снабжен атрибутом AttributeUsageAttribute, в конструкторе которого перечисляются все возможные области применения данного атрибута:

[AttributeUsage ( AttributeTargets.Property ) ]  
public class PrintableAttribute: Attribute  
{  
}

Изолируйте (устанавливайте модификатор sealed) классы атрибутов, где это возможно, чтобы от них нельзя было наследоваться.

## Исключения

Имя класса исключения всегда должно содержать суффикс Exception. Класс исключения должен быть наследником System.ApplicationException. Остальные требования такие же, как и у обычных классов (см. пункт ).

И наоборот, если имя класса содержит суффикс Exception, он должен быть наследником System.Exception.

## Делегаты

В качестве имен классов делегатов используются глаголы или глагольные группы. Применяется нотация Pascal. Суффикс Delegate не добавляется.

## События

### Классы событий

В качестве имен классов событий используются глаголы или глагольные группы. Применяется нотация Pascal.

В качестве имен обработчиков событий используются глаголы или глагольные группы. Применяется нотация Pascal.

### Обработчики событий

Имя обработчика события должно содержать суффикс EventHandler. И наоборот, если имя класса содержит суффикс EventHandler, он должен быть наследником System.EventHandler.

В обработчике события должны быть определены два параметра: sender и ea. Первый параметр всегда должен иметь тип object, даже если есть возможность использовать более специфический тип. Второй параметр должен содержать информацию о событии и принадлежать соответствующему специфическому типу.

Пример обработчика события:

public delegate void MouseEventHandler   
 (  
 object sender,  
 MouseEventArgs ea   
 );

### Аргументы событий

В имени класса аргумента события должен присутствовать префикс EventArgs. И наоборот, если имя класса содержит суффикс EventArgs, он должен быть наследником System.EventArgs.

Пример:

public class MouseEventArgs: EventArgs  
{  
 public int X;  
 public int Y;  
}

Имена событий «до» должны содержать префикс “Before”, событий «после» — префикс “After”. Например: BeforeClose, AfterClose.

На каждое публичное событие должен быть обеспечен protected метод с именем OnXxx. Этот метод должен иметь только параметр ea, т.к. отправителем всегда является данный экземпляр типа.

## Разрешения

Имя класса разрешения должно содержать суффикс Permission. И наоборот, если в имени класса содержится суффикс Permission, он должен реализовывать System.Security.IPermission.

## Контейнеры

Имена классов-контейнеров подчиняются следующим правилам:

|  |  |  |
| --- | --- | --- |
| **Контейнер** | **Суффикс имени** | **Условие** |
| Стек (магазин) | Stack | Наследник System.Collections.Stack |
| Очередь | Queue | Наследник System.Collections.Queue |
| Словарь | Dictionary | Реализует System.Collections.IDictionary |
| Коллекция | Collection | Реализует System.Collections.ICollection |
| Список | List | Реализует System.Collections.IList |

# Оформление синтаксических конструкций

## Оператор if

Блоки кода в операторе if/else обязательно заключаются в фигурные скобки (даже состоящие из единственного оператора).

Группы в булевом выражении заключаются в скобки. Например:

if ( ( a == 0 ) && ( b == 0 ) )  
{  
 // some code here  
}

Пример сложного оператора if:

if ( reader.IsDebtor () )  
{  
 HandleDebtorReader ( reader );  
}  
else if ( reader.IsBlocked () )  
{  
 HandleBlockedReader ( reader );  
}  
else  
{  
 HandleNormalReader ( reader );  
}

## Операторы for и foreach

Блок кода операторов for и foreach обязательно должен заключаться в фигурные скобки.

Условное выражение в операторе for записывается по тем же правилам, что и в операторе if (см. пункт ).

Пример оформления оператора foreach:

foreach ( Author author in allAuthors )  
{  
 authorIndex.Add ( author );  
 Console.WriteLine ( author.Name );  
}

Пустой оператор for/foreach записывается следующим образом:

for ( Book book = First; book != null; book = book->Advance () )   
{  
 // nothing to do  
}

## Операторы while

Условное выражение в операторе while записывается по тем же правилам, что и в операторе if (см. пункт ).

## Оператор switch

Пример оформления оператора switch:

switch ( condition )  
{  
case 1:  
 // some code  
 break;  
case 2:  
 // some code  
 break;  
default:  
 // some code  
 break;  
}

## try/catch

Пример оформления блока try/catch:

try  
{  
 CallWiseCode ();  
}  
catch ( MyException ex )  
{  
 Console.WriteLine ( "Exception occurred: " + ex.ToString () );  
}

Вариант без обработки (подавление исключений):

try  
{  
 CallWiseCode ();  
}  
catch   
{  
 // eat any exception  
}

Обратите внимание на явное указание подавления исключений.

## Оператор using

Блок кода оператора using обязательно должен заключаться в фигурные скобки. Пример:

using ( IMarcRecordReader rdr = new MarcIsoRecordReader ( file ) )  
{  
 MarcRecord rec = rdr.ReadRecord ();  
}

# Нумерация версий

В этом разделе должны быть описаны правила нумерации версий.

# Прочие замечания

## Именование элементов управления

Для элементов управления предпочтительным является следующий стиль именования:

Button cancelButton;  
TextBox nameTextBox;  
Panel upperPanel; // and so on

## Конструкторы экземпляров классов

### Функциональность конструкторов

Сведите к минимуму работу, выполняемую в конструкторе. Конструкторы должны только «раскладывать» свои параметры по [приватным] полям экземпляра.

### Метод Initialize

Реальную инициализацию экземпляра (если инициализация отложенная, т. е. осуществляется не в конструкторе) желательно оформлять в виде защищенного метода Initialize:

protected void Initialize ();

Кроме того, в метод Initialize рекомендуется выносить общий функционал перегруженных конструкторов, если не удаётся оформить её через взаимные вызовы конструкторов.

Такая практика помогает легко находить, где происходит инициализация объекта (кроме того, инициализационный код собирается в одном месте).

### Классы, экземпляры которых не должны создаваться клиентским кодом

public sealed class LoanDepartment  
{  
 // internal constructor prevents the class  
 // from being created by client code.  
 internal LoanDepartment  
 {  
 // Some code may be here  
 }  
}

## Очистка объекта

### Рекомендуемая очистка объекта

public class MyClass : IDisposable  
{  
 ~MyClass ()  
 {  
 Dispose ( false );  
 }  
   
 public void Dispose ()  
 {  
 Dispose ( true );  
 GC.SuppressFinalize ( this );  
 }  
  
 protected virtual void Dispose ( bool disposing )  
 {  
 if ( disposing )  
 {  
 ...  
 }  
 else  
 {  
 ...  
 }  
 }  
}

# Система контроля версий

По мере работы над проектом необходимо вести учёт внесённых в функционал изменений в файле Changes.docx.

## Чеклист действий перед коммитом

|  |  |
| --- | --- |
| № | Действие |
|  | Убедиться, что солюшн можно пересобрать полностью (Rebuild All). |
|  | Убедиться, что все тесты запускаются и успешно проходят. |
|  | Пересобрать солюшн в пакетном режиме (Batch Build) для всех предусмотренных конфигураций (Debug, Release, какие ещё появятся). |
|  | Убедиться, что все изменения зафиксированы в Changes.docx, ничего не забыто. Переименовать файл в ChangesN.docx, где N – номер следующей ревизии в репозитории. |

В систему контроля версий можно и нужно добавлять содержимое папки bin (в частности, подпапок bin/Debug и bin/Release) (чтобы скачавший мог взять готовые сборки, не пересобирая проект), но нельзя добавлять содержимое папки obj, чтобы не замусоривать систему контроля версий.

Кроме того, нельзя добавлять следующие настроечные файлы:

* \*.DotSettings.user
* \*.Resharper\*
* \*.suo

Нельзя также папки, созданные инструментарием вроде Resharper, XUnit или аналогичными.

## Чеклист действий после коммита

|  |  |
| --- | --- |
| № | Действие |
|  | Скачать (checkout) солюшн в пустую папку. |
|  | Убедиться, что солюшн можно пересобрать полностью (Rebuild All). |
|  | Убедиться, что все тесты запускаются и успешно проходят. |
|  | Залить в рабочее пространство проекта |

При обнаружении ошибки, выявить недостающие (не добавленные в систему контроля версий) файлы и добавить следующим коммитом.

## Создание публичного (рекомендуемого) билда

При значимом изменении версии проекта (например, с 1.2 на 1.3 или на 2.0) создаётся zip-архив, который выкладывается в рабочее пространство.

Также zip-архив выкладывается при устранении критических ошибок в коде.

zip-архив должен содержать рабочую копию проекта за вычетом служебных папок SVN и настроечных файлов, перечисленных в пункте 9.1.

# Ссылки

Документы, использованные при составлении Соглашения:

1. Шатохина Н.А. Рекомендации по разработке библиотеки классов. <http://www.gotdotnet.ru>
2. Krüger M. C# Coding Style Guide. <http://www.icsharpcode.net>
3. Vorontsov S. aka V©R©N Соглашения о кодировании. <http://www.aspnetmania.com>
4. Lowy J. C# Coding Standard. <http://www.idesign.net>
5. Соглашения об оформлении кода команды RSDN <http://rsdn.ru>