L2/26-077

Piotr Grochowski 2026-02-20

Preliminary proposal for compatibility characters for legacy terminal Arabic encodings

1. Evidence of usage

The following Win32 C code will output 256 characters in system console codepage into the character grid, capture those character tiles in UCS-2 if possible, and then output the current console codepage number.

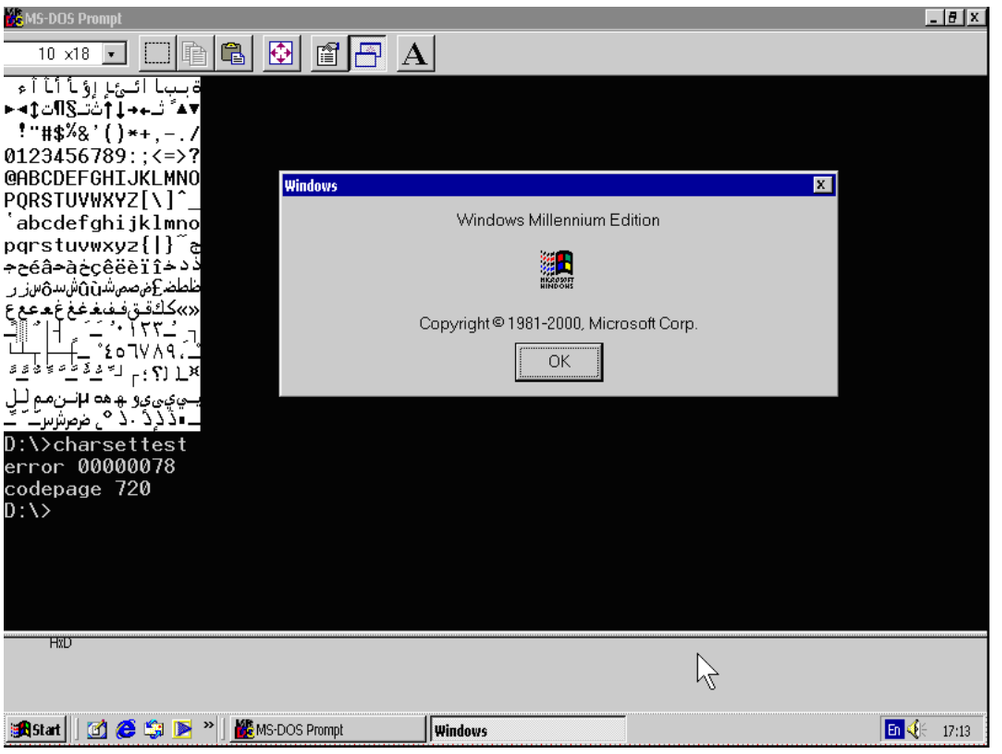
```
#include <windows.h>
#include <stdio.h>
int main(){
 HANDLE hConsole=GetStdHandle(STD_OUTPUT_HANDLE);
 CHAR_INFO screen[256];
 COORD size={16,16,};
 COORD pos={0,0,};
 SMALL_RECT rect={0,0,15,15,};
 for(int i=0;i<256;i++){
  screen[i].Attributes=0xF0;
  screen[i].Char.AsciiChar=i;
 }
 WriteConsoleOutputA(hConsole,screen,size,pos,&rect);
 CHAR_INFO screenu[256];
 if(ReadConsoleOutputW(hConsole,screenu,size,pos,&rect)){
  for(int i=0;i<256;i++) printf("%04X ",screenu[i].Char.UnicodeChar);
 }
 else{
  printf("error %08X\n",GetLastError());
 }
 printf("codepage %u",GetConsoleOutputCP());
}
```

In most cases, whenever a legacy Win32 codepage is used, the application can run on Windows NT to capture the UCS-2 mapping of those character cells to the BMP (although for CJK codepages a more complex setup would be necessary due to thousands of fullwidth characters with 2-byte sequences, which is out of scope for this proposal).

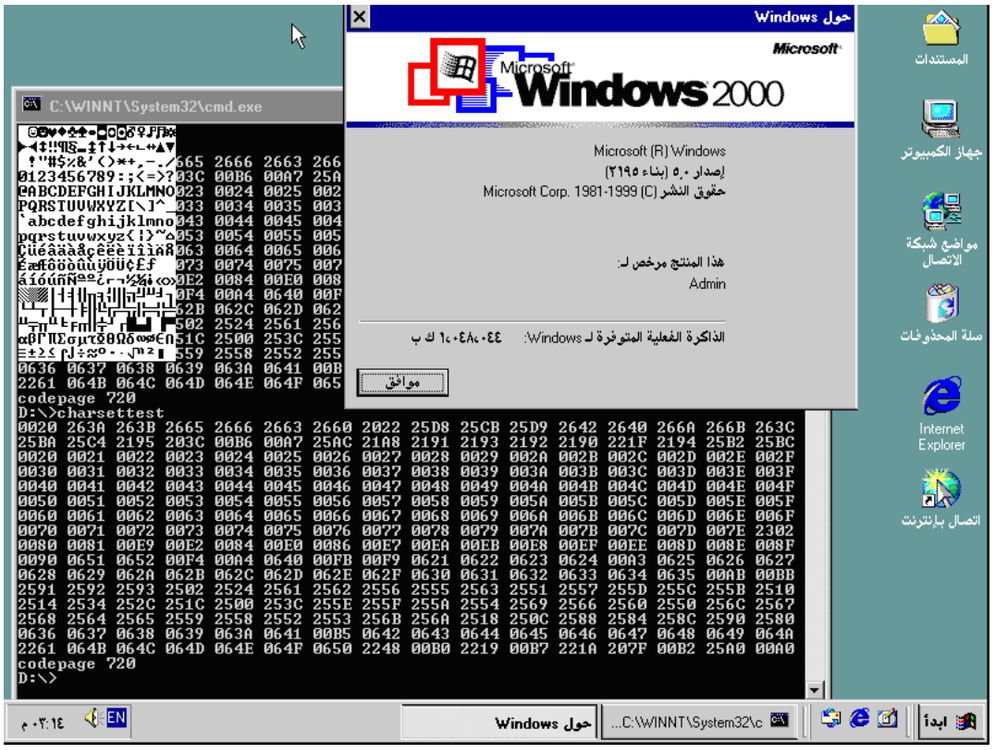
However, in Arabic versions of Windows 9x (95/98/ME) the resulting character set has presentation forms that are not in Unicode. 5×12, 7×12, 8×12, 10×18, 10×20, and 12×16 bitmap fonts have been attested with that character set (VGAOEM.FON, 8514OEM.FON, DOSAPP.FON). It also claims to be using codepage 720, but many characters differ from their CP720 mappings, including the bundled CP_720.NLS mappings (for example, ـ (U+0640 ARABIC TATWEEL) is 0x95 in CP720, but in the console 0x95 is ش instead, and the tatweel is at 0xFF). On Windows 9x, ReadConsoleOutputW is not supported so the UCS-2 mappings of the console character tiles cannot be captured (error 0x00000078 ERROR_CALL_NOT_IMPLEMENTED).

When that program runs on Arabic versions of Windows NT, the visual output is of the CP437 character set if one of the bundled bitmap fonts is used, or the CP720 set if Lucida Console is used, with the Arabic letters either having glitchy font substitution (NT 4.0, NT 5.0/2000) or the .notdef glyph (NT 5.1/XP and up). In fact, the only Arabic bitmap fonts that are found in Windows NT are CP1256 fonts, which are not used in terminals.

Running charsettest in Windows ME Arabic:



Running charsettest in Windows 2000 Arabic (it shows the CP720 mapping but does not include a bitmap Arabic terminal font):



The character set in all attested bitmap font sizes:

5×12	7×12	8×12

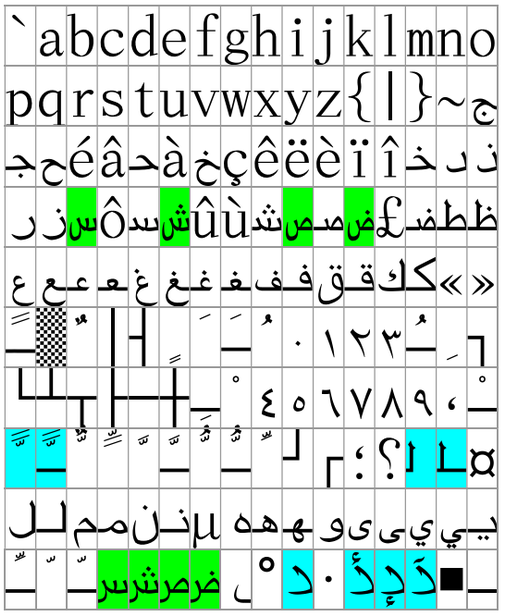
Differences in mapping between sizes:
- in 5×12 and 10×20 sizes, 0x93 has diaeresis instead of circumflex in ô; this is most likely a font error since o with diaeresis does not typically occur in French
- in 10×20, the glyph for 0x97 (ù) is missing

cyan - 8 missing characters
green - 8 characters to be disunified (from currently 4), therefore 4 missing

Total of 12 characters not in Unicode (ﺿﺻﺷﺳ ـًّ ﻟﺬﺬﺬ):
- 6 of them are pieces of lam-alef ligatures ﻟﺬﺬﺬ (DD DE F9 FB FC FD)
- 2 of them are shadda with fathatan ligatures without or with tatweel ًّـ (D0 D1)
-- in some legacy Microsoft fonts, shadda with fathatan is mapped to private use U+E818
- 4 of them are disunifications of seen/sheen/sad/dad occuring either with or without tail
-- ﹳ (U+FE73 ARABIC TAIL FRAGMENT) was originally encoded in Unicode 3.2 for CP864 compatibility; in that codepage, the forms of seen/sheen/sad/dad attach to the tail fragment
-- forms with included tail: ﺲﺶﺺﺾ (92 95 99 9B)
-- forms without tail, attaching to tail fragment like in CP864: ﺿﺻﺷﺳ (F3 F4 F5 F6)

Character table for the attested Windows 9x Arabic terminal character set:

	ء	آ	ﺂ	أ	ﺄ	ؤ	إ	ﺈ	ﺊ	ﺋ	ا	ﺎ	ب	ﺑ	ة
▶	◀	↕	ت	¶	§	ﺗ	ث	↑	↓	→	←	ﺛ	ًّ	▲	▼
	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	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	[	\	]	^	_



Existing characters:

س/ﺱ - U+FEB1 (ARABIC LETTER SEEN ISOLATED FORM)
ش/ﺵ - U+FEB5 (ARABIC LETTER SHEEN ISOLATED FORM)
ص/ﺹ - U+FEB9 (ARABIC LETTER SAD ISOLATED FORM)
ض/ﺽ - U+FEBD (ARABIC LETTER DAD ISOLATED FORM)

Windows 9x Arabic terminal compatibility requires disunifying one of those two sets to add 4 characters:

 - E000 (ARABIC LETTER SEEN ISOLATED FORM WITH TAIL)
 - E001 (ARABIC LETTER SHEEN ISOLATED FORM WITH TAIL)
 - E002 (ARABIC LETTER SAD ISOLATED FORM WITH TAIL)
 - E003 (ARABIC LETTER DAD ISOLATED FORM WITH TAIL)

 - E00C (ARABIC LETTER SEEN ISOLATED FORM WITHOUT TAIL)
 - E00D (ARABIC LETTER SHEEN ISOLATED FORM WITHOUT TAIL)
 - E00E (ARABIC LETTER SAD ISOLATED FORM WITHOUT TAIL)
 - E00F (ARABIC LETTER DAD ISOLATED FORM WITHOUT TAIL)

And adding 8 more characters as well:

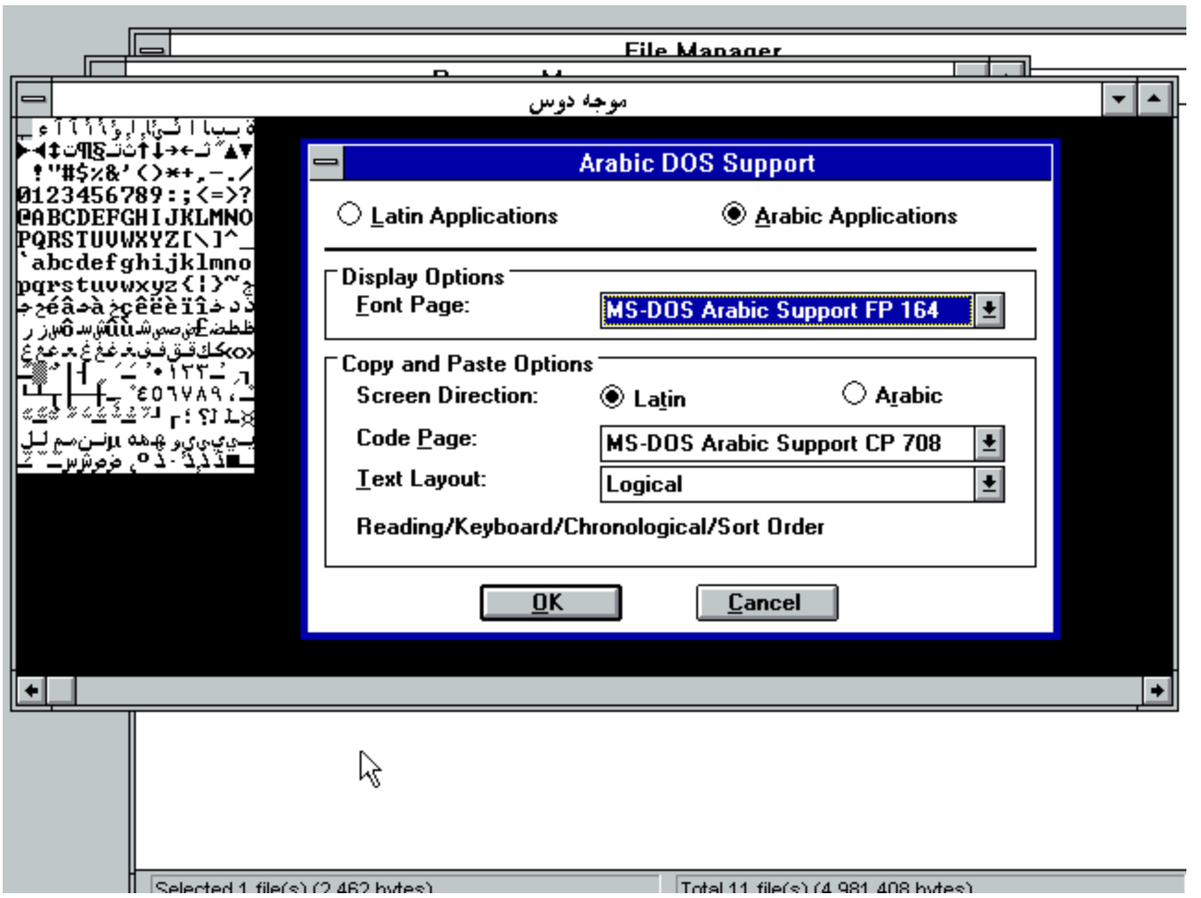
 - E004 (ARABIC LIGATURE SHADDA WITH FATHATAN ISOLATED FORM)
 - E005 (ARABIC LIGATURE SHADDA WITH FATHATAN MEDIAL FORM)
 - E006 (RIGHT HALF ARABIC LIGATURE LAM WITH ALEF ISOLATED FORM)
 - E007 (RIGHT HALF ARABIC LIGATURE LAM WITH ALEF FINAL FORM)
 - E008 (LEFT HALF ARABIC LIGATURE LAM WITH ALEF)
 - E009 (LEFT HALF ARABIC LIGATURE LAM WITH ALEF WITH HAMZA ABOVE)
 - E00A (LEFT HALF ARABIC LIGATURE LAM WITH ALEF WITH HAMZA BELOW)
 - E00B (LEFT HALF ARABIC LIGATURE LAM WITH ALEF WITH MADDA ABOVE)

The following DOS C code will output 256 character tiles into the native text mode character grid:

```
#include <dos.h>
int main(void){
 union REGS regs;
 int i;
 regs.h.ah=0x00; regs.h.al=0x03;
 int86(0x10, &regs, &regs);
 for(i=0; i<256; i++){
  regs.h.ah=0x02; regs.h.bh=0x00;
  regs.h.dh=(i>>4)&0xF; regs.h.dl=i&0xF;
  int86(0x10, &regs, &regs);
  regs.h.ah=0x09; regs.h.al=i;
  regs.h.bh=0x00; regs.h.bl=0xF0;
  regs.x.cx=1;
  int86(0x10, &regs, &regs);
 }
 regs.h.ah = 0x00;
```

```
 int86(0x16, &regs, &regs);
 return 0;
}
```

In Windows 3.1 Arabic, when setting Font Page to MS-DOS Arabic Support FP 164, the result is the same character set as in Windows 95/98/ME Arabic:



The list of available 'Font Page' options:

MS-DOS Arabic Support FP 161
MS-DOS Arabic Support FP 162
MS-DOS Arabic Support FP 163
MS-DOS Arabic Support FP 164
MS-DOS Arabic Support FP 165
Nafitha Enhanced Font
MA/2 Font
Arabic Word Font

Although those options are referred to as font pages, they are directly used in text interchange through int 10h, similarly to WriteConsoleOutputA in Win32.

8×8, 8×12, and 10×20 bitmap fonts (all defined in ARAAPP.FON) have been attested.

All available Arabic terminal encodings in 8×12 font size:

FP161	FP162	FP163	FP164

FP165	Nafitha Enhanced	MA/2	Arabic Word

By analyzing internal font mappings, those glyphs are either mapped to null (0x00) or to one of the 328 internal font glyphs (excluding unused glyphs) where 255 glyphs (01-FF) are in 'Arabic Terminal 1' (having all FP164 glyphs), and 73 glyphs (01 0C 11 18 1A 1B 22 23 26 27 36 3B 45 46 47 48 49 4A 4B 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 7C 7D 80 85 86 A8 DD DE E1 E2 E3 E4 E5 E6 EF F1 F2, with 0x100 offset) are in 'Arabic Terminal 2' (having non-FP164 glyphs). However, 0xD3 and 0x1F2 are visually identical in all font sizes, and they do never both coexist in any font page (glyph 0xD3 occurs in FP162 and FP164 at 0xD3, whereas glyph 0x1F2 occurs in FP165 at 0x92) corresponding to ﷼ (U+FC5F ARABIC LIGATURE SHADDA WITH KASRATAN ISOLATED FORM). 0x20 and 0x1F1 are both whitespace characters in all font sizes, and can be mapped to U+0020, U+00A0, or U+2007 as appropriate (or U+0080-U+009F in case of missing bytes). Therefore, there are 326 visually distinct character tiles throughout all the available encodings. 313 of those tiles are already in Unicode (!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~¢£¤§«¬-°±µ¶·»¼½×àâçèéêëîïôö÷ùûßð).

FP165 and Arabic Word encodings are very similar to CP864, and in fact are already in Unicode. The other Windows 3.1 Arabic terminal encodings (FP161, FP162, FP163, FP164, Nafitha Enhanced, MA/2) have characters not in Unicode 17.0.

12 characters (either E000-E00B or E004-E00F depending on the selected type of disunification) to be added to FP164 are also sufficient in order to complete support for FP161, FP163, Nafitha Enhanced, and MA/2, while a 13th character (E010) is necessary to additionally support FP162:

- E000 (ARABIC LETTER SEEN ISOLATED FORM WITH TAIL)
- 0x92 in FP161/FP162/FP163/FP164, 0x01 in Nafitha Enhanced
- E001 (ARABIC LETTER SHEEN ISOLATED FORM WITH TAIL)
- 0x95 in FP161/FP162/FP163/FP164, 0x02 in Nafitha Enhanced
- E002 (ARABIC LETTER SAD ISOLATED FORM WITH TAIL)
- 0x99 in FP161/FP162/FP163/FP164, 0x03 in Nafitha Enhanced
- E003 (ARABIC LETTER DAD ISOLATED FORM WITH TAIL)
- 0x9B in FP161/FP162/FP163/FP164, 0x04 in Nafitha Enhanced

- E00C (ARABIC LETTER SEEN ISOLATED FORM WITHOUT TAIL)
- 0xF3 in FP161/FP162/FP163/FP164, 0xBC in FP165/Nafitha Enhanced/Arabic Word, 0xE4 in MA/2
- E00D (ARABIC LETTER SHEEN ISOLATED FORM WITHOUT TAIL)
- 0xF4 in FP161/FP162/FP163/FP164/MA/2, 0xBD in FP165/Nafitha Enhanced/Arabic Word
- E00E (ARABIC LETTER SAD ISOLATED FORM WITHOUT TAIL)
- 0xF5 in FP161/FP162/FP163/FP164, 0xBE in FP165/Nafitha Enhanced/Arabic Word, 0xE5 in MA/2
- E00F (ARABIC LETTER DAD ISOLATED FORM WITHOUT TAIL)
- 0xF6 in FP161/FP162/FP163/FP164, 0xEB in FP165/Nafitha Enhanced/Arabic Word, 0xF5 in MA/2

- E004 (ARABIC LIGATURE SHADDA WITH FATHATAN ISOLATED FORM)

- 0xD0 in FP162/FP164
ﱡ - E005 (ARABIC LIGATURE SHADDA WITH FATHATAN MEDIAL FORM)
- 0xD1 in FP164
ﱢ - E006 (RIGHT HALF ARABIC LIGATURE LAM WITH ALEF ISOLATED FORM)
- 0x15 in FP161/FP162, 0xD4 in FP163, 0xDD in FP164, 0xA3 in Nafitha Enhanced, 0xF1 in MA/2
ﱣ - E007 (RIGHT HALF ARABIC LIGATURE LAM WITH ALEF FINAL FORM)
- 0xE6 in FP161/FP162, 0xD5 in FP163, 0xDE in FP164, 0x1C in Nafitha Enhanced, 0xF0 in MA/2
ﱤ - E008 (LEFT HALF ARABIC LIGATURE LAM WITH ALEF)
- 0xF9 in FP161/FP162/FP163/FP164, 0xDB in Nafitha Enhanced, 0xE3 in MA/2
ﱥ - E009 (LEFT HALF ARABIC LIGATURE LAM WITH ALEF WITH HAMZA ABOVE)
- 0xFB in FP161/FP162/FP163/FP164, 0xDC in Nafitha Enhanced, 0xE0 in MA/2
ﱦ - E00A (LEFT HALF ARABIC LIGATURE LAM WITH ALEF WITH HAMZA BELOW)
- 0xFC in FP161/FP162/FP163/FP164, 0xDD in Nafitha Enhanced, 0xE2 in MA/2
ﱧ - E00B (LEFT HALF ARABIC LIGATURE LAM WITH ALEF WITH MADDA ABOVE)
- 0xFD in FP161/FP162/FP163/FP164, 0xDE in Nafitha Enhanced, 0xE1 in MA/2

ﱨ - E010 (ARABIC LIGATURE SHADDA WITH KASRATAN MEDIAL FORM)
- 0xD1 in FP162

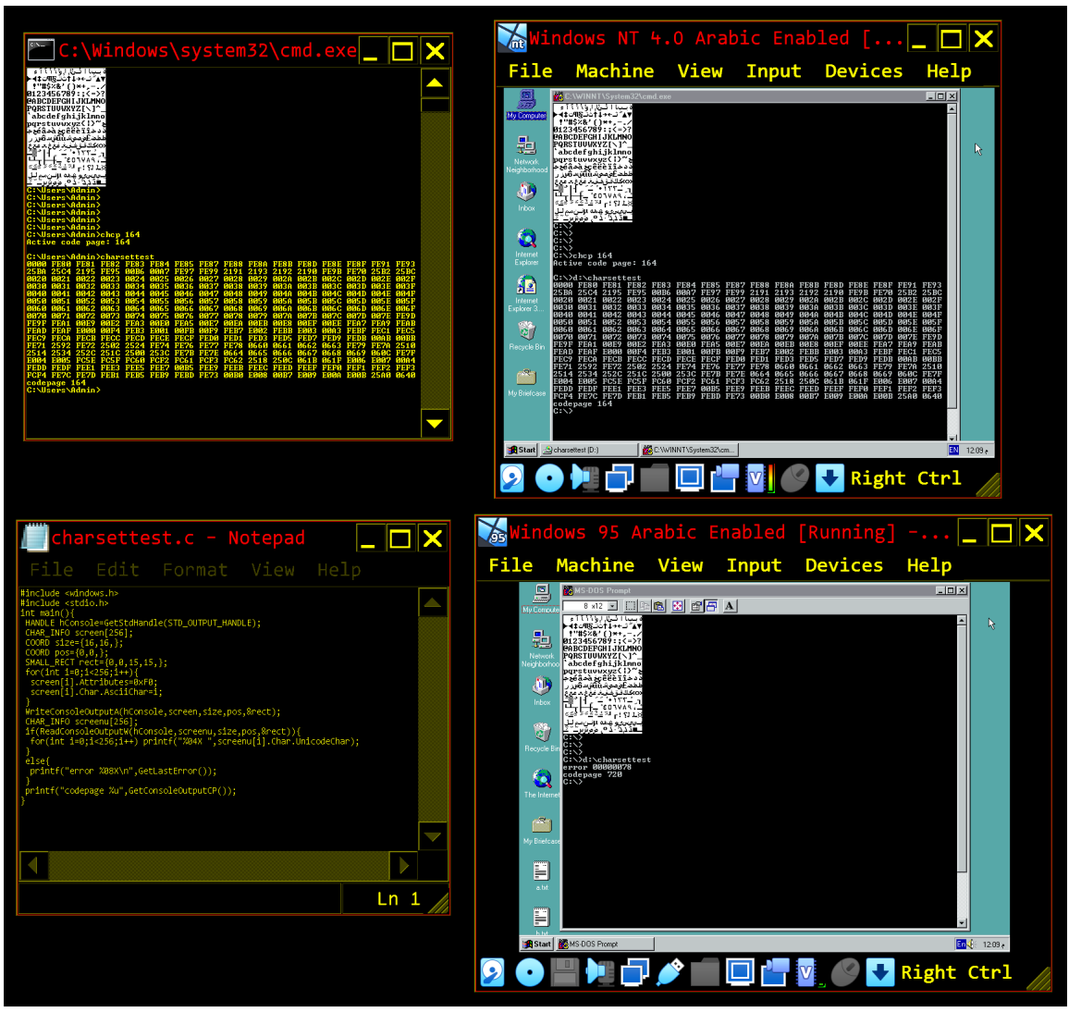
2. Compatibility considerations

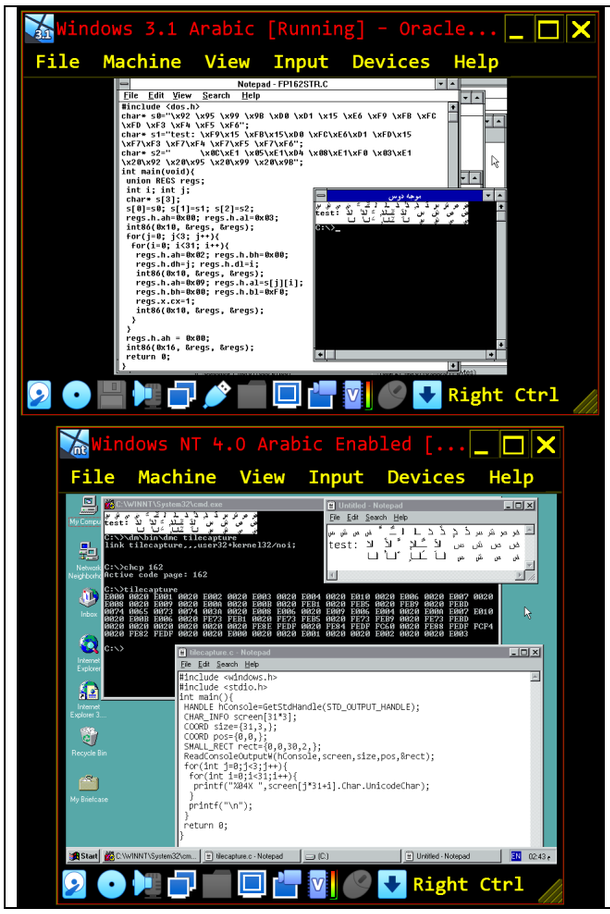
The attested Windows 3.1/95/98/ME Arabic terminal encodings are for random access terminals, where each character cell is independently stored in a constant number of bytes. 12 of the characters have been attested in Win32 console usage and 13 in DOS/Win16 console usage, so incorporating those into Unicode should lead to practical usage for Win32 console compatibility and random access on the character grid. In Windows NT, the character tiles are stored in a fixed-size 4 byte CHAR_INFO structure, consisting of 2 bytes for a union that may store either an 8-bit character or a 16-bit UCS-2 code point, and 2 bytes for attributes, and this layout can be exported with ReadConsoleOutputW. Such a mapping from 8-bit codepage to UCS-2 maintains random access onto the character grid and has already been attested with various legacy codepages such as CP437 and CP852 on Win32. Therefore, for ReadConsoleOutputW to function properly on the CP164 characters, they have to all be located within the Basic Multilingual Plane (BMP).

Some terminals may support the usage of non-BMP characters, variation selector sequences, or combining character sequences in a single displayed character cell. This may be done either by shifting all following character cells to the left like in Windows 95 Vietnamese terminal (therefore reducing the total width of the line), or by increasing the amount of bytes taken by the character cell like in various UTF-8 terminals typically found in Unix-like environments (therefore creating a variable length encoding). However, attempting to use either of those methods to emulate CP164 terminals not only overcomplicates the implementation, but also fails for Win32 compatibility, because if the legacy terminal characters were to be used to fill the entire line, but were encoded as an SMP/VS/combining sequence in Unicode, the terminal would run out of space on the line of text when attempting to store those characters in the array of CHAR_INFO structures. Therefore, encoding those characters as atomic BMP characters is the only reasonable solution.

As of Unicode 17.0, there are 1336 unassigned code points in BMP (55653/56989) (although not all the code points may be used for all purposes). Due to the limited UCS-2 code point space, allocations have to be performed carefully. Characters that have been attested for compatibility on platforms that are linked or associated with the usage of UCS-2 or BMP should be prioritized. The 1336 unassigned code points are mostly scattered throughout existing BMP blocks in Unicode. The only BMP code points not located in any block are in ranges 2FE0-2FEF, but using that space should only be a last resort due to fragmentation. The only code points in remaining BMP Arabic blocks are 0892-0896 FE75 FEFD-FEFE. Some BMP blocks with legacy compatibility characters also have free spaces such as 242A-243F, FB07-FB12, FB18-FB1C, FB37, FB3D, FB3F, FB42, FB45, etc. .

Users may inject legacy codepage support into Windows NT by supplying the appropriate .nls files, and then attaching it into the appropriate registry value (such as in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Nls\CodePage). Then users may install the appropriate fonts that supply the glyphs that correspond to the legacy terminal codepage. In case of bitmap .fon fonts, the glyphs are directly mapped to 8-bit bytes, but in case of TrueType .ttf fonts, the glyphs are mapped to Unicode and therefore have to match the encoding specified in the .nls file. In the screenshot below, the Arabic terminal test is ran on Windows 95 Arabic (representing a Windows 9x Arabic compatible application), and then the appropriate codepage and bitmap font is installed in Windows 7 and Windows NT 4.0. In that setup, both Windows 7 and Windows NT 4.0 display all the glyphs according to the font, and with ReadConsoleOutputW they output all the characters mapped to BMP (with the 12 missing characters being temporarily mapped to E000-E00B, or another option is to replace FEB1/FEB5/FEB9/FEBD with E00C-E00F, and then replace E000-E003 with FEB1/FEB5/FEB9/FEBD). In case of TrueType fonts, those mappings may vary depending on the font vendor. Those private use mappings are not standardized, which makes it difficult for users to reliably interchange text.





In this example, user runs a Windows 3.1 Arabic compatible DOS application with FP162 encoding. User then installs the appropriate font and codepage into Windows NT 4.0 Arabic and then runs the same DOS application in there. User can then subsequently run a Win32 application over the resulting output in order to capture the UCS-2 values of the character tiles, with the missing characters temporarily mapped to private use code points. User then copies the character tiles into the clipboard and pastes into Notepad. The ability for the DOS application to run directly in a Win32 environment that allows for capturing and interchanging the character tiles in UCS-2, especially given the fixed-size format of the CHAR_INFO structure (which excludes the possibility of using non-BMP characters or composition sequences), may warrant adding those characters to the BMP.

3. Frequently asked questions

Q: Is this proposal for a new script or is it for addition of characters to an existing block?
A: Neither. This proposal does not propose a new script, but it also does not prescribe any particular block to add the characters to, though it must be in BMP.

Q: How many characters are being proposed?
A: 12-13 characters. Windows 95/98/ME Arabic terminal compatibility results in 12 new characters. Adding Windows 3.1 Arabic increases that to 13 new characters.

Q: Are the proposed characters contemporary, specialized, major extinct, attested extinct, minor extinct, archaic, hieroglyphic, ideographic, obscure, or questionable?
A: That is subjective. The proposal L2/01-069 assigned the 'specialized, small' category to the tail fragment, but it could be argued that it is because it did not acknowledge the then-contemporary use in Windows ME Arabic terminals. However, the specialized category is usually associated with SMP allocations, while practical compatibility considerations require BMP allocations. Although the Windows 9x Arabic terminal encoding is not directly supported by Windows NT, it can still be made compatible by installing the appropriate font and codepage, which can be patched into the application or installer itself or given as simple instructions on top of existing software. The only missing link is the Unicode mappings to those 12 characters in the BMP. This is not the case with many other legacy computing platforms that require emulation in order to work correctly in modern systems. Win32 platform never falls out of contemporary use like other platforms do. The characters are therefore in between 'contemporary' and 'specialized small', with the 12 characters in Windows 9x Arabic terminal being closer to 'contemporary', and the other 1 character being further away as it is attested for DOS platforms, but can also run on 32-bit Windows platforms.

Q: Is a repertoire including character names provided?
A: Yes, all the attested non-Unicode 17.0 characters have been listed and named.

Q: Who will provide the appropriate computerized font to the UTC?
A: The LegacyArabicReferenceFont will be provided directly by the proposal owner.

Q: Are references to other character sets, dictionaries, or descriptive texts provided?
A: Yes, the proposed characters reference multiple legacy encodings.

Q: Are published examples of use of proposed characters attached?
A: Yes, the screenshots comprehensively demonstrate usage of all character tiles.

Q: Does the proposal address other aspects of character data processing such as input, presentation, sorting, searching, indexing, or transliteration?
A: No, because the characters are for a non-shaping character grid model that does not require the characters to have any particular properties beyond being unambiguously representable in the CHAR_INFO structure, where each character cell is associated with a single UCS-2 codepoint representing a BMP character (within the scope of Unicode) and a series of attributes (outside the scope of Unicode).

Q: Has another document been submitted before that proposes the same characters?
A: No prior proposal specifically addressing any of the proposed characters has been attested.

Q: Has contact been made to members of user communities?
A: Yes, one of the DOS/Windows communities has already been contacted in https://www.betaarchive.com/forum/viewtopic.php?t=30140.

Q: Information on the user community for the proposed characters is included?
A: Yes, Win32 communities span an extremely broad range of users due to its popularity, so relevant information is included throughout this document.

Q: What is the context of use for the proposed characters?
A: The context is in non-shaping character grid terminals, where the rectangular screen is divided into a series of non-overlapping independent rectangular character cells of equal size, and each character cell has a character code and attributes assigned to it. In the context of Win32 platforms with the appropriate codepage installed, each 8-bit character code needs to be convertable into a 16-bit UCS-2 code representing a code point in the BMP. In particular, tiles written using int 10h or WriteConsoleOutputA must be readable using ReadConsoleOutputW.

Q: Are the proposed characters in current use by the user community?
A: The characters are in use by users of Windows 3.1/95/98/ME Arabic terminals or users that install the appropriate fonts and codepages into Windows NT systems.

Q: After extensively evaluating the compatibility and BMP encoding principles, are you sure that the proposed characters must be entirely in the BMP?

A: Yes, because that is fundamentally required for the encodings to be compatible with the fixed size CHAR_INFO structure, which is used in all native Win32 terminals to store character tiles. This preliminary proposal does not allow for non-BMP allocations, as that would defeat the purpose.

Q: Should the proposed characters be kept together in a contiguous range rather than being scattered?
A: Whether the proposed characters are contiguous or scattered does not affect the compatibility usage of the characters.

Q: Can any of the proposed characters be considered a presentation form of an existing character or character sequence?
A: The proposed characters are presentation forms for various Arabic letters. Their inclusion in Unicode is justified by compatibility with Win32 environments that use the characters, but don't have the proper Unicode mapping to it yet.

Q: Can any of the proposed characters be encoded using a composed character sequence of either existing characters or other proposed characters?
A: No, the proposed characters are not equivalent to any composition sequences.

Q: Can any of the proposed characters be considered to be similar in appearance or function to existing characters?
A: The proposed characters are similar to other forms of existing Arabic letters, but their inclusion is essential for compatibility.

Q: Does the proposal include use of combining sequences and/or use of composite sequences?
A: No, as combining sequences are not relevant for fixed-size character grid terminals. While some terminals do use combining sequences to represent characters, those do not use a fixed-size structure to represent character tiles and cannot be natively exported with Win32 functions like WriteConsoleOutputW, and therefore are completely irrelevant in the context of this proposal.

Q: Does the proposal contain characters with any special properties such as control function or similar semantics?
A: The proposed characters should all have the appropriate right to left bidirectional property for consistency with existing Arabic presentation forms.

Q: Does the proposal contain any ideographic compatibility characters?
A: No, the compatibility characters being proposed in here are not ideographic.

Q: What would happen if SEW or UTC reject the idea of FP164 being in the BMP, claiming that those characters may be encoded in the SMP or by compositions or by complex font features?
A: Then that indicates the SEW or UTC do not sufficiently understand the compatibility considerations and therefore they cannot make an informed decision. In that case, there will be a follow up proposal that clarifies the specific reasons for requiring BMP allocation.

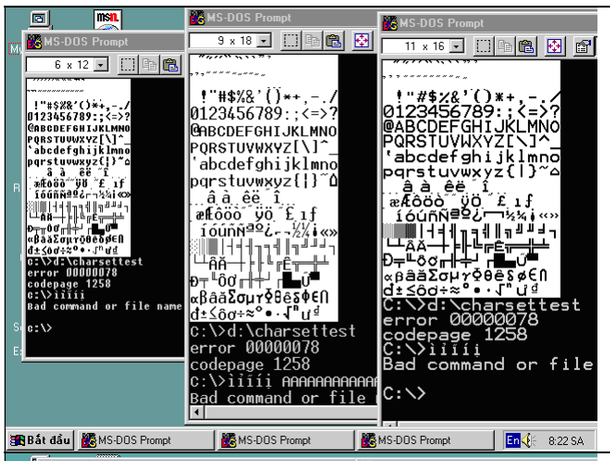
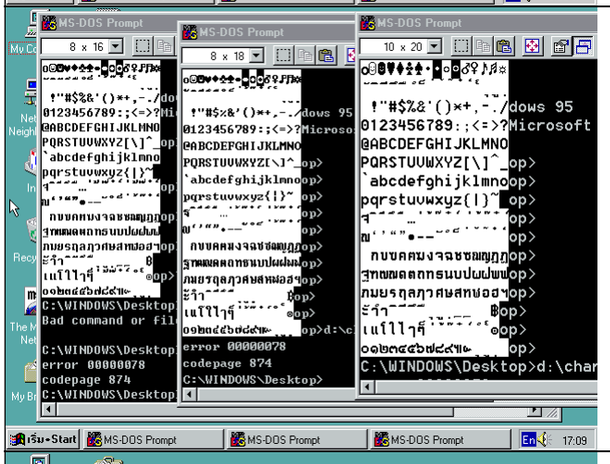
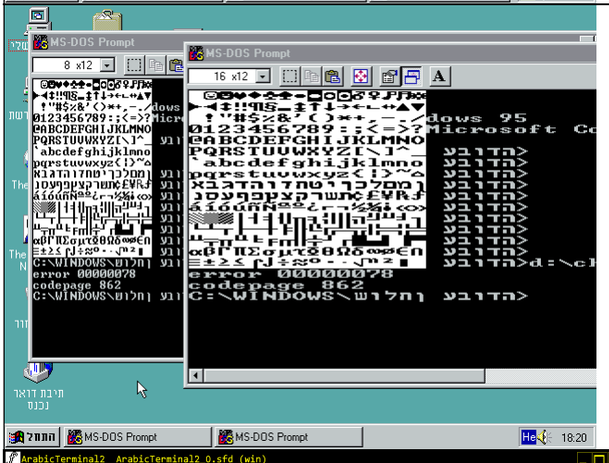
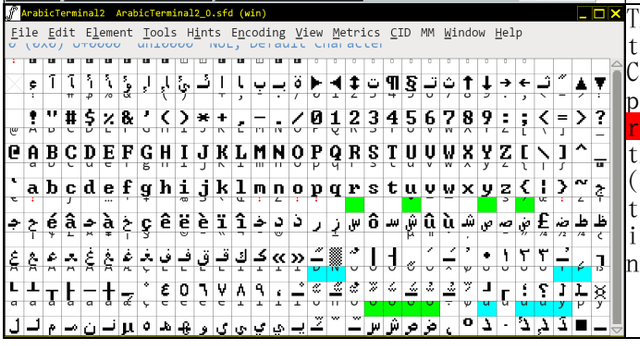
Q: What would happen if SEW or UTC understand the compatibility considerations for why the characters may not be encoded in SMP or by character compositions, but still reject the idea of FP164 being in the BMP anyway?
A: In that case, there would be no follow up proposal, since there is no other way for Unicode to resolve the lack of mapping. User communities and font vendors will then most likely assign BMP private use codepoints to the missing characters, which may differ from the temporary private use assignments used in this proposal. Users may then bundle the appropriate private use mappings in DOS/Win16/Win32 applications, fonts, compatibility layers, or text processing tools as appropriate.

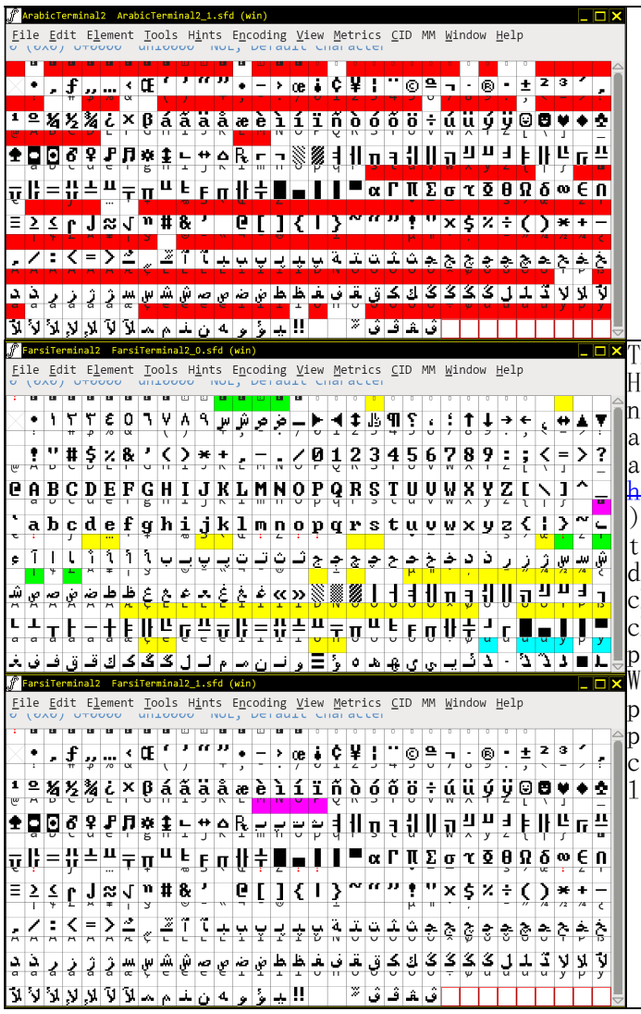
Q: What if SEW or UTC accept the idea of FP164 and other attested Arabic encodings being in the BMP, but reject the specific mapping being proposed?
A: In that case, there would be a follow up proposal that evaluates the feedback and corrects the mapping accordingly. However, defective mappings that map visually and functionally distinct tiles to the same character (such as the Unicode 16.0-17.0 mapping for HP 264x), or mappings that use non-BMP characters or compositions, will not be proposed.

4. Finiteness

Users may raise the concerns that there will be the endless possibility of other DOS/Win16/Win32 terminal encodings requiring new characters in Unicode (such as the precomposed Latin accented forms in CP776/CP777/CP778, or the OCR characters in CP876), and that a policy to encode them in BMP would therefore create duplications of existing non-BMP characters or composition sequences, and that it would increasingly fragment the script blocks and then run out of BMP space. This is in fact not the case, as we are only suggesting the inclusion of the terminal encodings attested in systems published by the Microsoft vendor and having a known compatibility path to Windows NT, not all DOS codepages published by all vendors. Most Win32 terminals map directly to existing codepages (CP437/CP737/CP775/CP850/CP852/CP855/CP857/CP866 for single byte codepages, CP932/CP936/CP949/CP950 for CJK codepages with both single and double byte characters) which are already in BMP and have

already been extensively analyzed by many users. The only exceptions to the 'well known codepage' model could be encodings for complex scripts like Arabic/Hebrew/Thai/Vietnamese for which directly mapping the tiles 1:1 may not have been sufficient on the target platform, resulting in the effective encoding being completely different, as happened with Windows 9x Arabic terminals, where the reported encoding of CP720 is completely different from the actual encoding of FP164. While Windows 9x Arabic supports virtualizing CP720 and CP864 encodings, users may still use the raw mode of the terminal where the FP164 tiles are directly exposed to input/output methods (and therefore are effectively used for text interchange).

	In Windows 95 Vietnamese, the terminal uses shaping to compose Vietnamese diacritics. The reported encoding is CP1258, and accented letters/combining marks are in same locations as in CP1258. The displayed font uses a mixture of CP437 and CP1258. In particular, there appear to be presentation forms of diacritics, and duplications of some accented letters at both CP437 and CP1258 locations with visual distinctions. However, since CP1258 Vietnamese letters already render correctly, it is apparent that the presentation forms and leftover duplicate CP437 letters are not intended for text interchange (unlike the CP164 letters for which there is no other way to render those letters within the same environment), so they can be mapped to the respective C0 and C1 code points, therefore not warranting new characters.
	In Windows 95 Thai, the terminal uses shaping to compose Thai diacritics. The characters already match the CP874 encoding, but the locations unused in CP874 are displayed as presentation forms. Since the terminal already uses shaping with CP874 combining marks to display Thai text, the extra locations can be interpreted as font-specific glyphs (unlike CP164 characters for which no other representation is possible) and therefore can be mapped to C0, C1, or private use, as is already done by the CP_874.NLS and c_874.nls mappings.
	Windows 95 Hebrew uses a non-shaping terminal with CP862, which already has a proper well known mapping where the entire 8-bit space is filled with BMP characters. Since all displayed character tiles match the well known codepage, there are no new characters.
	The internal font mapping of Windows 3.1 Arabic terminal, where the first page is the same mapping as CP164 and Windows 95 Arabic terminal, and the second page consists of non-CP164 characters. Highlighted in red are glyphs not attested in any encoding, and therefore not subject to analysis in this proposal (since there is no known usage of those characters in terminal text interchange, there is no possibility of it being an output of ReadConsoleOutputW, and therefore no known need to encode them in Unicode).



The internal mapping of Windows 3.1 Farsi terminal. Highlighted in magenta are five theoretical characters not in Windows 3.1 Arabic and not in Unicode. However, attempting to install Windows 3.1 Farsi fails (as has already been reported in https://www.betaarchive.com/forum/viewtopic.php?t=19121) so there is no known way to test the system for terminal encodings, and therefore impossible to determine whether those are glyph variations, distinct characters, or unused glyphs. Therefore, those characters are only theoretical, and are not being proposed in here. However, if the potential growth of Windows 3.1 legacy terminal characters in BMP is problematic, the SEW/UTC may choose to restrict this proposal to only the characters attested in Win32 compatible platforms (Windows 95/98/ME), resulting in 12 instead of 13 new characters.

5. Example mapping

This example mapping assumes that E000-E003 would be disunified, E00C-E00F would be mapped to existing FEB1/FEB5/FEB9/FEBD, and that all Windows 3.1 Arabic encodings are included. It also assumes that 10 of the slots in Alphabetic Presentation Forms and 3 of the slots in Arabic Presentation Forms-B will be occupied. However, depending on the decisions that UTC makes regarding this and other proposals, the characters may be allocated differently.

Valid variations: assigning the characters to other free BMP code points, disunifying E00C-E00F instead of E000-E003 (which better matches the existing code charts for FEB1/FEB5/FEB9/FEBD, but would break existing CP864 mappings), and/or excluding E010 (leaving only FP164 characters to be encoded), resulting in either 12 or 13 new characters

Example:

ﺳ - FB09 ← E000 (ARABIC LETTER SEEN ISOLATED FORM WITH TAIL)
ﺷ - FB0A ← E001 (ARABIC LETTER SHEEN ISOLATED FORM WITH TAIL)
ﺻ - FB0B ← E002 (ARABIC LETTER SAD ISOLATED FORM WITH TAIL)
ﺿ - FB0C ← E003 (ARABIC LETTER DAD ISOLATED FORM WITH TAIL)
ﱡ - FEFD ← E004 (ARABIC LIGATURE SHADDA WITH FATHATAN ISOLATED FORM)
ﱢ - FEFE ← E005 (ARABIC LIGATURE SHADDA WITH FATHATAN MEDIAL FORM)
ﻟ - FB0D ← E006 (RIGHT HALF ARABIC LIGATURE LAM WITH ALEF ISOLATED FORM)
ﻠ - FB0E ← E007 (RIGHT HALF ARABIC LIGATURE LAM WITH ALEF FINAL FORM)
ﻼ - FB0F ← E008 (LEFT HALF ARABIC LIGATURE LAM WITH ALEF)
ﻵ - FB10 ← E009 (LEFT HALF ARABIC LIGATURE LAM WITH ALEF WITH HAMZA ABOVE)
ﻷ - FB11 ← E00A (LEFT HALF ARABIC LIGATURE LAM WITH ALEF WITH HAMZA BELOW)
ﻹ - FB12 ← E00B (LEFT HALF ARABIC LIGATURE LAM WITH ALEF WITH MADDA ABOVE)
ﱣ - FE75 ← E010 (ARABIC LIGATURE SHADDA WITH KASRATAN MEDIAL FORM)