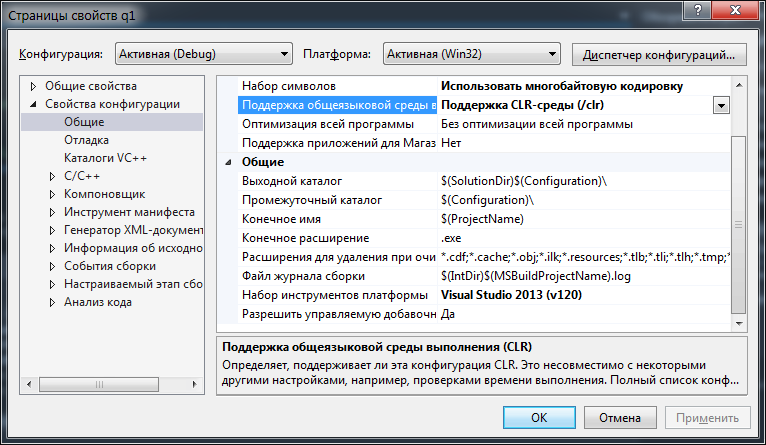
**Property в С++**

Потрібно використовувати режим підтримки CLR



**property (расширения компонентов C++)**

Объявляет свойство, являющееся функцией-членом, которая действует и доступна как элемент данных или элемент массива.

Можно объявить один из следующих типов свойств.

Простое свойство

По умолчанию создает метод доступа set, который присваивает значение свойства, метод доступа get, который получает значение свойства, и создаваемый компилятором закрытый элемент данных, содержащий значение свойства.

Блок свойств

Используется для создания определяемых пользователем методов доступа get и/или set. Свойство доступно для чтения и записи, если заданы оба метода доступа — get и set, доступно только для чтения, если задан только метод доступа get, и доступно только для записи, если задан только метод доступа set.

Вы должны явно объявить элемент данных для хранения значения свойства.

Индексированное свойство

Блок свойств, который можно использовать для получения и задания значения свойства, указанного одним или несколькими индексами.

Вы можете создать индексированное свойство, которое имеет либо заданное пользователем имя, либо имя по умолчанию. Имя свойства индекса по умолчанию — это имя класса, в котором это свойство определено. Чтобы объявить свойство по умолчанию, укажите ключевое слово default вместо имени свойства.

Вы должны явно объявить элемент данных для хранения значения свойства. Для индексированного свойства элементом данных обычно является массив или коллекция.

**es7h5kch.collapse_all(ru-ru,VS.140).gifСинтаксис**

C++

property type property\_name;

property type property\_name {

access-modifier type get() inheritance-modifier {property\_body};

access-modifier void set(type value) inheritance-modifier {property\_body};

}

property type property\_name[index\_list] {

access-modifier type get(index\_list) inheritance-modifier {property\_body};

access-modifier void set(index\_list, value) inheritance-modifier {property\_body};

}

property type default[index\_list] {

access-modifier type get(index\_list) inheritance-modifier {property\_body};

access-modifier void set(index\_list, value) inheritance-modifier {property\_body};

}

**es7h5kch.collapse_all(ru-ru,VS.140).gifПараметры**

type

Тип данных значения свойства и, следовательно, самого свойства.

property\_name

Имя свойства.

access-modifier

Квалификатор доступа. Допустимые квалификаторы — static и virtual.

Методы доступа get или set не требуется согласовывать с квалификатором virtual, но они должны быть согласованы с квалификатором static.

inheritance-modifier

Квалификатор наследования. Допустимые квалификаторы — abstract и sealed.

index\_list

Разделенный запятыми список из одного или нескольких индексов. Каждый индекс состоит из типа индекса и необязательного идентификатора, который может использоваться в теле метода свойства.

value

Значение для назначения свойству в операции set или извлечения в операции get.

property\_body

Тело метода свойства метода доступа set или get. Параметр property\_body может использовать index\_list для доступа к базовому элементу данных свойства или в качестве параметров в пользовательской обработке.

[**Среда выполнения Windows**](javascript:void(0))

Дополнительные сведения см. в разделе [Свойства (C++/CX)](https://msdn.microsoft.com/library/windows/apps/hh755807.aspx).

**es7h5kch.collapse_all(ru-ru,VS.140).gifТребования**

Параметр компилятора: **/ZW**

[**Среда CLR**](javascript:void(0))

Синтаксис

modifier property type property\_name;

modifier property type property\_name {

modifier void set(type);

modifier type get();

}

modifier property type property\_name[index-list, value] {

modifier void set(index-list, value);

modifier type get(index-list);

modifier property type default[index];

}

Параметры

modifier

Модификатор, который может использоваться в объявлении свойства или в методе доступа get/set свойства. Возможными значениями являются static и virtual.

тип

Тип значения, представленного свойством.

property\_name

Параметр или параметры для метода raise; необходимо соответствие сигнатуре делегата.

index\_list

Разделенный запятыми список из одного или нескольких индексов, указанных в квадратных скобках (оператор индекса, ([])). Для каждого индекса укажите тип и (необязательно) идентификатор, который может использоваться в теле метода свойства.

Примечания

В первом примере синтаксиса показано простое свойство, которое неявно объявляет методы set и get. Компилятор автоматически создает закрытое поле для хранения значения свойства.

Во втором примере синтаксиса показан блок свойств, который явно объявляет методы set и get.

В третьем примере синтаксиса показано определяемое пользователем свойство индекса. Свойство индекса принимает параметры, а также значение для установки или получения. Необходимо указать имя для свойства. В отличие от простого свойства методы set и get свойства индекса должны быть явно определены, и необходимо указать имя для свойства.

В четвертом примере синтаксиса показано свойство по умолчанию, которое предоставляет подобный массиву доступ к экземпляру типа. Ключевое слово default служит только для указания свойства по умолчанию. Имя свойства по умолчанию — это имя типа, в котором это свойство определено.

Ключевое слово property может встречаться в классе, интерфейсе или типе значения. Свойство может иметь функцию get (только чтение), функцию set (только запись) или обе эти функции (чтение и запись).

Имя свойства не может совпадать с именем управляемого класса, к которому оно относится. Функция получения свойства должна иметь тип, который совпадает с типом последнего параметра в соответствующей функции задания свойства.

В отношении клиентского кода свойство имеет вид обычного элемента данных и может записываться или считываться с помощью того же синтаксиса, что и элемент данных.

Методы get и set не требуется согласовывать с модификатором virtual.

Доступность методов get и set может отличаться.

Определение метода свойства может отображаться за пределами тела класса, так же, как в случае обычного метода.

Методы get и set для свойства должны быть согласованными с модификатором **static**.

Свойство является скалярным, если его методы get и set соответствуют следующему описанию.

* Метод get не имеет параметров и возвращает значение типа T.
* Метод set имеет параметр типа T и возвращает значение типа **void**.

Должно существовать только одно скалярное свойство, объявленное в области с тем же идентификатором. Скалярные свойства не могут быть перегружены.

При объявлении элемента данных свойства компилятор вставляет элемент данных — иногда называемый «резервным хранилищем» — в класс. Однако имя элемента данных имеет такую форму, что вы не можете ссылаться на этот элемент в источнике, как если бы это был фактический элемент данных содержащего класса. Используйте Ildasm.exe для просмотра метаданных для вашего типа и созданного компилятором имени для резервного хранилища свойства.

В блоке свойств разрешается разная доступность для методов доступа. То есть метод set может быть открытым, а метод get может быть закрытым. Однако было бы ошибкой, если бы метод доступа имел менее строгий доступ, нежели в объявлении самого свойства.

property — контекстно-зависимое ключевое слово. Дополнительные сведения см. в разделе [Контекстные ключевые слова (расширения компонентов C++)](https://msdn.microsoft.com/ru-ru/library/8d7y7wz6.aspx).

Дополнительные сведения о свойствах см. в следующих разделах.

* [Индексированные свойства](https://msdn.microsoft.com/ru-ru/library/52d21xwx.aspx)
* [Статические свойства](https://msdn.microsoft.com/ru-ru/library/6b6t25z7.aspx)
* [Виртуальные свойства](https://msdn.microsoft.com/ru-ru/library/zdkaw5ss.aspx)
* [Многомерные свойства](https://msdn.microsoft.com/ru-ru/library/btd684da.aspx)
* [Перегрузка методов доступа к свойству](https://msdn.microsoft.com/ru-ru/library/s0sydf7x.aspx)
* [Практическое руководство. Объявление абстрактных и запечатанных свойств](https://msdn.microsoft.com/ru-ru/library/ms235576.aspx)

**es7h5kch.collapse_all(ru-ru,VS.140).gifТребования**

Параметр компилятора: **/clr**

**es7h5kch.collapse_all(ru-ru,VS.140).gifПримеры**

В следующем примере показано объявление и использование элемента данных свойства и блока свойств. В этом примере также показывается, что метод доступа свойства можно определить вне класса.

// mcppv2\_property.cpp

// compile with: /clr

using namespace System;

public ref class C {

int MyInt;

public:

// property data member

property String ^ Simple\_Property;

// property block

property int Property\_Block {

int get();

void set(int value) {

MyInt = value;

}

}

};

int C::Property\_Block::get() {

return MyInt;

}

int main() {

C ^ MyC = gcnew C();

MyC->Simple\_Property = "test";

Console::WriteLine(MyC->Simple\_Property);

MyC->Property\_Block = 21;

Console::WriteLine(MyC->Property\_Block);

}

Вывод

тест

**Приклад.**

#include<iostream>

using namespace std;

ref class my

{

array<int>^ a= gcnew array<int>(5) ;

int m;

public:

property int Sim;

property int derr{

int get()

{

return m;

}

void set(int value)

{

m = 2 \* value;

}

}

property int MyArray[int]{

int get(int j)

{

return a[j];

}

void set(int j, int value)

{

a[j] = value;

}

}

property int default[int]{

int get(int j)

{

return a[j];

}

void set(int j, int value)

{

a[j] = value;

}

}

my(){};

~my(){};

private:

};

void main()

{

my^ M=gcnew my();

M->Sim = 10;

cout << M->Sim << endl;

M->derr = 12; // M->derr.set(12);

cout << M->derr << endl;

M->MyArray[0] = 17;

M->MyArray[1] = 55;

M[0] = 13;

cout << M[0] << endl;

cout << M[1] << endl;

system("pause");

}

ПРИКЛАД 2. Приклад інкапсуляції у класичному С++ і у С++ з керованими типами даних

---------------- Класичний підхід --------------------

// ConsoleApplication104.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include<iostream>

#include"Rectangle.h"

using namespace std;

class Rect {

private:

double a\_, b\_;

public:

void setA(double value)

{

if (value >= 0)

a\_ = value;

else

throw "error";

}

double getA()

{

return a\_;

}

void setB(double value)

{

if (value >= 0)

b\_ = value;

else

throw "error";

}

double getB()

{

return b\_;

}

Rect(double a, double b) {

setA(a);

setB(b);

}

Rect()

{

}

void input() {

cout << "a=";

double t;

cin >> t;

setA(t);

cout << "b=";

cin >> t;

setB(t);

}

};

============================== Підхід з керованими типами даних =============

#pragma once

ref class Rectangle

{

private:

double a\_, b\_;

public:

property double Temp;

property double A {

void set(double value)

{

if (value >= 0)

a\_ = value;

else

throw "error";

}

double get()

{

return a\_;

}

}

property double B {

void set(double value)

{

if (value >= 0)

b\_ = value;

else

throw "error";

}

double get()

{

return b\_;

}

}

property double default[int]{

void set(int sideNumber, double value)

{

switch (sideNumber)

{

case 1:A = value;

break;

case 2: B = value;

default: throw "bad side number";

}

}

double get(int sideNumber)

{

switch (sideNumber)

{

case 1:return A;

case 2: return B;

default: throw "bad side number";

}

}

}

Rectangle(double a, double b);

Rectangle()

{

}

~Rectangle();

};

--------------------------- cpp

#include "stdafx.h"

#include "Rectangle.h"

Rectangle::Rectangle(double a, double b)

{

A = a;

B = b; //B.set(b)

}

Rectangle::~Rectangle()

{

}

===================================== main ============

int main()

{

Rect\* r2= new Rect(); //Класичний некерований тип даних

Rectangle^ r1 = gcnew Rectangle(); //Керований тип даних

r2->setA(3);

r1->A = 3; //r1->A.set(3);

r1[1] = 3;

r1[2] = 56; //r1->B=56;

cout << "A=" << r1->A << endl; //cout << "A=" << r1->A.get() << endl;

cout << "A=" << r1[1] << endl;

cout << endl;

system("pause");

return 0;

}