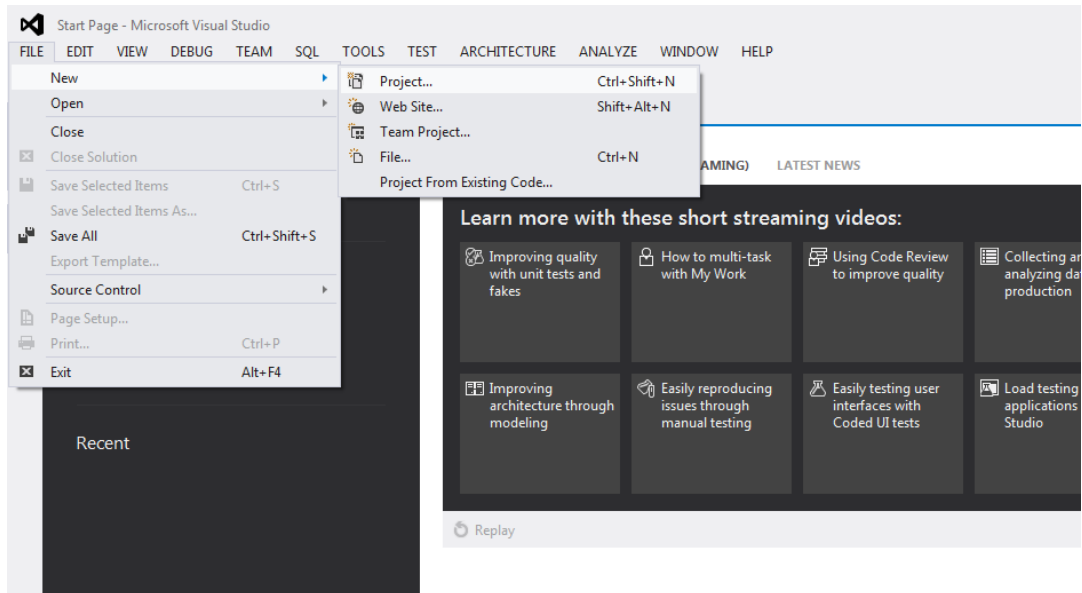
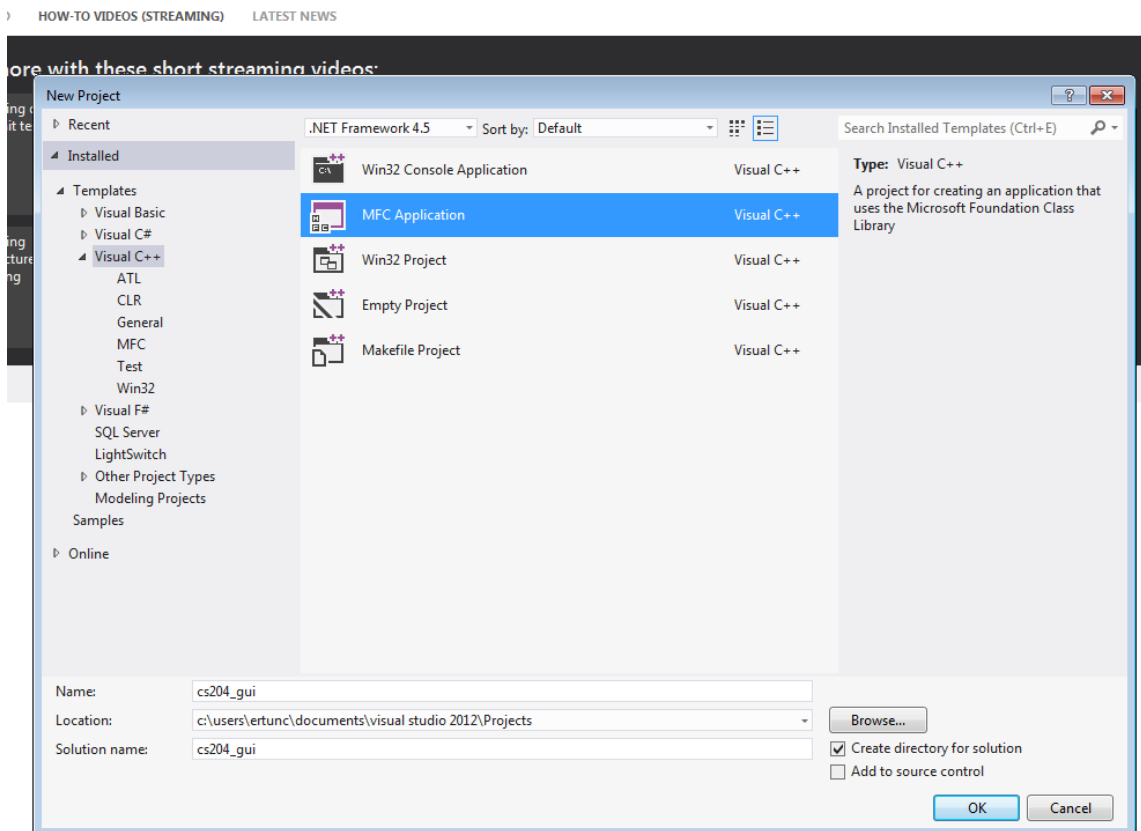


Creating an MFC Project in Visual Studio 2012

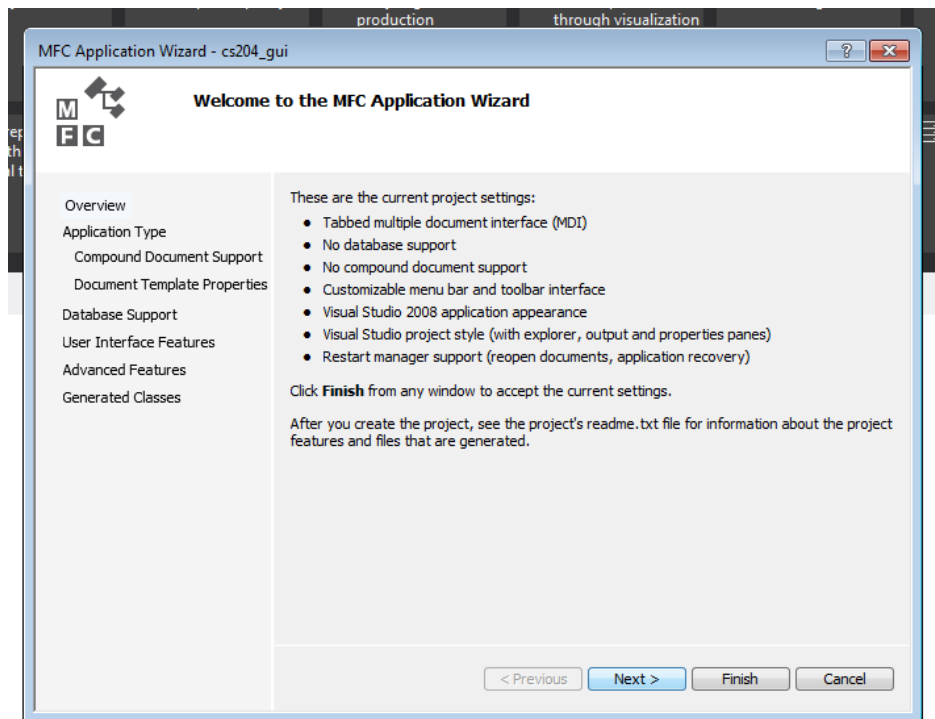
Step1:



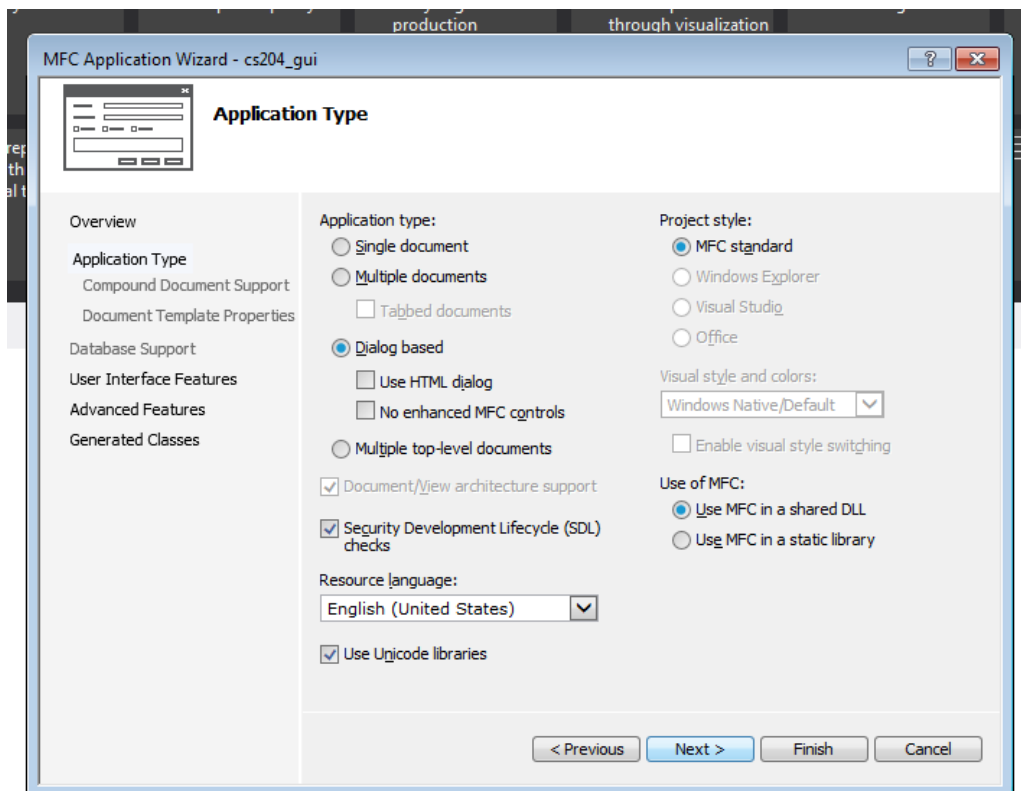
Step2:



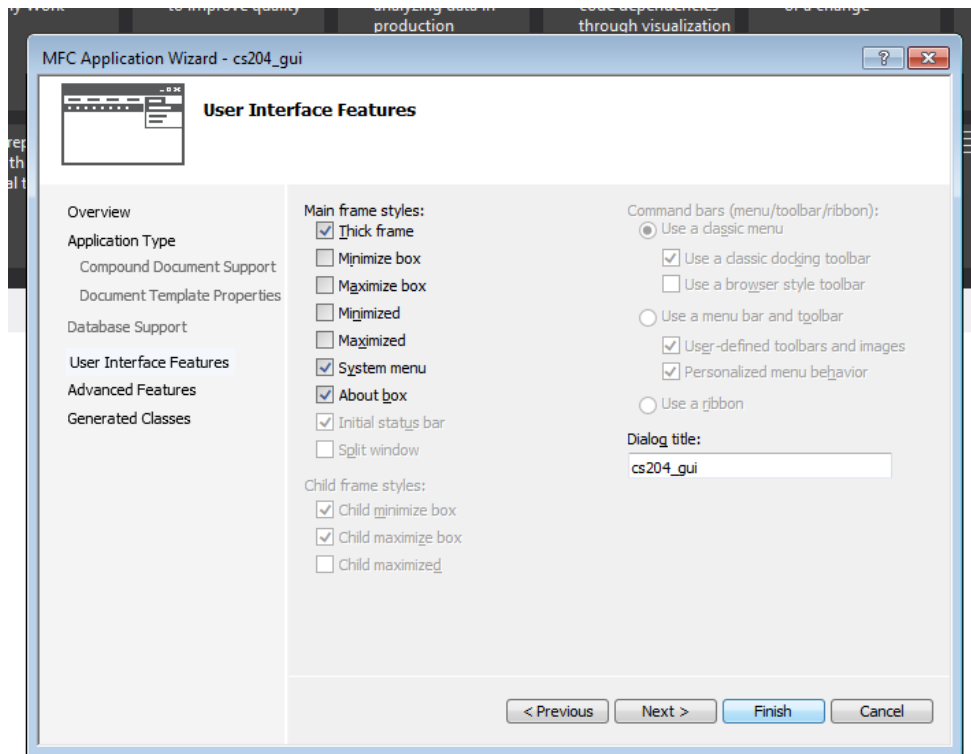
Step3:



Step4:



Step5:





You don't need to continue this wizard any longer, you can press Finish to finish creating your project.

We will introduce some of the core members of the toolbox that are widely used in Graphical User Interface (GUI) design.

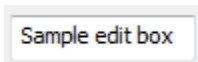
Button



Button is one of the mostly used member of the toolbox and is a CButton object. Its mostly used event is the click event. You can open code window of the click event by double clicking the button or selecting the click event from the event list represented by  icon in the properties window. Codes written in this event will be executed when the button is clicked.

Properties of the button (setting caption, alignment etc.) can be changed from the properties list represented by  icon in the properties window. You can also disable/enable button using EnableWindow(bool) member function.

EditControl



EditControl (also known as EditBox in some other languages) is a member of the toolbox to get input from the user or to display output.

EditControl is a CEdit object.

In order to get the value entered in an EditControl; we have to assign a variable name to corresponding EditControl. We can assign a variable to an EditControl by right clicking and choosing *add variable*. Then, we can write a unique variable name to the *variable name* field.

The properties and event of the EditControl can be changed as is in Button.

In order to set the text of an EditControl from the code window we can use the SetWindowTextW(CString c) member function of the EditControl object.

In order to get the text of an EditControl from the code window we can use the GetWindowTextW(CString&c) member function of the EditControl object.

Static Text

A small icon representing a Static Text control, showing the word "Static" in a light blue box.

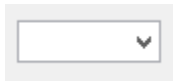
Static Text is a member of the toolbox to display output or to write a static text.

Static Text is a CStatic object.

In order to assign a variable name to a corresponding Static Text, we first have to change the ID property of the Static Text to a different one than its default value. We have to assign variable to Static Text if it is only used for display purpose. If we write a static string to Static Text, we can change it from the caption property of the properties list.

In order to set the text of a Static Text from the code window we can use the `SetWindowTextW(CString c)` member function of the Static Text object.

ComboBox



ComboBox can is a combination of dropdown list and and edit control, allowing the user to either type a value directly or select a value from the list. Transition among these several types can be made through changing type property of this object.

ComboBox is a CComboBox object.

A list of data can be added using the data property of this object by entering each line (data) by separating semicolon, e.g., `data1;data2;data3`.

The selected data can be obtained from the code using the `GetWindowTextW(CString&c)` member function of the CComboBox object.

In order for your program to automatically select one of the items by default and to display it, you must specify the index of that item in `OnInitDialog()` function. Assuming your project is called `Test`, your ComboBox is named `myBox` with items `data1;data2;data3`, and you want to initially select and display `data2` by default when the program starts, the following snippet would achieve what you want, where the relevant part is highlighted yellow:

```

BOOL TestDlg::OnInitDialog()
{
    ...

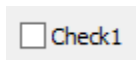
    // Set default value of ComboBox myBox to "data2" which has index 1
    myBox.SetCurSel(1);

    ...

    return TRUE; // return TRUE unless you set the focus to a control
}

```

CheckBox



Check Box are used when we only want to let the user to choose a one or more option from a list of options.

Check Box is a CButton object.

A Check Box can take two values either selected or not selected. Among a group of Check Box components, any number of them can be selected or non-selected (including none and all).

In order to find whether a Check Box is selected or not, we can use bool GetCheck() member function of the Check Box.

ListBox



A **List Box** is a toolbox member that allows the user to select one or more items from a list or display output in the list, multiple line text box.

List Box is a CListBox object.

In order to call some member functions of `ListBox`, we have to assign a variable name to corresponding `ListBox`. We can assign a variable to a `ListBox` by right-clicking and choosing *add variable*. Then, we can write a unique variable name to the *variable name* field.

We can use `AddString(CString c)` member function to add text to the list.

The properties and event of the `ListBox` can be changed as is in `Button`.

Vertical scrollbar can easily be enabled by setting the “Vertical Scroll” property to true.

Enabling horizontal scrollbar is a bit more complicated than enabling a vertical scrollbar. You need to write some code to extend horizontal limit of the `ListBox`. To achieve this, you first have to set the “Horizontal Scroll” property to true. Then, you have to write necessary codes to the event where you want to extend the horizontal limit of the `ListBox`. Below, you can find a code snippet written to click event of a `Button`. Lines which help us to extend horizontal limit are highlighted with yellow and comments are written aside.

```
int maxSize = 0; // You have to keep size of the string which have maximum length.

void Ccs204_gui2Dlg::OnBnClickedButton1()
{
    // TODO: Add your control notification handler code here

    CString name, surname, result;

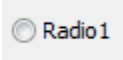
    //get a DC
    CDC *pDC = GetDC(); // This class is necessary to get the length of the string
    CSize cz; // An object which keeps the length of the string.

    input1.GetWindowTextW(name);
    input2.GetWindowTextW(surname);

    result = _T("Good Morning ") + name + _T(" ") + surname;
    list.AddString(result);
    cz = pDC->GetTextExtent(result); // Returns the size of the string to cz object
    if(cz.cx > maxSize) // cz.cx keeps the horizontal length of the string. We compare
        maxSize = cz.cx; // it with maxSize and update the maxSize if the condition
    // is satisfied

    list.SetHorizontalExtent(maxSize); // Set the Horizontal scrollbar length to
    //maxSize.
}
```

RadioButton



Radio Buttons are used when we only want to let the user choose one option from a list of options.

Radio Button is a CButton object.

A Radio Button can take two values either selected or not selected.

In order to create a list of options using Radio Button, we have to group the Radio Buttons in the list. There can be only one selected Radio Button in a group. To create a group of Radio Buttons, we have to set the *group* property of the first (only the first) Radio Button true.

In order to be able to decide which radio button object is chosen, we should do the following: First we have to set the *group* property of the first Radio Button true. Then, only for the first radio button object, click add variable, check the Control Variable check box, set Category to value, give a variable name and set the variable type as int. After setting these properties, you can decide which radio button is chosen in the code by using the given variable name. This variable takes value of 0 if the first radio button is chosen, takes value of 1 if the second radio button is chosen and so on. If no radio button is selected, then this value is -1. Prior to the value control, you should also call `UpdateData(TRUE);` function to get the most recent values of the variables. Using a button group is also useful for selecting a default radio button because normally none is selected. Assuming you have added the abovementioned control variable as int, and its name is `first`, pass the index of the radio button that should initially be selected as value into the constructor of your dialog. In the sample code snippet below your project is called `Test` and the default radio button that we want to select is the one with index 0. The important part that initializes the variable is marked with yellow:

```
TestDlg::TestDlg(CWnd* pParent): CDialogEx(Test::IDD, pParent)
    , first(0)
{
    ...// Here might be some more code
}
```


Enabling/Disabling a MFC Object

In some cases, we may want to disable an object and then enable it as the program executes. When an object is disabled, it is shown grayed out and no operation can be done on it.

When an object is first created, by default it is enabled. In order to disable it initially when the program first starts, you have to set the object's property called `Disabled` to `True`.

To enable/disable an object dynamically in the program, you have to write some more code. Suppose for example, you want to employ a **CheckBox** in order to enable/disable a MFC object (ListBox, ComboBox, EditControl, etc.). More specifically, whenever the CheckBox, say it is called `cb`, is clicked, you must enable/disable the MFC object in `cb`'s `onBnClicked()` function, which is called whenever `cb` is clicked by the user. The relevant code to this problem is marked with yellow again, assuming your project is called "Test" and the MFC Object has the name `obj`:

```
void TestDlg::OnBnClickedcb()
{
    // If user clicked on cb and it was previous not selected
    if (cb.GetCheck())
    {
        // Enable the MFC object, so that the user can interact with it
        obj.EnableWindow(TRUE);
    }
    else
    {
        // Disable the MFC Object
        obj.EnableWindow(FALSE);
    }
}
```

Some useful functions to convert well-known data types to and from CString

- **Convert int to CString**
`myCString.Format(_T("%d"), myInt);`
Value of the integer variable `myInt` will be converted into CString and will be stored in the CString variable `myCString`.
- **Convert CString to int:**
`int myInt = _ttoi(myCString);`
Value of the CString variable `myCString` will be converted into integer and will be stored in the integer variable `myInt`.
- **Convert unsigned int to CString**
`myCString.Format(_T("%u"), myUnsignedInt);`

Value of the unsigned integer variable `myUnsignedInt` will be converted into `CString` and will be stored in the `CString` variable `myCString`.

- **Convert `CString` to unsigned int**

```
unsigned int myUnsignedInt = _tcstoul(myCString, nullptr, 10);
```

Value of the `CString` variable `myCString` will be converted into unsigned integer and will be stored in the unsigned integer variable `myUnsignedInt`. In this function, the second parameter is for end pointer and the third parameter is the base (radix) for the integer value. You can use the second and third parameters of the function as given above for decimal numbers.

- **Convert `CString` to double**

```
double myDouble = _ttof(myCString);
```

Value of the `CString` variable `myCString` will be converted into double and will be stored in the double variable `myDouble`.

Moreover, you can process a `CString` object's individual characters using `[]` operator one by one (as in the regular strings). In order to get the length of a `CString` object you can use `CString`'s `GetLength()` member function. For example, if `myCString` is a `CString` object, the following partial loop checks individual characters of `myCString`.

```
for (int i=0; i < myCString.GetLength(); i++)
{
    if (myCString[i] == ..... )
        .....
}
```