Borland C++ Builder Strings

The Fundamentals of Strings

String Construction and Declaration

String operations in C++ Builder are mainly performed using a class called AnsiString. The **AnsiString** class is not derived from **TObject**. Therefore, it has a high level of independence and flexibility from the application or the control that wants to use it.

It is simply incredible the level of work and support provided by the VCL to its strings and text-related operations. Almost any possible operation that can be performed on a string is supported. There are so many functions that we will review only those that are most used in this book but all functions that were created in the libraries are highly valuable and can save you a tremendous amount of code writing and headache.

Many controls use AnsiString properties. All controls that use a caption (forms, panels, labels, etc) have their Caption property set as an AnsiString value. Many other controls such as the edit box use the AnsiString class as the basis of their text. Based on these two facts, you have already used and implemented AnsiString values. In some other scenarios you will need to declare and possibly initialize a string before using it.

To declare a string, use the AnsiString word followed by a valid C++ name. Here is an example:

```
AnsiString Country;
```

Since AnsiString is a class with its own constructor, you can also declare its variable with empty parentheses, which would be calling the class' constructor. Here is an example:

```
AnsiString City();
```

String Initialization

There are two main ways you can initialize an AnsiString variable. After declaring it, you can assign the desired value to the variable using the assignment operator. Here is an example:

```
AnsiString Country;
Country = “Madagascar”;
```

You can also initialize the string variable when declaring it, again, using the assignment operator and providing the desired value. Here is an example:

```
AnsiString Province("British Columbia");
```

Once you have defined the string you can use it as you see fit. You can use it to change a control's caption:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        AnsiString Country;
        Country = "Madagascar";
        Panel1->Caption = Country;
}
//---------------------------------------------------------------------------
```

You can also use it to fill out an edit control:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        AnsiString City = "Antananrivo";
        Edit1->Text = City;
}
//---------------------------------------------------------------------------
```

General Purpose String Functions

String Emptiness

General purpose functions are those that perform on strings regardless of any other consideration. For example, before performing any operation on a string, sometimes you will need first to find out whether the string contains something or is empty. Eventually, you will decide what to do if the string is empty.

There are two main ways you can check whether the content of a text-based control is empty. You can just use the AnsiString "=="overloaded operator. Here is an example:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        AnsiString Original = Edit1->Text;

        if(Original == "")
                Edit2->Text = "Why is Edit1 empty?";
}
//---------------------------------------------------------------------------
```

The **AnsiString** class provides its own function used to check whether a string is empty. Its syntax is:

```
bool __fastcall IsEmpty() const;
```

This member function can be used to check whether a text-based control contains nothing but it cannot empty a text control. The following example shows two ways of using the **AnsiString::IsEmpty()** method:

```
//---------------------------------------------------------------------------
void __fastcall TOKBottomDlg::OKBtnClick(TObject *Sender)
{
        String UserName = edtUserName->Text;

        if( Edit1.IsEmpty() )
                Panel1->Caption = "Please provide a User Name";
        if( Edit2->Text == "" )
                Panel1->Caption = "You need to type a password";
        if( Edit3->Text.IsEmpty() )
                Panel1->Caption = "Your account is not complete";

        edtUserName->SetFocus();
}
//---------------------------------------------------------------------------
```

The Length of a String

When a string has been initialized or at least is not empty, it has a length. There are various ways you can get or control the length of a string.

To find the length of a string, if the string is from the C string class, you can first convert it using the **AnsiString::c_str()** function, then use the strlen() function to get its length:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        char* S = Edit1->Text.c_str();
        Edit2->Text = strlen(S);
}
//---------------------------------------------------------------------------
```

To get the length of an AnsiString variable, use the **AnsiString::Length()** method. Its syntax is:

```
int __fastcall Length() const;
```

This function returns the length of the string as an integer. Here is an example:

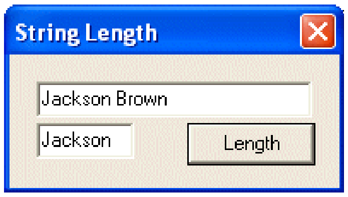
```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        AnsiString S = Edit1->Text;
        Edit2->Text = S.Length();
}
//---------------------------------------------------------------------------
```

The AnsiString class allows you to impose the number of characters of a string. It uses the following method:

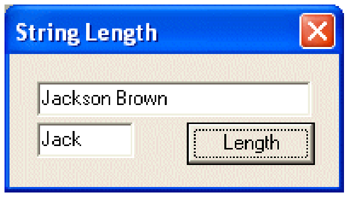
```
AnsiString& __fastcall SetLength(int NewLength);
```

This method takes one argument which is the length you want the AnsiString variable to have. If the *NewLength* argument is less than the length of the string, the resulting string would be the first *NewLength* characters of the original string. If the *NewLength* value is less than the length of the string, the original would be preserved and assigned to the resulting string:

```
//-----------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    AnsiString S = Edit1->Text;
    Edit2->Text = S.SetLength(7);
}
//-----------------------------------------------------
```



```
//-----------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    AnsiString S = Edit1->Text;
    Edit2->Text = S.SetLength(4);
}
//-----------------------------------------------------
```



String Trimming

Trimming a string is an operation that gets rid of leading or ending spaces in a string. To remove any (empty) space on the left side of a string, you can use the **AnsiString::TrimLeft()** method. Its syntax is:

```
AnsiString __fastcall TrimLeft() const;
```

If the original string has space on its left, this function would remove it and return a string that is like the original without the leading space. If the original does not have any leading space, the function would return the same string:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        Edit2->Text = Edit1->Text.TrimLeft();
}
//---------------------------------------------------------------------------
```

Another function used to perform the same operation is the TrimLeft(). Its syntax is:

```
AnsiString _fastcall TrimLeft(const AnsiString S);
```

As opposed to the **AnsiString::TrimLeft()** method, the (global) **TrimLeft()** function takes one argument which is the string that needs to be left trimmed. The function returns a new string that is the same as the original omitting the leading space (if any exists):

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        Edit2->Text = TrimLeft(Edit1->Text);
}
//---------------------------------------------------------------------------
```

To remove any space on the right side of a string, you can use the **AnsiString::TrimRight()** method. Its syntax is:

```
AnsiString __fastcall TrimRight() const;
```

If the original string has space on its right, this function would remove it and return the same string without any trailing space. Otherwise, the original string would be returned:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Edit2->Text = Edit1->Text.TrimRight();
}
//---------------------------------------------------------------------------
```

Another function used to perform the same operation is the **TrimRight()**. Its syntax is:

```
AnsiString _fastcall TrimRight(const AnsiString S);
```

The (global) **TrimRight()** function requires one argument as the string that needs to be trimmed. The function returns the original string without the trailing space:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Edit2->Text = TrimRight(Edit1->Text);
}
//---------------------------------------------------------------------------
```

Other functions allow you to combine the last two operations into one. You can use the **AnsiString::Trim()** method to remove spaces on both sides of a string. Its syntax is:

```
AnsiString __fastcall Trim() const;
```

Here is an example of using this method:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        Edit2->Text = Edit1->Text.Trim();
}
//---------------------------------------------------------------------------
```

Alternatively, you can use the global Trim() function to perform the same operation. Its syntax is:

```
AnsiString _fastcall Trim (const AnsiString S);
```

Here is an example:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        Edit2->Text = Trim(Edit1->Text);
}
//---------------------------------------------------------------------------
```

String Conversions

C/C++ Data Types Conversion to AnsiString

A value that the user types in a control such as an edit box is considered a string. This is because the compiler cannot assume the kind of value the user or the client of an Edit control would supply. For this reason, after a value has been provided to a control that uses the AnsiString as the basis of its content, if you want to perform any mathematical operation on the string you must convert the string to a valid data type.

The AnsiString class provides a lot of constructors that allow you to create a string of any kind. For example you can use it to declare:

- A character:
  AnsiString Symbol = 'H';
- An integer:
  AnsiString Int = 120;
- A long integer:
  AnsiString Longer = -73495745;
- A floating-point value:
  AnsiString WeeklyEarnings = 675.15;
- A double-precision number:
  AnsiString WeeklyEarnings = 675.15;
  AnsiString Silver = 2.15e28;
- A string:
  AnsiString GirlFriend = "Micheline Phoon";

Any of these variables can be declared using their equivalent constructors: AnsiString Symbol('H');

```
AnsiString Int(120);
AnsiString GirlFriend("Micheline Phoon");
AnsiString WeeklyEarnings(675.15);
AnsiString Longer(-73495745);
AnsiString Silver(2.15e28);
```

Based on the configurations of the AnsiString constructors, you can convert any value and make it available to a control that uses an AnsiString property. For example, you can convert and display:

- A character:
  char Sign = 'q';
  Edit1->Text = AnsiString(Sign);
- An interger:
  Integer Number = 808;
  Caption->Text = AnsiString(Number);
- A long integer:
  long Value = 497783L;
  Panel1->Caption = AnsiString(Value);
- A floating-point value:
  Float Distance = 1205.62;

Label1->Caption = AnsiString(Distance);

- A double-precision number:
  Double YearlyIncome = 24588;
  Edit1->Text = AnsiString(YearlyIncome);
- A string:
  AnsiString Food = "Peanut Butter";
  Button2->Caption = AnsiString(Food);

AnsiString and C-Strings

The AnsiString class is configured to recognize null-terminated strings of the classic C string functions. The AnsiString class has a constructor that can convert a null-terminated C string to AnsiString. Thanks to this constructor, the AnsiString(const char* Source), you can declare a C string and use it as you see fit:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        char Status[] = "Employee Status";
        Caption = Status;
        char *FullName = "Antoine Williams";
        Edit1->Text = FullName;
}
//---------------------------------------------------------------------------
```

Based on this constructor, you can declare a C string, manipulate its value and use it in your application. To convert an AnsiString value to a null-terminated string, use the **AnsiString::c_str()** method. The syntax of this function is:

```
char* __fastcall c_str() const;
```

This function is very valuable because it can help you perform various types of string operations.

Strings Cases: Lowercase to Uppercase Conversion

There are various techniques you can use to convert a string from lowercase to uppercase and vice-versa. An alphabetical character is recognized as being in lowercase if it is one of the following characters: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z. On the other hand, a character qualifies as uppercase if it is one of A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z. All the other symbols are ignored even if on the keyboard you would press Shift to type them.

To convert a lowercase character or string to uppercase, you can use the **AnsiString::UpperCase()** member function. Its syntax is:

```
AnsiString __fastcall UpperCase() const;
```

This member function considers an AnsiString variable and examines each one of its characters. If a character is an alphabetic character in lowercase, it would be converted to uppercase. If the character is either an alphabetical character in uppercase or it is not an alphabetic character, it would be kept “as is”. This method also considers the Regional Settings of the computer being used, as set in Control Panel.

If you want to convert a single character to uppercase, after initializing or getting, call this method. Here is an example:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        String S = 's';
        Edit1->Text = S.UpperCase();
}
//---------------------------------------------------------------------------
```

You can use this same method to convert a string to uppercase as follows:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        String S1 = "James N. Fame!";
        String S2 = S1.UpperCase();
        Edit1->Text = S2;
}
//---------------------------------------------------------------------------
```

Besides the AnsiString method, you can use the UpperCase() function to convert a character or string to uppercase. Its syntax is:

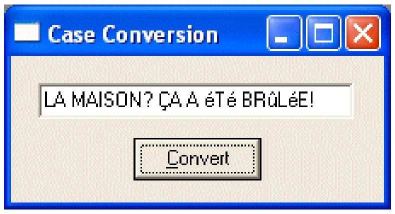
```
AnsiString __fastcall UpperCase(const AnsiString S);
```

This function uses the same algorithm as the AnsiString::UpperCase() method. Here is an example of using it:

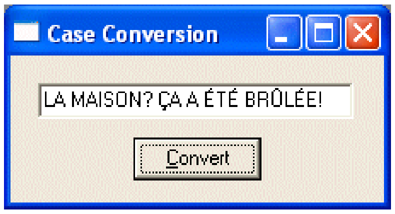
```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        String S1 = "James N. Fame!";
        String S2 = UpperCase(S1);
        Edit2->Text = S2;
}
//---------------------------------------------------------------------------
```

The AnsiUpperCase() function uses the same syntax as the UpperCase() function and applies the same algorithm as the AnsiString::UpperCase() method. Unlike the UpperCase() function, AnsiUpperCase() considers the Regional Settings of the computer being used. Look at how these two functions convert the same French string:

```
//---------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    String S1 = "La maison? Ça a été brûlée!";
    String S2 = UpperCase(S1); Edit1->Text = S2;
}
//---------------------------------------------------------------------
```



```
//---------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    String S1 = "La maison? Ça a été brûlée!";
    String S2 = AnsiUpperCase(S1);
    Edit1->Text = S2;
}
//---------------------------------------------------------------------
```



Strings Cases: Uppercase to Lowercase Conversion

You can use the **AnsiString::LowerCase()** method to convert an uppercase character or string to lowercase. Its syntax is:

```
AnsiString __fastcall LowerCase() const;
```

Using the Regional Settings, this function examines each character of the provided AnsiString variable. If a character is an alphabetic character in uppercase, it would be converted to lowercase. The case of all the other characters would be ignored.

If you want to convert a single character to uppercase, after initializing or getting, call this method. Here is an example:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::btnConvertClick(TObject *Sender)
{
        String String1 = "[Borland C++ Builder]";
        Edit1->Text = String1.LowerCase();
}
//---------------------------------------------------------------------------
```

You can also use the LowerCase() function to convert a character or string to lowercase. Its syntax is:

```
AnsiString __fastcall LowerCase(const AnsiString S);
```

This function uses the same algorithm as the **AnsiString::UpperCase()** method. Here is an example of using it:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::btnConvertClick(TObject *Sender)
{
        String String1 = "[Borland C++ Builder]";
        Edit1->Text = LowerCase(String1);
}
//---------------------------------------------------------------------------
```

If the local settings you are using or distributing your program to are a factor, you should use the **AnsiLowerCase()** function. It processes the string in the same way as the **AnsiString::UpperCase()** method but uses the same syntax as the **UpperCase()** function:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::btnConvertClick(TObject *Sender)
{
        String S1 = "La version Française de Borland C++ Builder est là. "
                    "Elle est arrivée!";
        edtConvert->Text = AnsiLowerCase(S1);
}
//---------------------------------------------------------------------------
```

Strings Addition

The Addition Operator

To add one AnsiString value to another, you use the addition operator (it was overloaded to respond appropriately). The operation would produce a new string that combines the first and the second string. Here is an example:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        Edit3->Text = Edit1->Text + Edit2->Text;
}
//---------------------------------------------------------------------------
```

You can then use the addition operation to add as many strings as you see fit.

Appending Strings

Appending two strings consists of adding one string to another string. This operation usually involves two strings: a destination and a source strings. To append two strings, besides the addition operator "+", you can use the **AppendStr()** member function. Its syntax is:

```
void __fastcall AppendStr(AnsiString &Destination, const AnsiString Source);
```

This function takes two arguments as strings. The source is added to the destination. The function returns the destination already changed by the operation. Here is an example:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        AnsiString Destination = "Paul ";
        AnsiString Source = "Lombard";
        AppendStr(Destination, Source);
        Edit1->Text = Destination;
}
//---------------------------------------------------------------------------
```

Strings Comparison Functions

Introduction

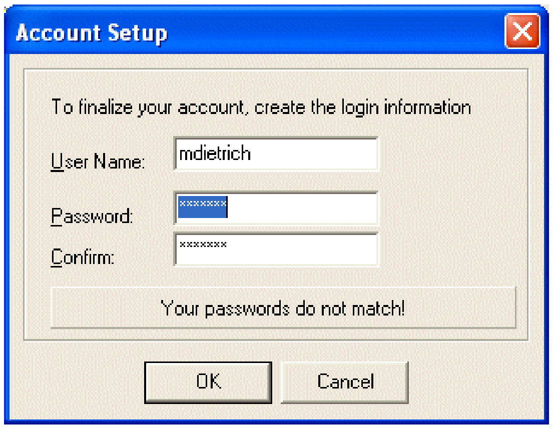
String comparison allows you to find out which one of two strings is longer or whether both strings are equal. When comparing two strings, the compiler sometimes checks lowercase and uppercase characters. Depending on the function you are using, you can ask the compiler to consider the case of the characters; this is referred to as case-sensitivity. Some of the functions, when performing their comparison on Dates or currency values, would refer to the Regional Settings of your computer as set in the Control Panel.

String Comparison With Case-Insensitivity

The **SameText()** function is used to compare two strings. Its syntax is:

```
bool __fastcall SameText(const AnsiString String1, const AnsiString String2);
```

The function requires two AnsiString variables and compares them. The comparison is performed without case-sensitivity. If the strings are the same, the result is true (the integer equivalent is 1). Otherwise the comparison yields false (0). You can use the SameText() function on a validation dialog like this one:


Account Setup
To finalize your account, create the login information
User Name:
mdietrich
Password:
Confirm:
Your passwords do not match!
OK
Cancel

Then you can implement the **OnClick()** event of the OK button as follows:

```
//---------------------------------------------------------------------------
void __fastcall TOKBottomDlg::OKBtnClick(TObject *Sender)
{
        String Password1 = edtPassword1->Text;
        String Password2 = edtPassword2->Text;
        Boolean Result = SameText(Password1, Password2);

        if(Result == False)
        {
                Panel1->Caption = "Your passwords do not match!";
                edtPassword1->SetFocus();
        }
        else
        {
                Panel1->Caption = "Your account has been setup.";
                Close();
        }
}
//---------------------------------------------------------------------------
```

The **AnsiString::AnsiCompareIC()** method performs a comparison of two strings, considering the Regional Settings. Like **SameText()**, this function does not care about case-sensitivity

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        AnsiString String1 = Edit1->Text;
        AnsiString String2 = Edit2->Text;

        if(String1.AnsiCompareIC(String2) < 0)
                Edit3->Text = "True";
        else if(String1.AnsiCompareIC(String2) > 0)
                Edit3->Text = "False";
        else
                Edit3->Text = "Equal";
}
//---------------------------------------------------------------------------
```

Alternatively, you can use the **CompareText()** function to compare strings. Unlike the **AnsiString::AnsiCompareIC()** method, this function does not care about the Windows Regional Settings. The syntax of this function is:

```
int __fastcall CompareText(const AnsiString First, const AnsiString Second);
```

The function takes two string arguments and examines their characters incrementally. After the comparison, the function returns:

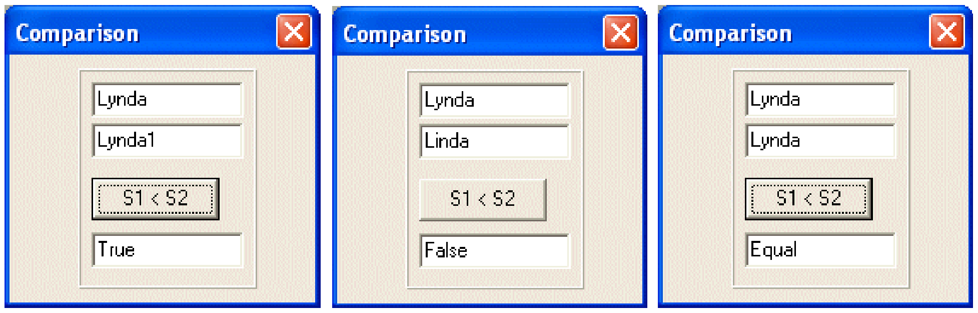
- a negative value if the First string is less than the Second
- a positive value if the First string is greater than the Second
- 0 if both strings are the same

String Comparison With Case-Sensitivity

The **AnsiString::AnsiCompare()** method is used to compare two strings with regards to case sensitivity. This function, when performed on dates and currency values, considers the Regional Settings of the user's computer. Here is an example:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        AnsiString String1 = Edit1->Text;
        AnsiString String2 = Edit2->Text;

        if(String1.AnsiCompare(String2) < 0)
                Edit3->Text = "True";
        else if(String1.AnsiCompare(String2) > 0)
                Edit3->Text = "False";
        else
                Edit3->Text = "Equal";
}
//---------------------------------------------------------------------------
```



Lynda
Lynda1
True

Lynda
Linda
False

Lynda
Lynda
Equal

Besides the **AnsiString::AnsiCompare()** method you can use the **AnsiCompareStr()** function to compare strings. Like the **AnsiString::AnsiCompare()** method, this function takes into consideration the Windows Regional Settings. Its syntax is:

```
int __fastcall AnsiCompare(const AnsiString& OtherString) const;
```

The function considers its own string and compares it to the string argument it takes. This function returns:

- a negative value if your string is less than the *OtherString*
- a positive value if your string is greater than the *OtherString*
- 0 if both strings are the same

To compare strings, you can also use the **CompareStr()** function. Unlike the **AnsiString::AnsiCompare()** method, this function does not care about the Windows Regional Settings. The syntax of this function is:

```
int __fastcall CompareStr(const AnsiString First, const AnsiString Second);
```

The function takes two string arguments and examines their characters incrementally. After the comparison, the function returns:

- A negative value if the First string is less than the Second
- A positive value if the First string is greater than the Second
- 0 if both strings are the same

Strings Boolean Comparisons

The **AnsiString** class and the **sysutils** library provide techniques of comparing strings. The functions we have used to perform comparisons returned integral values at the end of their comparisons. Sometimes, when performing specific algorithms, such as comparing passwords, performing mathematical calculations, performing spell checks in text documents, etc, you will only need to know whether two strings are equal. This type of comparison renders a Boolean value of true or false. Both libraries can perform any sort of comparison.

When you have two strings and would like to find out whether both are equal, you can use the (overloaded) == operator. If both strings are equal, the conditional comparison would return true.

You can also use the AnsiSameStr() function. Its syntax is:

```
bool __fastcall AnsiSameStr(const AnsiString First, const AnsiString Second);
```

The function takes the Windows Regional Settings into consideration when comparing the First and the Second strings with case-sensitivity. If both strings are the same, the function return true. If they are not the same, the result is false. Here is an example:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::btnSendClick(TObject *Sender)
{
        String EMail1 = edtEMail1->Text;
        String EMail2 = edtEMail2->Text;

        if(AnsiSameStr(EMail1, EMail2))
        {
                frmCongratulations->ShowModal();
                Close();
        }
}
//---------------------------------------------------------------------------
```

Alternatively, to compare two strings, you can use the **AnsiSameText()** function. Both functions use the same syntax. Like the **AnsiSameStr()** function, the **AnsiSameText()** considers the Regional Settings. Unlike the **AnsiSameStr()** function, the **AnsiSameText()** function does not consider the case of the characters in the strings.

If the strings you want to compare are captions, such as those you see on a form, it would be cumbersome to write a comparison function that would examine them. This is because the captions on a form usually have an ampersand used to underline one of their characters. Examples include First Name, Address, City, Full Name or Department. Fortunately, Borland provides the **AnsiSameCaption()** function. Its syntax is:

```
bool __fastcall AnsiSameCaption(const AnsiString First, const AnsiString Second);
```

This function takes two captions and compares them considering the Regional Settings . Regardless of where the ampersand is positioned, the other characters of the captions would be examined. If both captions are the same, the function would return true. In the following example, two captions are set as First Name and First Name respectively. A regular comparison would find them different, but the **AnsiSameCaption()** function finds that both strings are the same:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
        lblCaption1->Caption = "&First Name";
        lblCaption2->Caption = "First &Name";
}
//---------------------------------------------------------------------------
void __fastcall TForm1::btnCompareCaptionsClick(TObject *Sender)
{
        String Caption1 = lblCaption1->Caption;
        String Caption2 = lblCaption2->Caption;

        if(AnsiSameCaption(Caption1, Caption2))
                Panel1->Caption = "Same captions";
        else
                Panel1->Caption = "Completely Different";
}
//---------------------------------------------------------------------------
```

Besides all available comparison methods of the AnsiString class and comparison functions throughout the VCL, you can use the Boolean operators to perform any type of comparisons on strings. These operators can be used the same way you would proceed with regular numeric variables. Therefore, the operators are:

- Equal: ==
- Not Equal: !=
- Less Than <
- Less Than Or Equal To <=
- Greater Than >
- Greater Than Or Equal To >=

Characters and Sub-Strings

The Last Character of a String

There are many operations you can perform on individual characters of an AnsiString variable. These include checking for a character, finding the position of a character, or deleting a character or characters. These operations can be valuable when creating objects such as login dialog boxes.

To find the last character of a string, use the **AnsiString::AnsiLastChar()** method. Its syntax is:

```
char* __fastcall AnsiLastChar() const;
```

You can use this method to find out the last character of a given string. In the following example, the last character of the string in the Edit1 edit box displays in the Edit2 edit box when the user clicks the Button1 control:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        String S = Edit1->Text;
        Edit2->Text = S.AnsiLastChar();
}
//---------------------------------------------------------------------------
```

If the **AnsiString** is used on a console application, you can use this same method to find the last character of an **AnsiString** variable.

Character Deletion

Sometimes you will want to get rid of a character in a string. This is done using the **AnsiString::Delete()** method. The syntax is:

```
AnsiString& __fastcall Delete(int Index, int Count);
```

This member function takes two integer arguments. The first argument specifies the position where the compiler would start considering the deletion. The second argument specifies the number of characters that should be deleted from the AnsiString variable.

If you declare an AnsiString variable called Country. You can use **Country.Delete(14, 11)** to delete 11 characters starting at the 14th character:

```
AnsiString Country("United States of America");
AnsiString After;
puts(Country.c_str());
After = Country.Delete(14, 11);
puts(After.c_str());
```

Sub-String Creation

A sub string is a string that is created from, or is included in, another string. C++ and C++ Builder allow you to find a sub string in an original string, to get the position of a sub string in a string, etc.

With a string, you can create a new string retrieved from the original using the AnsiString::SubString() method. Its syntax is:

```
AnsiString __fastcall SubString(int StartPosition, int HowManyChars) const;
```

This method takes two integer arguments. From the original string, the first argument specifies the position of the character where the new string would start. The second argument specifies the number of characters that would be considered when creating the new string. Here is an example:

```
//-----------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        AnsiString Original = Edit1->Text;
        AnsiString SubOne = Original.SubString(9, 4);
        Edit2->Text = SubOne;
}
//-----------------------------------------------------
```

The Position of a Sub String

The AnsiString class allows you to analyze a string and find out whether it contains a certain sub string. If it does, you can get the position of the substring, using the AnsiString::Pos() method. Its syntax is:

```
int __fastcall Pos(const AnsiString& SubString) const;
```

In many cases, you can also use the AnsiString::AnsiPos() method). Its syntax is:

```
extern PACKAGE int __fastcall AnsiPos(const AnsiString Substr, const AnsiString S);
```

Character or Sub-String Replacement

When performing your algorithms or other specific types of operations on strings, you may want to find out whether a certain character or group of characters has been provided in a string. If so, you may want to replace it with a different character or with a new sub string. This operation is handled by the **StringReplace()** function. Its syntax is:

```
AnsiString __fastcall StringReplace(const AnsiString Original,
                                    const AnsiString LookFor,
                                    const AnsiString ReplaceWith,
                                    TReplaceFlags FlagToUse);
```

The **StringReplace()** function will look for the *LookFor* character or sub-string in the Original string. If it finds it, then it will replace the LookFor character or sub string with the *ReplaceWith* character or sub-string. You control how this operation is performed by using the *FlagToUse* argument. The values you can use are to replace all occurrences of *LookFor* with *ReplaceWith*. The flag used would be *rfReplaceAll*. You can also ask the compiler not to take the character(s) case into consideration, which is done with *rfIgnoreCase*. Once the *TReplaceFlags* argument is a set, you can use one or both of the flags. Here is an example:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        Edit1->Text = StringReplace(Edit1->Text, " ", "",
        TReplaceFlags() << rfReplaceAll);
}
//---------------------------------------------------------------------------
```

String Quotations

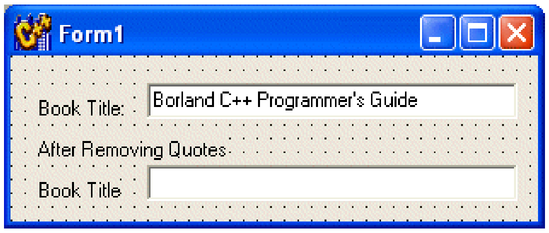
Regular String to Quoted String Conversion

In the strict of string routines, a quote is a character or symbol used to delimit a string. It sets the beginning and end of a string. In the English language, a quote is represented with the double-quote symbol ". The VCL is equipped with functions used to insert or remove quotes from a string.

The **AnsiQuotedStr()** function is used to "convert" a string into a quoted string. Its syntax is:

```
AnsiString __fastcall AnsiQuotedStr(const AnsiString Source, char Quote);
```

This function takes one string, the Source argument, and returns it added the Quote characters on both sides of the string. Here is an example:



```
//---------------------------------------------------------------------------
void __fastcall TForm1::edtQuotedExit(TObject *Sender)
{
        char *BookTitle = edtQuoted->Text.c_str();
        char Quote = '"';
        AnsiString Quoted = AnsiQuotedStr(BookTitle, Quote);
        edtBookTitle->Text = Quoted;
}
//---------------------------------------------------------------------------
```

Quoted String to Regular String Conversion

The **QuotedStr()** function is used to add a single-quote on both sides of a string. Its syntax is:

```
AnsiString __fastcall QuotedStr(const AnsiString S);
```

This function takes one string and returns it after adding a single-quote on both sides. Here is an example:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::edtQuotedExit(TObject *Sender)
{
        char *BookTitle = edtQuoted->Text.c_str();
        AnsiString Quoted = QuotedStr(BookTitle);
        edtBookTitle->Text = Quoted;
}
//---------------------------------------------------------------------------
```

String Quotes Removal

When a string is provided with quotes and you want to remove the quotes, use the AnsiExtractQuotedStr() function. Its syntax is:

```
AnsiString __fastcall AnsiExtractQuotedStr(char * &Source, char Quote);
```

This function takes two arguments. The Source parameter is a null-terminated string that is returned as an AnsiString value. When using the function, you must specify what character or symbol is used as Quote. Here is an example:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::edtQuotedExit(TObject *Sender)
{
        char *BookTitle = edtQuoted->Text.c_str();
        char Quote = '"';
        AnsiString RemoveQuotes = AnsiExtractQuotedStr(BookTitle, Quote);
        edtBookTitle->Text = RemoveQuotes;
}
//---------------------------------------------------------------------------
```

Home