Examples

Opening the text in a FolderItem into a **TextArea**.

```
Dim f as FolderItem
f=GetOpenFolderItem(FileTypes1.Text)

If f <> Nil Then
   If Not TextArea1.Open(f) Then
      MsgBox "Open failed"
   End if
End if
```

The File Type Set defines the common file type, Text, and the GetOpenFolderItem call lets the user open the desired text document. The Open method returns a Boolean that is True if the Open operation was successful.

Saving the text in a **TextArea** to a FolderItem:

```
Dim f as FolderItem
f=GetSaveFolderItem(FileTypes1.Text,"Untitled")

if f<> Nil Then
  If Not TextArea1.Save(f,False) Then
    MsgBox "Save failed"
  End if
End if
```

Saving is largely the mirror image of opening a text file. GetSaveFolderItem gets a FolderItem that's used to save the contents of the TextArea. The Save method then does the save, returning True if the save was successful. If you pass True as the second parameter of Save, then the styled text information will be saved.

See Also

Font, FontCount functions; Paragraph, Range, StyleRun, StyledText, StyledTextPrinter, TextEdit classes.

TextField Control

The standard editable text field. A **TextField** control can contain one line of text, with one font, font size, and style. The TextArea control can contain multiple lines of text and display multiple fonts and styles.

Super Class

TextEdit

TextField inherits from TextEdit, the base class for both TextArea and **TextField**. TextEdit is an abstract class and it is not intended to be instantiated. TextArea is the multiline and styled text control. **TextField** is the single-line text control.

Because this is a RectControl, see the RectControl for other properties and events that are common to all RectControl objects.

Properties

Name	Type	Description
CueText	String	Use to display a cue or suggested value in the field. The CueText appears in the **TextField** in grey text. CueText is currently supported on Windows and Linux. It will be available in Macintosh Cocoa builds.
LimitText	Integer	The maximum number of characters allowed in the **TextField**. The value of zero does not limit text. LimitText works for normal text entry, copy and paste, and drag and drop.
Password	Boolean	If True, bullets appear in field instead of the characters the user typed. The Cut and Copy Edit menu items are automatically disabled. On Mac OS X 10.3 and above, the Password property enables and disables secure input.
ReadOnly	Boolean	If True, the text cannot be modified.
SelAlignment	Integer	Controls paragraph alignment of the selected text. The class constants and values are shown below: AlignDefault(0): Default alignment AlignLeft(1): Left aligned AlignCenter (2): Centered AlignRight (3): Right aligned -1: Mixed—The selection spans multiple paragraphs with different alignments. Currently, the Default alignment is the same as Left alignment.
SelPlain	Boolean	If True, the selected text is plain.
SelTextColor	Color	Used to get and set the text color of the **TextField**'s text.
SelTextFont	String	Used to get and set the text font of the **TextField**'s text.
SelTextSize	Integer	Used to get and set the font size of the **TextField**'s text.

Events

Name	Parameters	Return Type	Description
GotFocus			The cursor has moved into the **TextField**.
LostFocus			The cursor has left the **TextField**.

Name	Parameters	Return Type	Description
MouseDown	x as Integer, y as Integer	Boolean	The mouse button was pressed inside the **TextField**'s region at the location passed in to x,y. Return True if you are going to handle the MouseDown. Returning True prevents the **TextField** from handling the mouse click.
MouseUp	x as Integer, y as Integer		The mouse button was released inside the control's region at the location passed in to x,y. This event will not occur unless you return True in the MouseDown event.
SelChange			The range of characters highlighted has changed.
TextChange			The Text property of the **TextField** has been changed.
ValidationError	InvalidText as String, StartPosition as Integer		The user has tried to enter a character that is prohibited by the **TextField's** Mask property. *InvalidText* is the entire character string up to and including the invalid text. *StartPosition* is the starting character position in which the actual invalid text was entered. The first character is numbered 1. If no code is provided in this event and a validation error occurs, REAL Studio plays the system beep.

Methods

Name	Parameters	Return Type	Description
Copy			Copies the selected text in the **TextField** to the Clipboard, including the styled text information.
Paste			Pastes the styled or unstyled text on the Clipboard into the **TextField** at the insertion point, adding the text to the existing text.
SelectAll			Selects all of the text in the **TextField**.
SetFocus			Gives the **TextField** the focus, sending all keydown events to the **TextField** and moving the cursor there.

Class Constants

The following class constants can be used to specify the value of the Alignment property.

Class Constants	Description
AlignDefault	Default alignment, currently the same as Left alignment.
AlignLeft	Left alignment.
AlignCenter	Center alignment.
AlignRight	Right alignment.

Notes

Execution order of MenuHandlers

The intrinsic control menu handlers (such as TextField.SelectAll) are handled after any user-defined menu handlers on the TextField subclass (if it was subclassed). This means that if you have a SelectAll handler on the Window of the TextField, it will no longer be called when the TextField has focus, because the TextField will now handle it first. In this situation, create a TextField subclass that defines its own SelectAll handler, and handle the desired behavior there.

Masks

Use the Mask property to filter user input on a character-by-character basis and add formatting characters. For example, a mask for a Telephone number field can add parentheses, spaces, and dashes as literals, that are used for formatting, and the digit mask symbol '#' to restrict entry to numbers only.

The following table shows the characters that you can use to define a mask.

Mask Character	Description
#	The single digit placeholder. The user can type only a digit (numeric) character in this position. For example, the mask "(###) ###-####" accepts the entry 5551212121" and returns "(555) 121-2121".
.	Decimal placeholder. The decimal placeholder that is actually used is specified in the user's International settings. The character is treated as a literal (formatting) character for masking purposes. For example, the mask "##.##" accepts the entry "2344" and returns "23.44" (for US systems).
,	Thousands separator. The thousands separator that is actually used is specified in the user's International settings. The character is treated as a literal (formatting) character for masking purposes. For example, the mask "####,###" accepts the entry "123456" and returns "123,356".
:	Time separator. The time separator that is actually used is specified in the user's International settings. The character is treated as a literal (formatting) character for masking purposes.
/	Date separator. The date separator that is actually used is specified in the user's International settings. The character is treated as a literal (formatting) character for masking purposes. For example, the mask "99/99/\2099" accepts the entry "123109" and returns "12/31/2009". The "\20" enters the default century and decade and only accepts the year in the first decade of the century.

Mask Character	Description
\	Mask escape character. Treats the next character in the mask as a literal. The escape character enables you to use the '#', '&', 'A', '?' (and so on) characters in the mask. The escapted character is treated as a literal (formatting) character. For example, the mask "\C\C-9999" accepts the entry "1234" and returns "CC-1234".
&	Character or space placeholder. It accepts one character. Valid values are the ASCII characters 32-126 and the non-ASCII characters 128-255. For example, the mask "&&-99999" accepts "li20520" and returns "li-20520".
C	Character or space placeholder, where entry is optional. It operates like the '&' placeholder. For example, the mask "CCCC-CC" formats "1233ed" as "1233-ed".
>	Convert all the characters that follow to uppercase. Uppercasing works beyond the ASCII range where appropriate, e.g., ü becomes Ü. For example, the mask ">&&-#####" accepts the string "li20520" and returns "LI-20520".
<	Convert all the characters that follow to lowercase. Lowercasing works beyond the ASCII range where appropriate, e.g., Ü becomes ü.
A	Alphanumeric character placeholder, where entry is mandatory. For example, the spec "AAA" specifies three alphanumeric characters.
a	Alphanumeric character placeholder, where entry is optional.
0	The literal "0" (zero). For example, the mask "99.00" formats the entry "22" as "22.00". The mask "\C\C0-9999" accepts the entry "1234" and returns it as "CC0-1234". The mask "##,###.00" accepts the entry "12345" and returns "12,345.00". The mask "99.00" accepts "21" and returns "21.00".
9	A Single (numeric) digit.
?	Alphabetic placeholder. Entry is optional. For example, the mask "???" accepts three alphabetic characters. It rejects numeric characters.
Any literal	All other symbols are displayed as literals for formatting purposes. For example, the mask "99[9]" accepts the entry "333" and returns "33[3]".
~	Reserved for future use. If you use "~" it will trigger an exception error. Use \~ instead.

Here are some examples

:

Mask	Description
999,999.99	Formats 2222222223 as "2222,222.23" (Using the US thousands separator.
99.00	Formats "22" as "22.00."

Mask	Description
###-##-####	//US Social Security number. Fomats “111111111” as 111-11-1111.

```
TextField1.Mask="###-##-####"
TextField1.Mask="(###) ###-####"  //US Phone number, with area code
```

If the user tries to enter a character that is prohibited by the mask, a ValidationError event occurs. The character that the user attempted to enter and the character position is passed to the ValidationError event, where you can handle the keystroke as you like.

To cancel the Mask, set it to the empty string:

```
TextField1.Mask=""
```

Adding Text to a TextField

When appending text to a **TextField**, you may notice some flicker as REAL Studio redraws the **TextField** to show the new text. This will happen if you appended the Text property of the **TextField** like this:

```
TextField1.Text=TextField1.Text+"my new text"
```

This occurs because the entire contents of the TextField has to be redrawn. To avoid this flicker, call the AppendText method instead. Simply pass it the text to be appended. For example, this code reads an external text file into a TextField using the Read method of the Readable class interface. The text is read in groups of 255 characters until the end-of-file is reached.

```
Dim f as FolderItem
Dim i as integer
Dim stream as BinaryStream
f=GetOpenFolderItem(FileTypes1.Text) //file type defined in File type set
If f<> Nil Then
 stream=BinaryStream.Open(f,False)
 Do
  TextField1.AppendText  stream.Read(255,Encodings.WindowsANSI)
 Loop Until stream.EOF
 stream.Close
End if
```

Text Encoding

TextFields store all text internally in Unicode, which is able to represent a mixture of characters from different writing systems. When you extract the text via the Text or SelText properties, this text is returned in UTF-8.

Using Class Constants

The following class constants are available to set paragraph alignments. Set the alignment of the entire contents of the TextField by assigning a constant to the Alignment property.

Value	Class Constant
0	AlignDefault
1	AlignLeft
2	AlignCenter
3	AlignRight

For example, the following code in the Action event of a control array sets the alignment of the text in a **TextField**. The Action event is passed an index parameter that indicates which control was clicked.

```
Sub Action (Index as Integer)
 Select Case Index
 Case 0
  TextField1.Alignment=TextField.AlignDefault
 Case 1
  TextField1.Alignment=TextField.AlignLeft
 Case 2
  TextField1.Alignment=TextField.AlignCenter
 Case 3
  TexField1.Alignment=TextField.AlignRight
 End Select
```

See Also

Font, FontCount functions; Paragraph, Range, StyleRun, StyledText, StyledTextPrinter, TextEdit classes.

TextInputStream Class

In order to read text from a file, you need to create a **TextInputStream** object. **TextInputStreams** have methods that allow to read from a file, check to see if you are at the end of the file, and close the file when you are done reading from it. They are created by calling the Open shared method. If you are working with a file with an encoding that is not UTF-8, you should set the value of the Encoding property.

On Windows and Macintosh only, **TextInputStream** objects can work with files larger than 2 gigabytes.

Super Class

Object