Aivosto › Articles › Optimize strings III

Optimize string handling in VB6 - Part III

Processing strings with Visual Basic 6.0 can get considerably faster if you know the tricks. Part III of this article studies the performance of Left$, Mid$ and Right$ in detail. We learn to quickly examine individual characters with Asc and AscW. We check out the differences between Asc and AscW and also Chr and ChrW. We also see how a badly placed pair of extra parentheses can degrade performance.

Part I | Part II | Part III

Part III of this article goes deep in the specifics of string processing in Visual Basic 6. Read the previous parts to get an understanding of the basics of optimization and how strings work in VB6.

In this article:

Related articles
How not to optimize in Visual Basic
Optimize loops
Restructuring Visual Basic code
Save memory
Toolbox for project manager
VB InStr
VB tips — Optimize for memory and speed

VB6 functions in this article: Asc, AscW, Chr$, ChrW$, Left$, Mid$, Right$.

Left$, Mid$ and Right$

The functions Left$, Mid$ and Right$ are essential in string processing. If you've read Part I of this article, you already know that the string versions ($) of these functions run faster than the variant versions (without $). Since these functions are so important, let's take a deeper look into the various functions, their parameters and what exactly it is in these functions that may slow down your apps.

These functions return a partial copy of the input string (s). The input gets copied to the output in whole or in part. Usually in part.

Left$(s, n) returns n characters from the start of s.
Mid$(s, x) returns the rest of s starting with position x.
Mid$(s, x, n) returns n characters of s starting with position x.
Right$(s, n) returns n characters from the end of the s.

Note: If the string is too short, the output will be shorter than n.

In this article we will use the following variables:

s = input string
n = length of output string (number of characters)
x = start position for Mid$

Avoid a useless full copy

Quite obviously, if Left$, Mid$ or Right$ return a full copy of the whole of s, there is no point calling them at all. You can as well use s. The following calls are useless:

Left$(s, n) when n ≥ Len(s)
Mid$(s, 1) always
Mid$(s, 1, n) when n ≥ Len(s)
Right$(s, n) when n ≥ Len(s)

If there is a risk for such calls, test for n < Len(s) before calling these functions.

Performance of Left$, Mid$ and Right$

Now, what are the essential factors affecting the performance of Left$, Mid$ and Right$? Is it the size of input (s), the size of output (n), the position of output (x) or the choice of the function (Left$, Mid$ or Right$)?

To test this, we created a small VB6 program that executed each of Left$, Mid$ and Right$ 5 million times. As input s we used strings of 1, 10, 100, 1000 and 10000 characters. As output sizes, we used n = 1, 10, 100, 1000 and 10000, but no longer than length of s. In addition, we tested whether there is any difference in Mid$ from start-of-string compared to end-of-string.

Performance test results: Left$, Mid$ and Right$

1. Output size (n) dictates speed. The longer the returned string, the slower the functions run.
2. High n equals slow performance.
3. Input size (length of s) has no effect.
4. Middle parameter to Mid$(.., x, ..) has no significant effect when output size is the same.
5. Left$ and Right$ run faster than Mid$. The difference is only significant with small output sizes (1, 10 and 100). With large output (1000 and 10000) the speed difference is negligible.
6. Mid$ with 2 parameters is marginally faster (a few percent) than Mid$ with 3 parameters if output is the same.
7. Left$, Mid$ and Right$ run in O(n) time, where n is the number of characters returned.

In a summary, Left$, Mid$ and Right$ spend their time making a (partial) copy of the input string. Copying is the performance bottleneck. Now, how can we take advantage of these findings?

Guidelines for Left$, Mid$ and Right$

Left$(s, n)   Mid$(s, x)   Mid$(s, x, n)   Right$(s, n)

- Don't copy too many characters. Use as low n as possible.
- Use Left$ and Right$ where you would intuitively use them. Don't use Mid$ instead.
- Replace Mid$(s, 1, n) with Left$(s, n).
- Replace Mid$(s, x) with Mid$(s, x, n) if truncated output is OK. This limits the output size to a reasonable n. Note that this optimization is impossible when you need the end of a string. See next.
- To retrieve the end of a string, Right$(s, n) is fastest, then Mid$(s, x), then Mid$(s, x, n). Note that since the functions take different parameters, accurately computing the parameters may pose an intellectual challenge.
- Where Mid$(s, x, n) returns the end of a string, replace it with Right$(s, n). This change is risky since the calls are not exactly the same. Mid$ may return less than n characters depending on x. Make sure you don't add a

bug with this optimization.

Left$, Mid$ and Right$ nested inside Asc or AscW

The functions Asc and AscW are frequently used together with Left$, Mid$ and Right$ to tell what a specific character is. Asc and AscW are indeed good fast functions for this purpose. AscW is actually faster, but we'll go into that a bit later.

In the following, what is said about AscW also applies to Asc.

AscW(Left$(..)) is useless

Don't nest AscW and Left$. It makes no sense. AscW(Left$(s, n)) is equivalent to AscW(s). Since AscW only looks at the first character of s, the call to Left$ just slows down your program without adding anything useful.

AscW(Mid$(..)) considerations

Don't copy too many characters with Mid$. AscW only examines the first character anyway. AscW(Mid$(s, x, 1)) is the best call. Note that if you call AscW(Mid$(s, x)) without the third parameter, Mid$ executes slowly when s is a long string.

Example of potentially slow code:

```
For x = 1 To Len(s)
  If AscW(Mid$(s, x)) = ... Then ...
Next
```

The above is better written as:

```
For x = 1 To Len(s)
  If AscW(Mid$(s, x, 1)) = ... Then ...
Next
```

The performance difference becomes apparent when s is a fairly long string. If you don't test your program with long inputs, you might not notice the performance problem. Users with long inputs will notice it.

AscW(Right$(..)) considerations

When calling AscW(Right$(s, n)) we need to consider n, the number of characters returned by Right$. Performance problems don't exist when n is small. When s is a long string and n can get large, Right$ should not be used.

This call is unoptimal:

```
AscW(Right$(s, n))
```

Replace Right$ with Mid$ that returns one character only. Here:

```
AscW(Mid$(s, Len(s) - n + 1, 1))
```

Both calls examine the nth character from the end of s. The first call copies n characters, while the second call copies just one character.

Use Unicode: AscW and ChrW$

VB6 works internally with Unicode. Every string is in Unicode, which takes 2 bytes per character. Unicode makes a developer's life simpler. Unfortunately it doesn't make a VB6 developer's life any simpler! While VB6 uses Unicode for strings, it still uses Ansi for input, output and forms.

Because of historical reasons, many VB developers stick to the good old Asc() and Chr$() unless they intend to write international applications. You don't have to be writing international applications to take advantage of a couple of Unicode optimizations. If you're concerned about speed, use the "wide" Unicode versions of these functions: AscW() and ChrW$().

AscW() is not the same as Asc(). They can return different values for the same character.

ChrW$() is not equal to Chr$() either. They take different parameter values. Alternatively, they can return a different character for the same input value.

The good news for string optimizers is that for characters in the plain old ASCII range (0 – 127), AscW equals Asc and ChrW$ equals Chr$. Very good! So go ahead and replace Asc with AscW and Chr with ChrW as long as you keep in the 0 – 127 range, that is, as long as you process plain 7-bit ASCII.

> Suggestion: Run Project Analyzer on your code to detect the slower Asc and Chr versions for replacement.

Asc/AscW and Chr/ChrW tables

What if you need to work outside the range 0 – 127? Differences (bugs) will show up if you mix Asc/AscW or Chr/ChrW outside that range.

The differences are easiest to understand in the form of codepage tables. The following tables show what Asc and AscW return for characters beyond 0 – 127. If you are not familiar with codepages, English-speaking users and those in Western Europe and Americas will usually use codepage 1252 (Latin I). After the tables we are going to discuss how to convert Asc to AscW and Chr to ChrW, and the problems there are to expect.

(skip tables)

ch = Character
Asc = ANSI value of char, specific to codepage
AscW = Unicode value of char, independent of codepage
Example using first table: Asc("€")=128, AscW("€")=8364, Chr$(128)=ChrW$(8364)="€"

Asc/AscW values in codepage 1250 ANSI Central European

ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW
€	128	8364		144	144		160	160	°	176	176	Ŕ	192	340	Đ	208	272	ŕ	224	341	đ	240	273
	129	129	‘	145	8216	ˇ	161	711	±	177	177	Á	193	193	Ń	209	323	á	225	225	ń	241	324
‚	130	8218	’	146	8217	˘	162	728	˛	178	731	Â	194	194	Ň	210	327	â	226	226	ň	242	328
ƒ	131	131	“	147	8220	Ł	163	321	ł	179	322	Ă	195	258	Ó	211	211	ă	227	259	ó	243	243
„	132	8222	”	148	8221	¤	164	164	´	180	180	Ä	196	196	Ô	212	212	ä	228	228	ô	244	244
…	133	8230	•	149	8226	Ą	165	260	µ	181	181	Ĺ	197	313	Ő	213	336	ĺ	229	314	ő	245	337
†	134	8224	–	150	8211	¦	166	166	¶	182	182	Ć	198	262	Ö	214	214	ć	230	263	ö	246	246
‡	135	8225	—	151	8212	§	167	167	·	183	183	Ç	199	199	×	215	215	ç	231	231	÷	247	247
ˆ	136	136	˜	152	152	¨	168	168	¸	184	184	Č	200	268	Ř	216	344	č	232	269	ř	248	345
‰	137	8240	™	153	8482	©	169	169	ą	185	261	É	201	201	Ů	217	366	é	233	233	ů	249	367
Š	138	352	š	154	353	Ş	170	350	ş	186	351	Ę	202	280	Ú	218	218	ę	234	281	ú	250	250
‹	139	8249	›	155	8250	«	171	171	»	187	187	Ë	203	203	Ű	219	368	ë	235	235	ű	251	369
Ś	140	346	ś	156	347	¬	172	172	Ľ	188	317	Ě	204	282	Ü	220	220	ě	236	283	ü	252	252
Ť	141	356	ť	157	357		173	173	˝	189	733	Í	205	205	Ý	221	221	í	237	237	ý	253	253
Ž	142	381	ž	158	382	®	174	174	ľ	190	318	Î	206	206	Ţ	222	354	î	238	238	ţ	254	355
Ź	143	377	ź	159	378	Ż	175	379	ż	191	380	Ď	207	270	ß	223	223	ď	239	271	˙	255	729

Asc/AscW values in codepage 1251 ANSI Cyrillic

ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW
Ђ	128	1026	ђ	144	1106		160	160	°	176	176	А	192	1040	Р	208	1056	а	224	1072	р	240	1088
Ѓ	129	1027	‘	145	8216	Ў	161	1038	±	177	177	Б	193	1041	С	209	1057	б	225	1073	с	241	1089
‚	130	8218	’	146	8217	ў	162	1118	І	178	1030	В	194	1042	Т	210	1058	в	226	1074	т	242	1090
ѓ	131	1107	“	147	8220	Ј	163	1032	і	179	1110	Г	195	1043	У	211	1059	г	227	1075	у	243	1091
„	132	8222	”	148	8221	¤	164	164	ґ	180	1169	Д	196	1044	Ф	212	1060	д	228	1076	ф	244	1092
…	133	8230	•	149	8226	Ґ	165	1168	µ	181	181	Е	197	1045	Х	213	1061	е	229	1077	х	245	1093
†	134	8224	–	150	8211	¦	166	166	¶	182	182	Ж	198	1046	Ц	214	1062	ж	230	1078	ц	246	1094
‡	135	8225	—	151	8212	§	167	167	·	183	183	З	199	1047	Ч	215	1063	з	231	1079	ч	247	1095
€	136	8364	˜	152	152	Ё	168	1025	ё	184	1105	И	200	1048	Ш	216	1064	и	232	1080	ш	248	1096
‰	137	8240	™	153	8482	©	169	169	№	185	8470	Й	201	1049	Щ	217	1065	й	233	1081	щ	249	1097
Љ	138	1033	љ	154	1113	Є	170	1028	є	186	1108	К	202	1050	Ъ	218	1066	к	234	1082	ъ	250	1098
‹	139	8249	›	155	8250	«	171	171	»	187	187	Л	203	1051	Ы	219	1067	л	235	1083	ы	251	1099
Њ	140	1034	њ	156	1114	¬	172	172	ј	188	1112	М	204	1052	Ь	220	1068	м	236	1084	ь	252	1100
Ќ	141	1036	ќ	157	1116		173	173	Ѕ	189	1029	Н	205	1053	Э	221	1069	н	237	1085	э	253	1101
Ћ	142	1035	ћ	158	1115	®	174	174	ѕ	190	1109	О	206	1054	Ю	222	1070	о	238	1086	ю	254	1102
Џ	143	1039	џ	159	1119	Ї	175	1031	ї	191	1111	П	207	1055	Я	223	1071	п	239	1087	я	255	1103

Asc/AscW values in codepage 1252 ANSI Latin I

ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW
€	128	8364		144	144		160	160	°	176	176	À	192	192	Ð	208	208	à	224	224	ð	240	240
	129	129	‘	145	8216	¡	161	161	±	177	177	Á	193	193	Ñ	209	209	á	225	225	ñ	241	241
‚	130	8218	’	146	8217	¢	162	162	²	178	178	Â	194	194	Ò	210	210	â	226	226	ò	242	242
ƒ	131	402	“	147	8220	£	163	163	³	179	179	Ã	195	195	Ó	211	211	ã	227	227	ó	243	243
„	132	8222	”	148	8221	¤	164	164	´	180	180	Ä	196	196	Ô	212	212	ä	228	228	ô	244	244
…	133	8230	•	149	8226	¥	165	165	µ	181	181	Å	197	197	Õ	213	213	å	229	229	õ	245	245
†	134	8224	–	150	8211	¦	166	166	¶	182	182	Æ	198	198	Ö	214	214	æ	230	230	ö	246	246
‡	135	8225	—	151	8212	§	167	167	·	183	183	Ç	199	199	×	215	215	ç	231	231	÷	247	247
ˆ	136	710	˜	152	732	¨	168	168	¸	184	184	È	200	200	Ø	216	216	è	232	232	ø	248	248
‰	137	8240	™	153	8482	©	169	169	¹	185	185	É	201	201	Ù	217	217	é	233	233	ù	249	249
Š	138	352	š	154	353	ª	170	170	º	186	186	Ê	202	202	Ú	218	218	ê	234	234	ú	250	250
‹	139	8249	›	155	8250	«	171	171	»	187	187	Ë	203	203	Û	219	219	ë	235	235	û	251	251
Œ	140	338	œ	156	339	¬	172	172	¼	188	188	Ì	204	204	Ü	220	220	ì	236	236	ü	252	252
	141	141		157	157		173	173	½	189	189	Í	205	205	Ý	221	221	í	237	237	ý	253	253
Ž	142	381	ž	158	382	®	174	174	¾	190	190	Î	206	206	Þ	222	222	î	238	238	þ	254	254
	143	143	Ÿ	159	376	¯	175	175	¿	191	191	Ï	207	207	ß	223	223	ï	239	239	ÿ	255	255

Asc/AscW values in codepage 1253 ANSI Greek

ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW
€	128	8364		144	144		160	160	°	176	176	ΐ	192	912	Π	208	928	ΰ	224	944	π	240	960
	129	129	‘	145	8216	΅	161	901	±	177	177	Α	193	913	Ρ	209	929	α	225	945	ρ	241	961
‚	130	8218	’	146	8217	Ά	162	902	²	178	178	Β	194	914	□	210	-1798	β	226	946	ς	242	962
ƒ	131	402	“	147	8220	£	163	163	³	179	179	Γ	195	915	Σ	211	931	γ	227	947	σ	243	963
„	132	8222	”	148	8221	¤	164	164	΄	180	900	Δ	196	916	Τ	212	932	δ	228	948	τ	244	964
…	133	8230	•	149	8226	¥	165	165	µ	181	181	Ε	197	917	Υ	213	933	ε	229	949	υ	245	965
†	134	8224	–	150	8211	¦	166	166	¶	182	182	Ζ	198	918	Φ	214	934	ζ	230	950	φ	246	966
‡	135	8225	—	151	8212	§	167	167	·	183	183	Η	199	919	Χ	215	935	η	231	951	χ	247	967
ˆ	136	136	˜	152	152	¨	168	168	Έ	184	904	Θ	200	920	Ψ	216	936	θ	232	952	ψ	248	968
‰	137	8240	™	153	8482	©	169	169	Ή	185	905	Ι	201	921	Ω	217	937	ι	233	953	ω	249	969
Š	138	138	š	154	154	□	170	-1799	Ί	186	906	Κ	202	922	Ϊ	218	938	κ	234	954	ϊ	250	970
‹	139	8249	›	155	8250	«	171	171	»	187	187	Λ	203	923	Ϋ	219	939	λ	235	955	ϋ	251	971
Œ	140	140	œ	156	156	¬	172	172	Ό	188	908	Μ	204	924	ά	220	940	μ	236	956	ό	252	972
	141	141		157	157		173	173	½	189	189	Ν	205	925	έ	221	941	ν	237	957	ύ	253	973
Ž	142	142	ž	158	158	®	174	174	Ύ	190	910	Ξ	206	926	ή	222	942	ξ	238	958	ώ	254	974
	143	143	Ÿ	159	159	―	175	8213	Ώ	191	911	Ο	207	927	ί	223	943	ο	239	959	□	255	-1797

Asc/AscW values in codepage 1254 ANSI Turkish

ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW
€	128	8364		144	144		160	160	°	176	176	À	192	192	Ğ	208	286	à	224	224	ğ	240	287
	129	129	‘	145	8216	¡	161	161	±	177	177	Á	193	193	Ñ	209	209	á	225	225	ñ	241	241
‚	130	8218	’	146	8217	¢	162	162	²	178	178	Â	194	194	Ò	210	210	â	226	226	ò	242	242
ƒ	131	402	“	147	8220	£	163	163	³	179	179	Ã	195	195	Ó	211	211	ã	227	227	ó	243	243
„	132	8222	”	148	8221	¤	164	164	´	180	180	Ä	196	196	Ô	212	212	ä	228	228	ô	244	244
…	133	8230	•	149	8226	¥	165	165	µ	181	181	Å	197	197	Õ	213	213	å	229	229	õ	245	245
†	134	8224	–	150	8211	¦	166	166	¶	182	182	Æ	198	198	Ö	214	214	æ	230	230	ö	246	246
‡	135	8225	—	151	8212	§	167	167	·	183	183	Ç	199	199	×	215	215	ç	231	231	÷	247	247
ˆ	136	710	˜	152	732	¨	168	168	¸	184	184	È	200	200	Ø	216	216	è	232	232	ø	248	248
‰	137	8240	™	153	8482	©	169	169	¹	185	185	É	201	201	Ù	217	217	é	233	233	ù	249	249
Š	138	352	š	154	353	ª	170	170	º	186	186	Ê	202	202	Ú	218	218	ê	234	234	ú	250	250
‹	139	8249	›	155	8250	«	171	171	»	187	187	Ë	203	203	Û	219	219	ë	235	235	û	251	251
Œ	140	338	œ	156	339	¬	172	172	¼	188	188	Ì	204	204	Ü	220	220	ì	236	236	ü	252	252
	141	141		157	157		173	173	½	189	189	Í	205	205	İ	221	304	í	237	237	ı	253	305
Ž	142	142	ž	158	158	®	174	174	¾	190	190	Î	206	206	Ş	222	350	î	238	238	ş	254	351
	143	143	Ÿ	159	376	¯	175	175	¿	191	191	Ï	207	207	ß	223	223	ï	239	239	ÿ	255	255

Asc/AscW values in codepage 1255 ANSI Hebrew

ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW
€	128	8364		144	144		160	160	°	176	176	ְ	192	1456	׀	208	1472	א	224	1488	נ	240	1504
	129	129	‘	145	8216	¡	161	161	±	177	177	ֱ	193	1457	ׁ	209	1473	ב	225	1489	ס	241	1505
‚	130	8218	’	146	8217	¢	162	162	²	178	178	ֲ	194	1458	ׂ	210	1474	ג	226	1490	ע	242	1506
ƒ	131	402	“	147	8220	£	163	163	³	179	179	ֳ	195	1459	׃	211	1475	ד	227	1491	ף	243	1507
„	132	8222	”	148	8221	₪	164	8362	´	180	180	ִ	196	1460	װ	212	1520	ה	228	1492	פ	244	1508
…	133	8230	•	149	8226	¥	165	165	µ	181	181	ֵ	197	1461	ױ	213	1521	ו	229	1493	ץ	245	1509
†	134	8224	–	150	8211	¦	166	166	¶	182	182	ֶ	198	1462	ײ	214	1522	ז	230	1494	צ	246	1510
‡	135	8225	—	151	8212	§	167	167	·	183	183	ַ	199	1463	׳	215	1523	ח	231	1495	ק	247	1511
ˆ	136	710	˜	152	732	¨	168	168	¸	184	184	ָ	200	1464	״	216	1524	ט	232	1496	ר	248	1512
‰	137	8240	™	153	8482	©	169	169	¹	185	185	ֹ	201	1465	□	217	-1907	י	233	1497	ש	249	1513
Š	138	138	š	154	154	×	170	215	÷	186	247	ֺ	202	1466	□	218	-1906	ך	234	1498	ת	250	1514
‹	139	8249	›	155	8250	«	171	171	»	187	187	ֻ	203	1467	□	219	-1905	כ	235	1499	□	251	-1900
Œ	140	140	œ	156	156	¬	172	172	¼	188	188	ּ	204	1468	□	220	-1904	ל	236	1500	□	252	-1899
	141	141		157	157		173	173	½	189	189	ֽ	205	1469	□	221	-1903	ם	237	1501		253	8206
Ž	142	142	ž	158	158	®	174	174	¾	190	190	־	206	1470	□	222	-1902	מ	238	1502		254	8207
	143	143	Ÿ	159	159	¯	175	175	¿	191	191	ֿ	207	1471	□	223	-1901	ן	239	1503	□	255	-1898

Asc/AscW values in codepage 1256 ANSI Arabic

ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW
€	128	8364	گ	144	1711		160	160	°	176	176	ہ	192	1729	ذ	208	1584	à	224	224	ً	240	1611
پ	129	1662	‘	145	8216	،	161	1548	±	177	177	ء	193	1569	ر	209	1585	ل	225	1604	ٌ	241	1612
‚	130	8218	’	146	8217	¢	162	162	²	178	178	آ	194	1570	ز	210	1586	â	226	226	ٍ	242	1613
ƒ	131	402	“	147	8220	£	163	163	³	179	179	أ	195	1571	س	211	1587	م	227	1605	َ	243	1614
„	132	8222	”	148	8221	¤	164	164	´	180	180	ؤ	196	1572	ش	212	1588	ن	228	1606	ô	244	244
…	133	8230	•	149	8226	¥	165	165	µ	181	181	إ	197	1573	ص	213	1589	ه	229	1607	ُ	245	1615
†	134	8224	–	150	8211	¦	166	166	¶	182	182	ئ	198	1574	ض	214	1590	و	230	1608	ِ	246	1616
‡	135	8225	—	151	8212	§	167	167	·	183	183	ا	199	1575	×	215	215	ç	231	231	÷	247	247
ˆ	136	710	ک	152	1705	¨	168	168	¸	184	184	ب	200	1576	ط	216	1591	è	232	232	ّ	248	1617
‰	137	8240	™	153	8482	©	169	169	¹	185	185	ة	201	1577	ظ	217	1592	é	233	233	ù	249	249
ٹ	138	1657	ڑ	154	1681	ھ	170	1726	؛	186	1563	ت	202	1578	ع	218	1593	ê	234	234	ْ	250	1618
‹	139	8249	›	155	8250	«	171	171	»	187	187	ث	203	1579	غ	219	1594	ë	235	235	û	251	251
Œ	140	338	œ	156	339	¬	172	172	¼	188	188	ج	204	1580	ـ	220	1600	ى	236	1609	ü	252	252
چ	141	1670		157	8204		173	173	½	189	189	ح	205	1581	ف	221	1601	ي	237	1610		253	8206
ژ	142	1688		158	8205	®	174	174	¾	190	190	خ	206	1582	ق	222	1602	î	238	238		254	8207
ڈ	143	1672	ں	159	1722	¯	175	175	؟	191	1567	د	207	1583	ك	223	1603	ï	239	239	ے	255	1746

Asc/AscW values in codepage 1257 ANSI Baltic

ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW
€	128	8364		144	144		160	160	°	176	176	Ą	192	260	Š	208	352	ą	224	261	š	240	353
	129	129	‘	145	8216	□	161	-1796	±	177	177	Į	193	302	Ń	209	323	į	225	303	ń	241	324
‚	130	8218	’	146	8217	¢	162	162	²	178	178	Ā	194	256	Ņ	210	325	ā	226	257	ņ	242	326
ƒ	131	131	“	147	8220	£	163	163	³	179	179	Ć	195	262	Ó	211	211	ć	227	263	ó	243	243
„	132	8222	”	148	8221	¤	164	164	´	180	180	Ä	196	196	Ō	212	332	ä	228	228	ō	244	333
…	133	8230	•	149	8226	□	165	-1795	µ	181	181	Å	197	197	Õ	213	213	å	229	229	õ	245	245
†	134	8224	–	150	8211	¦	166	166	¶	182	182	Ę	198	280	Ö	214	214	ę	230	281	ö	246	246
‡	135	8225	—	151	8212	§	167	167	·	183	183	Ē	199	274	×	215	215	ē	231	275	÷	247	247
ˆ	136	136	˜	152	152	Ø	168	216	ø	184	248	Č	200	268	Ų	216	370	č	232	269	ų	248	371
‰	137	8240	™	153	8482	©	169	169	¹	185	185	É	201	201	Ł	217	321	é	233	233	ł	249	322
Š	138	138	š	154	154	Ŗ	170	342	ŗ	186	343	Ź	202	377	Ś	218	346	ź	234	378	ś	250	347
‹	139	8249	›	155	8250	«	171	171	»	187	187	Ė	203	278	Ū	219	362	ė	235	279	ū	251	363
Œ	140	140	œ	156	156	¬	172	172	¼	188	188	Ģ	204	290	Ü	220	220	ģ	236	291	ü	252	252
¨	141	168	¯	157	175		173	173	½	189	189	Ķ	205	310	Ż	221	379	ķ	237	311	ż	253	380
ˇ	142	711	˛	158	731	®	174	174	¾	190	190	Ī	206	298	Ž	222	381	ī	238	299	ž	254	382
¸	143	184	Ÿ	159	159	Æ	175	198	æ	191	230	Ļ	207	315	ß	223	223	ļ	239	316	˙	255	729

Asc/AscW values in codepage 1258 ANSI/OEM - Vietnamese

ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW
€	128	8364		144	144		160	160	°	176	176	À	192	192	Đ	208	272	à	224	224	đ	240	273
	129	129	‘	145	8216	¡	161	161	±	177	177	Á	193	193	Ñ	209	209	á	225	225	ñ	241	241
‚	130	8218	’	146	8217	¢	162	162	²	178	178	Â	194	194	̉	210	777	â	226	226	̣	242	803
ƒ	131	402	“	147	8220	£	163	163	³	179	179	Ă	195	258	Ó	211	211	ă	227	259	ó	243	243
„	132	8222	”	148	8221	¤	164	164	´	180	180	Ä	196	196	Ô	212	212	ä	228	228	ô	244	244
…	133	8230	•	149	8226	¥	165	165	µ	181	181	Å	197	197	Ơ	213	416	å	229	229	ơ	245	417
†	134	8224	–	150	8211	¦	166	166	¶	182	182	Æ	198	198	Ö	214	214	æ	230	230	ö	246	246
‡	135	8225	—	151	8212	§	167	167	·	183	183	Ç	199	199	×	215	215	ç	231	231	÷	247	247
ˆ	136	710	˜	152	732	¨	168	168	¸	184	184	È	200	200	Ø	216	216	è	232	232	ø	248	248
‰	137	8240	™	153	8482	©	169	169	¹	185	185	É	201	201	Ù	217	217	é	233	233	ù	249	249
Š	138	138	š	154	154	ª	170	170	º	186	186	Ê	202	202	Ú	218	218	ê	234	234	ú	250	250
‹	139	8249	›	155	8250	«	171	171	»	187	187	Ë	203	203	Û	219	219	ë	235	235	û	251	251
Œ	140	338	œ	156	339	¬	172	172	¼	188	188	̀	204	768	Ü	220	220	́	236	769	ü	252	252
	141	141		157	157		173	173	½	189	189	Í	205	205	Ư	221	431	í	237	237	ư	253	432
Ž	142	142	ž	158	158	®	174	174	¾	190	190	Î	206	206	̃	222	771	î	238	238	₫	254	8363
	143	143	Ÿ	159	376	¯	175	175	¿	191	191	Ï	207	207	ß	223	223	ï	239	239	ÿ	255	255

Asc/AscW values in codepage 874 MS-DOS Thai

ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW	ch	Asc	AscW
€	128	8364		144	144		160	160	ฐ	176	3600	ภ	192	3616	ะ	208	3632	เ	224	3648	๐	240	3664
	129	129	‘	145	8216	ก	161	3585	ฑ	177	3601	ม	193	3617	ั	209	3633	แ	225	3649	๑	241	3665
‚	130	130	’	146	8217	ข	162	3586	ฒ	178	3602	ย	194	3618	า	210	3634	โ	226	3650	๒	242	3666
ƒ	131	131	“	147	8220	ฃ	163	3587	ณ	179	3603	ร	195	3619	ำ	211	3635	ใ	227	3651	๓	243	3667
„	132	132	”	148	8221	ค	164	3588	ด	180	3604	ฤ	196	3620	ิ	212	3636	ไ	228	3652	๔	244	3668
…	133	8230	•	149	8226	ฅ	165	3589	ต	181	3605	ล	197	3621	ี	213	3637	ๅ	229	3653	๕	245	3669
†	134	134	–	150	8211	ฆ	166	3590	ถ	182	3606	ฦ	198	3622	ึ	214	3638	ๆ	230	3654	๖	246	3670
‡	135	135	—	151	8212	ง	167	3591	ท	183	3607	ว	199	3623	ื	215	3639	็	231	3655	๗	247	3671
ˆ	136	136	˜	152	152	จ	168	3592	ธ	184	3608	ศ	200	3624	ุ	216	3640	่	232	3656	๘	248	3672
‰	137	137	™	153	153	ฉ	169	3593	น	185	3609	ษ	201	3625	ู	217	3641	้	233	3657	๙	249	3673
Š	138	138	š	154	154	ช	170	3594	บ	186	3610	ส	202	3626	ฺ	218	3642	๊	234	3658	๚	250	3674
‹	139	139	›	155	155	ซ	171	3595	ป	187	3611	ห	203	3627	□	219	-1855	๋	235	3659	๛	251	3675
Œ	140	140	œ	156	156	ฌ	172	3596	ผ	188	3612	ฬ	204	3628	□	220	-1854	์	236	3660	□	252	-1851
	141	141		157	157	ญ	173	3597	ฝ	189	3613	อ	205	3629	□	221	-1853	ํ	237	3661	□	253	-1850
Ž	142	142	ž	158	158	ฎ	174	3598	พ	190	3614	ฮ	206	3630	□	222	-1852	๎	238	3662	□	254	-1849
	143	143	Ÿ	159	159	ฏ	175	3599	ฟ	191	3615	ฯ	207	3631	฿	223	3647	๏	239	3663	□	255	-1848

All characters might not show up on all systems. — Most codepages contain a few unused slots. Such slots may show up as a rectangle, a specific character or a regular character. These tables are intended for a general understanding of how VB6 works. Because of the unused slots, do not use these tables as a reliable source for converters. For an accurate definition of code pages, please see a code page reference. — Chinese, Japanese and Korean codepages have been left out as they use double-byte codes exceeding the 128 – 255 range.

Converting Asc to AscW and Chr to ChrW

Generally speaking, AscW and ChrW are safer to use than Asc and Chr. AscW and ChrW will perform the same everywhere. Asc and Chr, on the other hand, run differently in different locales. It thus makes sense to use AscW and ChrW for both optimization and internationalization purposes. The good news is that VB6 is fully capable of using all values of AscW and ChrW regardless of the locale. VB6 can handle Russian characters in USA and Hebrew in Greece and there's nothing extra the user needs to install. Displaying, outputting and inputting strange characters may require tricks, but internally VB6 handles all characters just fine. (Internationalization is beyond the scope of this article.)

The problem with routinely converting Asc to AscW and Chr to ChrW is that your code may change in a subtle way, causing new bugs to be inserted. There are two kinds of bugs to expect:

- Bug 1. Asc and AscW return different values, and so do Chr and ChrW. You can get an unexpected value or character after conversion.
- Bug 2. AscW returns a full range of values from -32768 to 32767. For single-byte codepages (i.e. not Korean/Chinese/Japanese), Asc returns values 0 to 255. Thus, many pieces of code expect only 0 – 255. Make sure your code can deal with negative values and also values exceeding 255. You must use the Integer or Long datatype to store the return value of AscW, whereas your code may have run nicely storing Asc values in a Byte.

If you are working with Latin I codepage (1252), as really many VB6 developers are, the problem characters you need to be aware of are Ansi 128 – 159. Within this range Asc differs from AscW and Chr differs from ChrW. In ranges 0 – 127 and 160 – 255 Unicode equals Ansi. For character values within those ranges converting Asc to AscW and Chr to ChrW should be straightforward and safe, and only make your program more international and faster.

Here are a few examples of how your code may go wrong:

- Asc("€")=128 everywhere else but in the Cyrillic codepage, where it is 136. Best to use AscW("€")=8364 everywhere.
- Asc("½")=189 in the Latin I codepage and some others, but in the Central European codepage, 189 represents the ˝ character. Best to use AscW("½")=189 everywhere.
- Chr$(223)="ß" in the Latin I codepage, but not in most others. Best to use ChrW$(223) to always get "ß".
- Testing whether there is a pound in s="£" succeeds with Asc(s)=163 in many locales. However, it fails in the Central European codepage. Best to test with AscW(s)=163.

As you can see, it's a really good idea to use the Unicode versions, but you must know what you're doing.

Caveat with extra parentheses

When passing strings as an argument in a procedure call, an extra pair of parentheses can lead to making an unnecessary copy of the string. Unfortunately VB6's syntax is somewhat tricky about when parentheses are required and when not. Sometimes parentheses are required while sometimes they are too much.

Consider the following procedure that takes a string parameter by reference. In VB6 there are 2 ways to declare a reference parameter, so we have provided two syntax examples for the same thing:

```
Sub Process(ByRef s As String) ' Preferred syntax
Sub Process(s As String)       ' Alternative syntax
```

Let's further assume the procedure doesn't modify s, but only reads its value. The purpose of ByRef (instead of ByVal) is to avoid making an unnecessary copy of s. So far so good. This looks like optimal coding.

Now, is this the correct way to call the Sub?

```
Process (s)
```

No! That's bad! The parentheses around (s) are extra in VB6. While they are required in VB.NET and a bunch of other programming languages, in VB6 you can (and should) do without them. Here is the correct way:

```
Process s
```

The difference is that s without parentheses is passed by reference, while (s) makes a copy of s.

The syntax is different when calling a function to get its return value. If Process is a function, the correct, optimal way to call it is this:

```
x = Process(s)
```

To make a copy of s, you need to add an extra pair of parentheses:

```
x = Process((s))
```

If you use the obsolete Call keyword, the correct syntax is:

```
Call Process(s)
```

To make a copy of s, you need to add an extra pair of parentheses:

```
Call Process((s))
```

Tricky, isn't it!

Summary of string optimization rules

The following table summarizes the optimization rules presented above.

String optimization rules, Part III

Slow	Fast	When
Left$(s, n)	s	n ≥ Len(s)
Mid$(s, 1)	s	
Mid$(s, 1, n)	s	n ≥ Len(s)
Mid$(s, 1, n)	Left$(s, n)	n < Len(s)
Mid$(s, x, n)	Right$(s, n)	need end-of-string (note bug risk)
Mid$(s, x)	Mid$(s, x, n)	need middle-of-string, can truncate
Right$(s, n)	s	n ≥ Len(s)
AscW(Left$(s, n))	AscW(s)	
AscW(Mid$(s, x))	AscW(Mid$(s, x, 1))	
AscW(Right$(s, n))	AscW(Mid$(s, Len(s) - n + 1, 1))	n > 1
Asc(s)	AscW(s)	return value in range 0..127
Chr$(i)	ChrW$(i)	i in range 0..127
Process (s)	Process s	pass s by reference

Part I | Part II | Part III

Related articles

- How not to optimize in Visual Basic
- Optimize loops
- Restructuring Visual Basic code
- Save memory
- Toolbox for project manager
- VB InStr
- VB tips — Optimize for memory and speed

Optimize string handling in VB6 - Part III
URN:NBN:fi-fe201003011417