



Unicode 의 이해

Understanding of Unicode

Release 1.0

2010/6/30

Copyright 2010 novo networks. All rights reserved.

All information contained herein is the property of novo networks. No part of this publication (whether in hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of novo networks.

No responsibility is assumed by novo networks for the use thereof nor for the rights of third parties which may be effected in any way by the use thereof. Any representation(s) in this document concerning performance of novo networks' product(s) are for informational purposes only and are not warranties of future performance

All other trademarks, service marks, registered trademarks, or registered service marks may be the property of their respective owners. All specifications are subject to change without prior notice.

목 차 (Table of Contents)

1. AUDIENCE	4
2. 일반 용어.....	5
■ CHARACTER SET (CHARSET, 문자셋).....	5
■ CHARACTER ENCODING (ENCODING, 인코딩)	6
3. UNICODE 개요.....	7
■ UNICODE 이전의 문자셋.....	7
■ UNICODE를 만든 이유.....	8
■ UNICODE의 시작.....	8
4. UNICODE 구조.....	10
■ UNICODE의 기본 개념.....	10
■ UNICODE 코드 체계.....	15
5. UNICODE 인코딩.....	17
■ UTF-32.....	18
■ UTF-16.....	19
■ UTF-8.....	21
6. 마치며.....	23
APPENDIX A. 문서정보.....	24
A.1 문서이력.....	24
APPENDIX B. REFERENCES.....	25

1. Audience

Unicode 는 전 세계의 문자를 컴퓨터에서 일관되게 표현하고 다룰 수 있게 설계된 표준입니다. 이 표준은 세상에 나온지 거의 20 년이 다 되어가고, 지금처럼 널리 사용된지도 10 년이 넘었습니다.

그럼에도 불구하고 개발자들 사이에서는 여전히 Unicode 에 대해 잘 모르는 경우가 많습니다. Unicode 는 문자 표현을 위한 표준으로서 특히 다국어를 지원하는 애플리케이션을 작성할 때 우선 고려 대상이 되어야 합니다.

그러기 위해서는 Unicode 에 대한 기본적인 이해가 선행되어야 하며, 본 문서는 이러한 Unicode 에 대한 개략적인 이해를 돕는데 목적을 두고 있습니다.

내용이 표준 규격에 대한 것이다 보니 자칫 용어가 딱딱해지기 쉽습니다. 이를 피하기 위해 정식 명칭 대신에 일반적으로 많이 쓰이는 용어로 줄여서 쓰기도 할 것입니다. 가끔 100% 정확한 단어나 용어가 아니더라도 미리 양해를 구합니다.

2. 일반 용어

■ Character Set (charset, 문자셋)

Character Set 이란 말그대로 문자의 집합, 문자 테이블입니다. 보통 문자셋 혹은 캐릭터셋이라고 불러줍니다.

C 언어를 배울 때 자연스럽게 접하게 되는 ASCII 는 영어 알파벳, 숫자, 제어문자 등 128 개의 문자를 모아놓은 문자셋입니다. 아래와 같이 생겼습니다.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

ASCII에는 한글이 없죠? 당연히 한글을 표현하기 위한 문자셋도 만들어져 있는데, KS C 5601¹이라는 것입니다. 이 문자셋에는 완성형한글(2350 자), 한자, 숫자, 로마/그리스/일본/러시아문자, 괄선문자 등이 망라되어 있습니다. 일부만 발췌해보면 아래와 같습니다.

B0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0		가	각	간	갈	갸	갓	갣	갤	갥	갦	갧	갨	갩	갪	갫
B0	갈	갬	갭	갱	갲	갳	갴	갵	갶	갷	갸	갹	갺	갻	갼	갽
C0	갾	갿	개	개	갠	갡	갢	갣	갤	갥	갦	갧	갨	갩	갪	갫

일본도 히라가나, 가타카나 등을 표현할 수 있어야 하므로 자신들의 문자셋을 정의하고 있고, 중국도 마찬가지로 자신들의 문자셋을 가지고 있습니다.

이렇듯 컴퓨터에서 문자를 표현하기 위해서는 어딘가에 문자셋이 정의되어 있어야 합니다.

¹ KSC 5601 이라고 공백없이 쓰기도 하는데요, KS와 C 사이에 공백을 두는 것이 맞습니다.

■ Character Encoding (encoding, 인코딩)

Character Encoding 은 줄여서 그냥 인코딩이라고 많이 얘기합니다.
 인코딩이란 "문자를 어떤 형태의 бай트로 변환시킬 것인가" 하는 것입니다.

앞서 보았던 ASCII 테이블을 다시 보겠습니다.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

빨간 사각형을 따라가보면 'A' 문자는 16 진수 값으로 0x41 입니다.
 즉, 컴퓨터에서 0x41 로 적으면 'A' 문자로 알아듣겠다는 것입니다.

KS C 5601 의 경우는 ASCII 보다는 좀 더 복잡합니다.

B0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0		가	각	간	감	갈	갸	갇	갈	갉	갊	갋	갌	갍	갎	갏
B0	갈	갇	강	개	객	갠	갡	갢	갣	갤	갥	갦	갧	갨	갩	갪
C0	갫	강	개	갠	갡	갢	갣	갤	갥	갦	갧	갨	갩	갪	갫	갬

위의 테이블을 읽는 방법을 '강' 문자로 설명드리자면,
 먼저 표의 좌측 상단 값을 취합니다. 테이블 구분 값입니다. (0xB0)
 두번째로 '강' 문자의 가로/세로 값을 더합니다. (0xC0 + 0x01 = 0xC1)
 그런 후 두 값을 순서대로 나열합니다. (0xB0C1)

이런 절차로 '강' 문자는 16 진수로 0xB0C1 입니다.
 ASCII 의 경우와 마찬가지로 컴퓨터에서 0xB0C1 이라고 적으면 '강' 문자로 알아
 듭니다. (당연히 한국에서만 알아듣습니다)

3. Unicode 개요

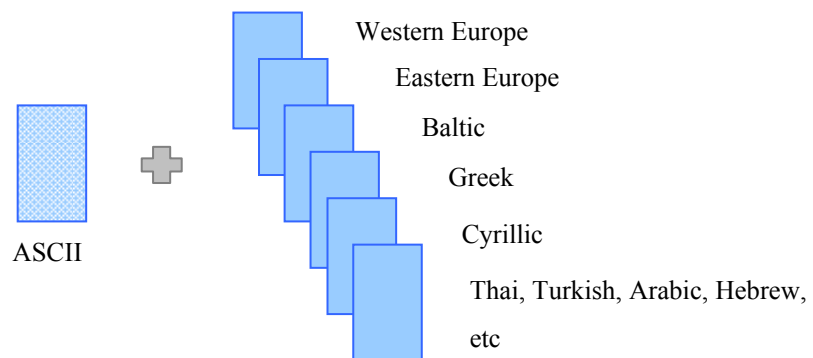
■ Unicode 이전의 문자셋

Unicode 가 나오기 전에는 각 나라마다 문자셋이 서로 달랐습니다.

예를 들어 독일에 있는 움라우트(Ä/ä, Ö/ö, Ü/ü)는 ASCII 에는 없는 문자입니다. 그래서 7비트짜리였던 ASCII 의 상위 한 비트를 더 쓰기로 했습니다. 그리고는 0~127 까지의 하위 128 개는 ASCII 의 것을 그대로 따르고, 상위 128 개 공간에 움라우트를 추가했습니다. 물론 움라우트 외에도 넣을 것이 많았을 겁니다.

이런 기본 아이디어를 그림으로 표현하면 아래와 같습니다.

상위 128 개 공간을 언어 권역별로 자신들만의 문자들로 정의한 것입니다.



그러다 보니 상위 128 개 문자들은 서로 충돌이 나기 때문에 독일에서 작성된 문서를 이스라엘에서 열어보면 알파벳이긴 한데 뭔가 이상한 문자들이 보이게 됩니다. 예를 들어 독일에서 *résumés*(이력서)를 이스라엘로 보내면 이스라엘에서는 *rsums* 라고 보일 것입니다.

■ Unicode 를 만든 이유

상황이 이렇다 보니 한 문서 내에서나, 한 시스템 내에서 여러 언어를 섞어쓰기가 쉽지 않았고, 언어권이 다른 두 지역 간에 문서를 주고 받을 수가 없었습니다. 학술적으로도 도서관과 같이 모든 언어권의 데이터가 전산화되어야 하는 곳에서도 필요성을 많이 느끼고 있었을 것입니다.

하지만 누구보다 강력히 필요성을 느낀 곳은 바로 미국의 소프트웨어 회사들이었을 것입니다. 지역별로 달라지는 인코딩 변환을 관리한다는 것은 매우 비용이 많이 드는 작업일 뿐만 아니라, 중국, 일본, 한국과 같이 문자 체계가 라틴 문자 계열과는 완전히 다른 동양권 시장을 공략하기에도 이 인코딩 문제는 큰 장벽이 되었기 때문입니다.

그래서 전 세계의 모든 문자를 표현할 수 있는 단 하나의 문자셋인 Unicode 를 만들게 되었습니다.

■ Unicode 의 시작

통일된 표준 문자셋을 만들기 위한 노력은 ISO(국제표준기구)에 의해 먼저 진행되었는데 ISO 10646 이란 국제 표준이 그것입니다. 이 국제 표준은 4 바이트 체계로서 모든 문자에 4 바이트를 할당하여 사용하자는 것이었습니다.

한편 4 바이트 체계에 대한 불만, 진행 절차에 대한 불만, 그리고 자신들의 OS 에 다국어 지원에 대한 부분을 강화하기 위해 Apple, Xerox, Sun, Microsoft, NeXT 등이 연합하여 1989 년 Unicode 컨소시엄을 조직했습니다. 당시 Unicode 컨소시엄에서 제안한 방식은 2 바이트 체계였습니다.

Unicode 컨소시엄은 ISO 에게 자신들의 표준을 사용할 것을 제안하고, 이 제안이 일부 받아들여지면서 1993 년 ISO 10646-1 이라는 국제 표준이 발표되었습니다. ISO 10646 표준의 정식 이름은 Universal (Multiple-Octet Coded) Character Set 입니다. 줄여서 UCS 라고 합니다.

ISO 와 Unicode 컨소시엄은 각자 목적이 약간 다르기 때문에 나름대로 자신들의 표준을 발전시켜 나가고 있습니다. 결국 Universal 문자셋에 관한 표준은 이 세상에 2 가지가 있는 것입니다. 그러나 다행히도 두 단체는 지속적인 협력 작업을 통해 문자셋에 들어가는 문자(종류, 코드포인트 등)에 대해서는 서로 일치시켜 나가고 있습니다.

2010 년 기준으로 최신 버전의 표준은 2009 년에 발표된 것으로서,

Unicode 컨소시엄은 Unicode 5.2

ISO 는 ISO/IEC 10646:2003 plus Amendments 1, 2, 3, 4, 5 and 6
입니다.

4. Unicode 구조

■ Unicode 의 기본 개념

간단히 말하자면 Unicode 는 전 세계의 문자에 특정 번호를 매겨서 테이블로 만들어 두겠다는 것입니다. 새로운 문자가 추가된다면 그저 새로운 번호를 매겨주면 됩니다. 이 번호를 코드 포인트 (Code Point) 라고 부릅니다.

코드 포인트 표기는 **U+0041** 과 같이 합니다.

U+ 는 Unicode 를 의미하고, 0041 은 코드 포인트 값으로서 16 진수로 적습니다.

U+0041 은 영어 알파벳 'A'를 의미하고, U+AC00 은 한글 '가'를 의미합니다.

코드 포인트는 그저 단순히 일련 번호는 아닙니다. 문자 종류별로 영역이 있어서 라틴 문자 범위, 한자 범위, 한글 범위 등 여러 블록으로 쪼개져 있습니다.

코드 포인트 영역에 대한 좀더 자세한 설명은 잠시후 살펴보도록 하고,

지금은 Unicode 에서 정의하고 있는 문자셋의 실례를 확인해 보겠습니다.

참고로 다음에 나오는 문자셋들은 아래 그림과 같이 세로 기준으로 나열되어 있습니다. 예를 들어 0000 은 NUL, 0030 은 '0', 0041 은 'A' 입니다.

	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073

먼저 U+0000-U+00FF 영역입니다. 이 영역은 ISO 8859-1 문자셋과 일치합니다.

ISO 8859-1 은 ASCII 확장 문자셋으로서 서유럽에서 많이 사용하는 알파벳을 상위 128 개 영역에 추가로 배치한 것입니다.

	000	001	002	003	004	005	006	007	008	009	00A	00B	00C	00D	00E	00F
0	NUL	DLE	SP	0	@	P	`	p	XXX	DCS	NB SP	°	À	Ð	à	ð
1	SOH	DC1	!	1	A	Q	a	q	XXX	PU1	¡	±	Á	Ñ	á	ñ
2	STX	DC2	"	2	B	R	b	r	BPH	PU2	¢	²	Â	Ò	â	ò
3	ETX	DC3	#	3	C	S	c	s	NBH	STS	£	³	Ã	Ó	ã	ó
4	EOT	DC4	\$	4	D	T	d	t	IND	CCH	¤	´	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u	NEL	MW	¥	µ	Å	Õ	å	õ
6	ACK	SYN	&	6	F	V	f	v	SSA	SPA	¦	¶	Æ	Ö	æ	ö
7	BEL	ETB	'	7	G	W	g	w	ESA	EPA	§	·	Ç	×	ç	÷
8	BS	CAN	(8	H	X	h	x	HTS	SOS	¨	¸	È	Ø	è	ø
9	HT	EM)	9	I	Y	i	y	HTJ	XXX	©	¹	É	Ù	é	ù
A	LF	SUB	*	:	J	Z	j	z	VTS	SCI	ª	º	Ê	Ú	ê	ú
B	VT	ESC	+	;	K	[k	{	PLD	CSI	«	»	Ë	Û	ë	û
C	FF	FS	,	<	L	\	l		PLU	ST	¬	¼	Ì	Ü	ì	ü
D	CR	GS	-	=	M]	m	}	RI	OSC	½	½	Í	Ý	í	ý
E	SO	RS	.	>	N	^	n	~	SS2	PM	¾	¾	Î	Þ	î	þ
F	SI	US	/	?	O	_	o	DEL	SS3	APC	¯	¿	Ï	ß	ï	ÿ

다음으로 한글은 U+AC00-U+D7AF 영역에 정의되어 있습니다.
영역이 너무 크기때문에 일부만 발췌해보면 아래와 같습니다.

	Hangul Syllables															
	AC00	AC01	AC02	AC03	AC04	AC05	AC06	AC07	AC08	AC09	ACA	ACB	ACC	ACD	ACE	ACFF
0	가	감	갠	갬	갈	각	갯	거	검	겐	갬	결	격	겟	고	곰
1	각	갑	갯	갹	갬	갯	갬	거	겁	갬	갹	결	겟	고	곰	곱
2	갯	갬	갹	갬	갹	갬	갹	갬	갹	갬	갹	갬	갹	갬	갹	갬
3	갬	갹	갬	갹	갬	갹	갬	갹	갬	갹	갬	갹	갬	갹	갬	갹
4	간	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬
5	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯
6	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯
7	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯
8	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯
9	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯
A	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯
B	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯
C	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯
D	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯
E	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯
F	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯	갬	갯

이런 식으로 전 세계의 많은 문자들이 할당되어 있습니다.
Unicode 전체 목록을 일부 나열해 보면 아래와 같습니다.

당연히 이 영역들을 다 알 필요는 없습니다. 단지 Unicode에 이렇게 많은 문자들이 할당되어 있다는 사실을 알려드리고자 합니다. 그리고 테이블 크기는 $0x10FFFF^2$ 로서 약 100 만 개 글자를 담을 수 있는 공간이란 점을 알면 더 좋습니다. (이 테이블에 실제로 할당된 문자는 약 10 만 개입니다)

그리고 목록을 자세히 보면 후반부에 비어 있는 영역들이 있는데 이는 실제로 문자가 할당되지 않은 빈 공간입니다. 앞으로 새로운 문자들이 이 곳에 할당될 것입니다.

처음	끝	문자 종류
0000	007F	제어 문자와 라틴 기본
0080	00FF	제어 문자와 라틴 보충
0100	017F	라틴 확장-A
0180	024F	라틴 확장-B
0250	02AF	국제 음성 기호 확장
02B0	02FF	조정 문자
0300	036F	조합 분음 기호(악센트)
0370	03FF	그리스어와 콥트어
0400	04FF	키릴 자모
0500	052F	키릴 자모 보충
0530	058F	아르메니아어
0590	05FF	히브리어
중략		
3040	309F	히라가나
30A0	30FF	가타카나
3100	312F	주음 부호
3130	318F	호환용 한글 자모
중략		
4E00	9FBF	한중일 통합 한자

² Unicode는 2 바이트 체계인 것 같은데 $0x10FFFF$ 라니요? 이상하죠?

이에 대한 내용은 다음절(Unicode 코드 체계)에서 설명하도록 하겠습니다.

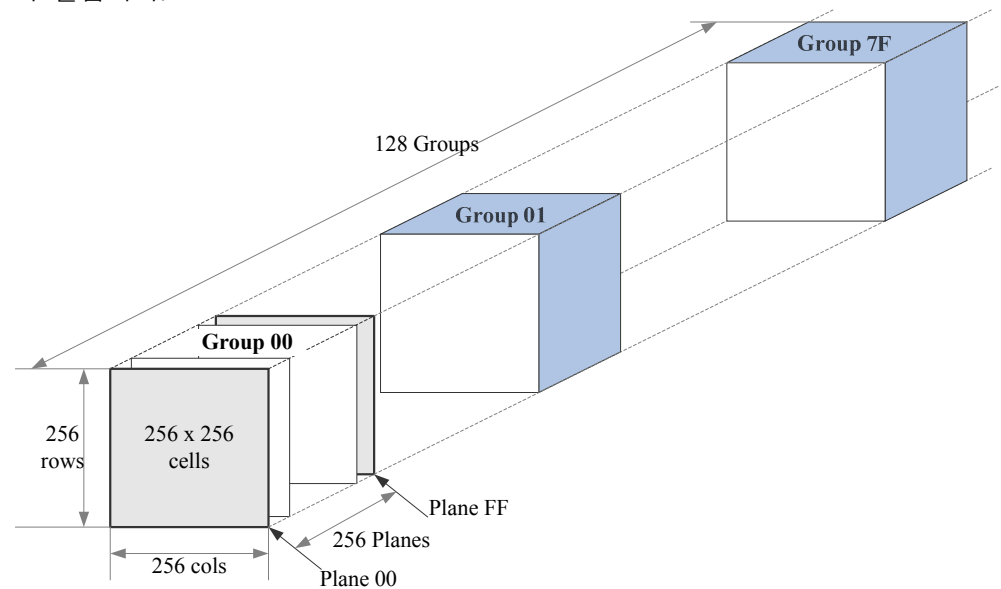
A000	A48F	이(Yi) 글자 마디
A490	A4CF	이(Yi) 부수
A700	A71F	어조 조정 문자
A720	A7FF	라틴 확장-D
A800	A82F	실헤티 나가리
A840	A87F	파스파 문자
AC00	D7AF	한글 글자 마디
D800	DBFF	상위 대체 영역 (High Surrogate) 문자 할당 안되어 있음
DC00	DFFF	하위 대체 영역 (Low Surrogate) 문자 할당 안되어 있음
중략		
FF00	FFEF	전각/반각 모양
FFF0	FFFF	특수 제어 문자
10000	1007F	선상 B 음절 문자
10080	100FF	선상 B 상형 문자
10100	1013F	에게(Aegean) 숫자
10140	1018F	옛 그리스 숫자
10300	1032F	옛 이탈리아 문자
중략		
1D100	1D1FF	악보용 기호
1D200	1D24F	고대 그리스 시대의 악보용 기호
1D300	1D35F	태현경 기호
1D400	1D7FF	수학식에서 쓰이는 알파벳
20000	2A6DF	한중일 통합 한자 확장-B
2F800	2FA1F	한중일 호환용 한자 보충
E0000	E007F	태그
E0100	E01EF	모양 구별 문자 보충
F0000	FFFFF	사용자 영역 보충-A
100000	10FFFF	사용자 영역 보충-B

[표 1] Unicode 범위 목록³³ 출처 : http://ko.wikipedia.org/wiki/유니코드_범위_목록

■ Unicode 코드 체계

Unicode 는 2 바이트 체계로 시작되었고, UCS 는 4 바이트 체계로 시작되었습니다. 하지만 두 체계가 완전히 다른 것이 아니라 Unicode 가 UCS 의 서브셋입니다. 따라서 UCS 의 구조에 대해 알게되면 Unicode 도 알 수 있습니다.

UCS 코드 체계에서는 모든 문자가 어떤 가상의 테이블 안에 할당됩니다. 이 테이블은 20 억 개의 문자가 수용될 수 있는 큰 테이블로서 구조는 아래 그림과 같습니다.



몇가지 새로운 용어가 보이는데요 그 의미는 아래와 같습니다.

용어	의미
Cell	한 개의 문자가 할당되는 공간
Plane (평면, 판)	256 x 256 (= 65536) 개 Cell 의 묶음 Plane 00 부터 Plane FF 까지 존재합니다.
Group	256 개 Plane 의 묶음 Group 00 부터 Group 7F 까지 존재합니다.

테이블은 총 128 개의 Group 으로 구성되고,
Group 마다 256 개의 Plane 이 존재하며,
각 Plane 은 65536 개의 cell 로 구성됩니다.
따라서 65536 x 256 x 128 하면 대략 20 억 개라는 숫자가 나옵니다.

그리고 이 테이블에서 최초 65536 개의 문자가 할당되는 영역을 특별히 BMP (Basic Multilingual Plane, 기본언어판)라고 부릅니다.

앞에서 Unicode 는 UCS 의 서브셋이라고 했습니다.

UCS 는 20 억 개의 문자가 할당될 수 있는 가상의 테이블을 정의하고 있습니다.

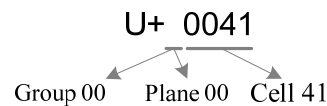
이에 반해 Unicode 는 Group00 의 Plane 00 ~ Plane 16 까지만 사용합니다.

즉 17 x 65536 하면 약 100 만 개의 문자가 수용될 수 있는 가상의 테이블을 사용하는 것입니다.

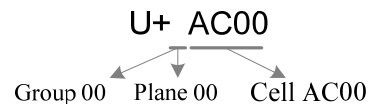
하지만 두 규격 모두 실제 할당된 문자는 서로 완전히 일치합니다.


테이블 구조에 대해 알아보았으므로, 이 구조를 토대로 코드 포인트를 다시 살펴볼 것입니다.

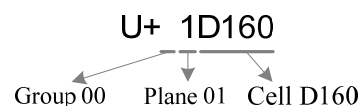
영어 알파벳 'A'는 U+0041 이라 했습니다. 0041 의 의미는 아래와 같습니다.



한글 '가'는 U+AC00 이라 했습니다. AC00 의 의미는 아래와 같습니다.



음표  는 U+1D160 입니다. 1D160 의 의미는 아래와 같습니다.



Unicode 에서는 Group 00 하나만 사용하므로 Group 표기는 생략합니다.

Plane 이 00 인 경우도 Plane 값은 생략하고 Cell 값만 표기합니다.

코드 포인트를 표현하기 위해서는 Plane 번호 5 비트, Cell 번호 16 비트하여, 21 비트의 공간이 필요합니다.

많은 분들이 Unicode 가 2 바이트(16 비트) 체계라고 알고 계시지만,

실제 Unicode의 코드 체계는 21 비트 체계라고 할 수 있습니다.

5. Unicode 인코딩

Unicode 인코딩이란 Unicode 문자에 매겨진 정수 숫자(=코드 포인트)가 일련의 바이트로 어떻게 표현되는가 하는 것입니다.

Unicode 에서는 서로 다른 3 가지 인코딩, UTF-32, UTF-16, UTF-8 을 제공합니다. 인코딩된 바이트 형태는 서로 다르지만 3 가지 인코딩 모두 Unicode 문자 전부를 표현할 수 있으므로 인코딩간 변환을 거쳐도 데이터 손실은 발생하지 않습니다.

각 인코딩에 대한 상세한 설명에 들어가기 전에 샘플을 먼저 보시죠.

A	Ω	語	Ⅲ	UTF-32
00000041	000003A9	00008A9E	00010384	
A	Ω	語	Ⅲ	UTF-16
0041	03A9	8A9E	D800 DF84	
A	Ω	語	Ⅲ	UTF-8
41	CE A9	E8 AA 9E	F0 90 8E 84	

UTF-32 는 모든 문자를 코드 포인트 값을 그대로 유지하면서 32 비트로 만듭니다.

UTF-16 은 대부분의 문자는 코드 포인트 값을 그대로 유지하면서 16 비트로 만들지만, BMP 범위를 벗어나는 문자는 특별한 변환을 거쳐 32 비트로 인코딩합니다. (위의 그림에서 나타나듯이 같은 32 비트라도 UTF-16 과 UTF-32 의 32 비트는 값이 다릅니다)

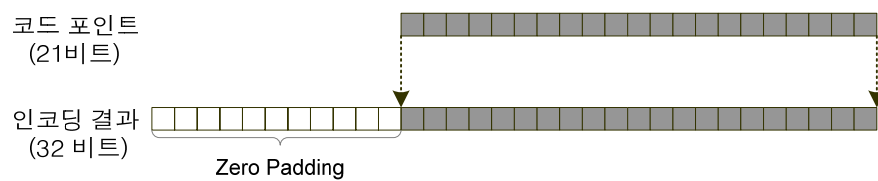
UTF-8 은 코드 포인트의 범위에 따라 1, 2, 3, 4 바이트로 서로 달리 인코딩됩니다.

■ UTF-32

UTF-32 는 가장 간단한 인코딩 방법입니다.

모든 코드 포인트가 그대로 32 비트 1:1 매핑됩니다.

앞에 남은 11 개 비트는 0 으로 채워집니다.



인코딩된 값 자체가 코드 포인트이기 때문에 해당하는 Unicode 문자를 찾기도 쉽습니다. 이에 반해 UTF-16 이나 UTF-8 에서는 인코딩된 값에 대해 특정한 변환 과정을 거쳐야만 문자를 알아낼 수 있는 경우가 있습니다.

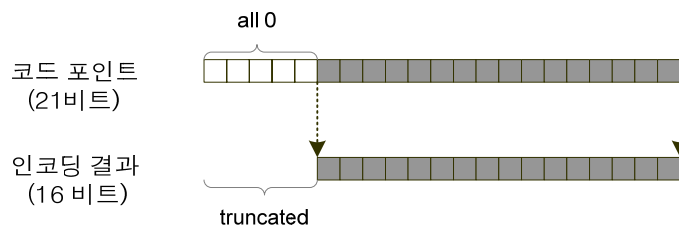
하지만 아무래도 다른 인코딩보다는 공간을 많이 차지하기 때문에 메모리 공간에 큰 제약이 없는 상황이거나 고정폭 인코딩이 특별히 필요한 경우에 유용합니다.

주로 Unix 환경에서 문자열을 메모리에서 다룰 때 사용되고 있습니다.

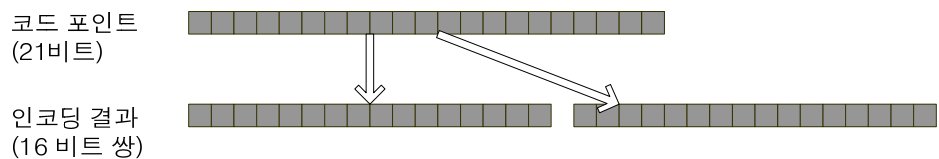
■ UTF-16

UTF-16 은 BMP 영역에 최적화된 인코딩 방법입니다.

BMP 영역인 U+0000-U+FFFF 범위의 문자에 대해서는 16 비트로 1:1 매핑됩니다.



BMP 영역을 벗어나는(U+10000-U+10FFFF) 문자에 대해서는 특별한 변환 규칙에 따라 32 비트로 인코딩됩니다. (변환 규칙에 대한 설명은 다음 절에서 하겠습니다)



사실 실제로 많이 사용되는 현대 문자의 99%는 BMP 영역에 있다고 합니다.

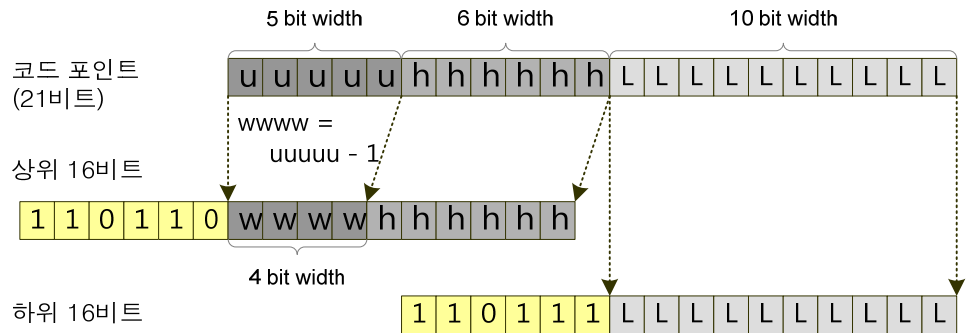
따라서 UTF-16 은 UTF-32 보다 저장 공간이 적게 들면서도, 인코딩된 값에서 Unicode 문자를 찾기에 비교적(!) 수월하므로 효율적인 인코딩이라 할 수 있습니다.


역사적으로 보면 처음 제안된 Unicode 는 2 바이트 체계였기 때문에 UTF-16 은 원래 16 비트 고정폭의 인코딩이었습니다. 세월이 흘러 많은 코드 포인트를 표현하려다 보니 위와 같이 가변폭의 인코딩으로 바뀐 것입니다.

주로 Java 플랫폼, Microsoft 플랫폼에서 문자열을 다룰 때 사용되는 인코딩입니다.

▪ BMP 영역 밖의 문자에 대한 UTF-16 변환 규칙

BMP 영역을 벗어나는 U+10000-U+10FFFF 의 문자에 대한 변환 규칙을 좀더 자세히 살펴보면 아래 그림과 같습니다.



위의 변환 규칙에 따라 음표  에 대해 실제로 UTF-16 인코딩을 해보면 아래와 같습니다.

구분	16 진수	2 진수
코드 포인트	1D160	00001 1101 0001 0110 0000
인코딩 (상위 16 비트)	D834	1101 1000 0011 0100
인코딩 (하위 16 비트)	DD60	1101 1101 0110 0000
인코딩 결과 (32 비트)	D834 DD60	

참고로 UTF-16 과 관련된 새로운 용어를 한가지 말씀드리겠습니다.

위의 변환 규칙은 인코딩된 16 비트 값이 0xD800~0xDFFF 사이의 값이 되도록 고안되었습니다. Unicode 코드 포인트 중 U+D800-U+DFFF 영역을 특별히 Surrogate 라 부르고, 인코딩된 상/하위 비트를 합쳐서 Surrogate Pair 라고 부릅니다.

보다시피 0xD800-0xDFFF 는 UTF-16 인코딩의 결과 값이 될 수 있으므로 Surrogate 영역에는 문자가 할당되어 있지 않습니다.

■ UTF-8

인코딩 효율성만 놓고 본다면 UTF-16 이 좋기는 한데 ASCII 문자마저도 2 바이트가 된다는 점이 문제였습니다.

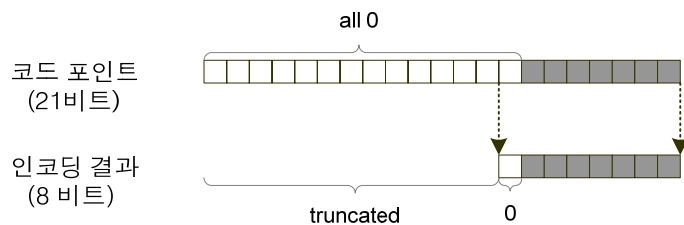
"ABC" 를 UTF-16 으로 인코딩하면 <00 41 00 42 00 43>이 됩니다. 문자열 중간에 널(null, 0)이 들어가버리므로 문자열 관련 C 함수들이 정상적으로 동작하지 않게 됩니다.

따라서 UTF-16 인코딩을 사용하려면 기존의 ASCII 기반의 프로그램을 모두 다시 작성해야 한다는 얘기가 되는데, 이는 현실적으로 거의 불가능한 것입니다. 특히 UNIX 및 LINUX 는 콘솔 기반 프로그램이 대부분이므로 그 정도가 더 심합니다.

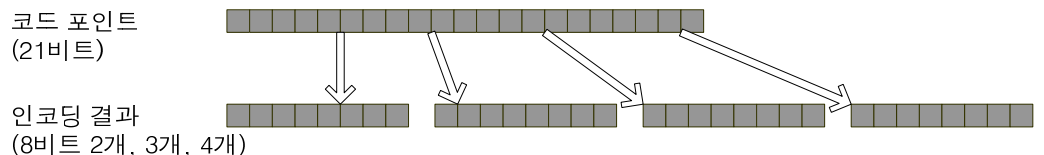
이 문제를 피하기 위해 UTF-8 이 제안되었습니다. 이 인코딩은 ASCII 호환성을 주목적으로 하기 때문에 중간에 널(null, 0)이 들어가지 않도록 고안되었습니다.

먼저 U+0000-U+007F 영역에 대해서는 1 바이트로 인코딩됩니다.

즉 ASCII 문자셋에 해당하는 값은 1 바이트가 됩니다.



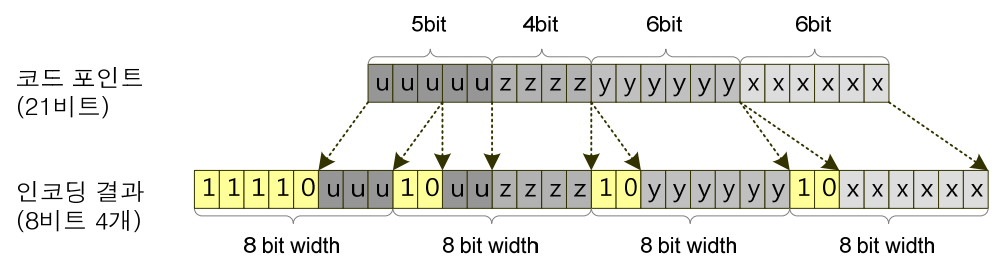
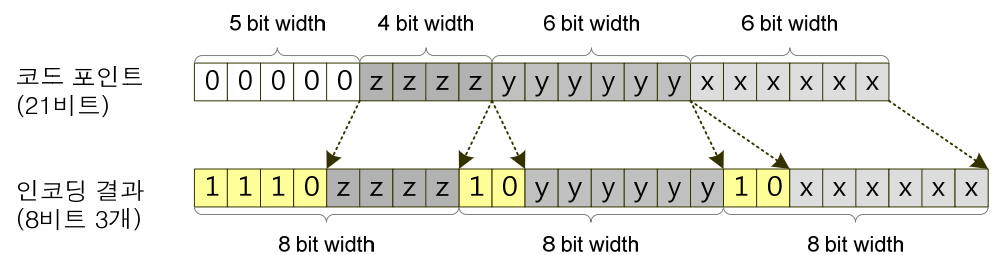
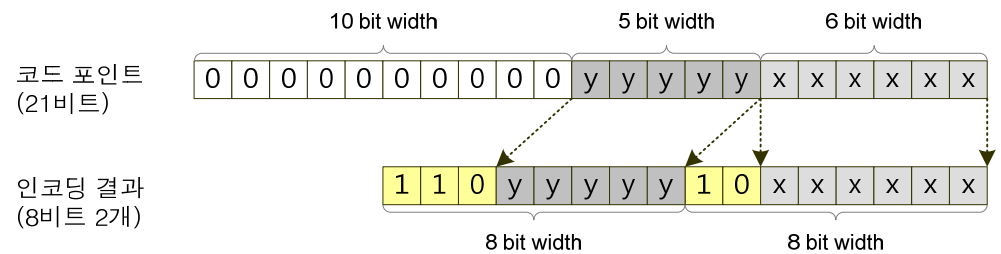
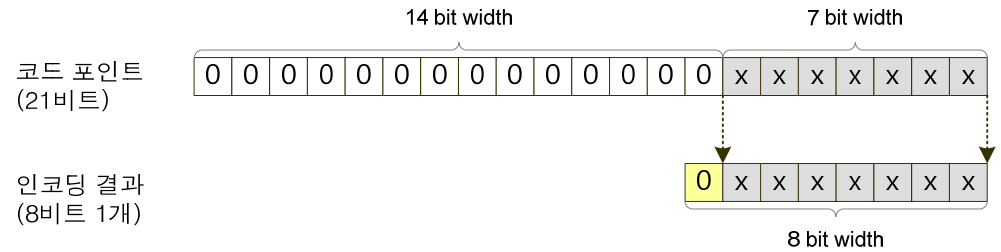
더 큰 코드 포인트에 대해서는 2 바이트, 3 바이트 혹은 4 바이트로 인코딩됩니다. (변환 규칙에 대한 설명은 다음 절에서 하겠습니다.)



UTF-8 은 HTML 문서나 XML 문서의 인코딩에 적합합니다. 실제로 2010 년 현재, Web 상에서 UTF-8 은 실질적인 표준 인코딩으로 자리잡고 있습니다.

▪ UTF-8 변환 규칙

UTF-8 는 코드 포인트의 범위에 따라 아래와 같이 다양하게 인코딩합니다.



UTF-8 인코딩에 대해 끝내기 전에 간단한 퀴즈입니다.

Q: 한글 '가'는 UTF-8 로 인코딩하면 몇 바이트가 될까요?

A: 3 바이트입니다.

이유를 아셔야 하는데요, 한번 생각해 보시길 바랍니다.

6. 마치며

이로써 Unicode 에 대한 간단한 설명을 마칩니다.

지금 당장 Unicode 프로그래밍을 하는 개발자 입장에서 보면 본 문서의 내용이 별로 도움이 안될지도 모르겠습니다. 여기에는 Unicode 프로그래밍에 관련된 내용이 없기 때문입니다.

Unicode 프로그래밍은 ASCII 일 때와 어떤 점이 다른지,
인코딩 변환을 위해서는 어떤 라이브러리가 있는지,
Database 는 어떻게 설정을 해줘야 Unicode 를 정상적으로 사용할 수 있는지,
인코딩이란 것을 보니 오래된 것부터 새 것까지 너무 많은데 이게 다 뭔지,
등 많은 질문이 있을 수 있습니다.

하지만 이런 현실적인 질문에 대한 답을 찾아가다 보면, 결국은 Unicode 자체에 대한 이해가 필요한 시점이 있습니다. 본 문서가 그런 시점에서 최소한의 가이드 역할을 할 수 있으면 좋겠습니다.

고맙습니다.

Appendix A. 문서정보

A.1 문서이력

- 문서목적
Unicode 에 대한 기본적인 이해를 돕는다.
- 작성자
김정환 / john@novonetworks.com
- 작성이력
최초작성 : 2010-06

Appendix B. References

Unicode Standard 5.2 (<http://www.unicode.org/versions/Unicode5.2.0>)

Unicode 매뉴얼입니다. 10MB 짜리 PDF 를 다운로드 받을 수 있습니다.
Unicode 에 관한 가장 정확한 정보는 이 문서에 있습니다.

Unicode Charts (<http://www.unicode.org/charts/>)

지역 카테고리에 따라 분류된 Unicode 문자를 확인할 수 있습니다.
코드 포인트별로 확인하려면 <http://rudhar.com/lingtics/uniclnks.htm>

Unicode Character Search (<http://www.fileformat.info/info/unicode/char/search.htm>)

코드 포인트로 Unicode 를 검색해주는 사이트
글자 모양, UTF-32, UTF-16, UTF-8 인코딩 결과 등을 한눈에 볼 수 있습니다.

WWW의 국제화와 한국화 (<http://www.w3c.or.kr/i18n/www-i18n>)

1996 년 11 월 29 일, 한국과학기술원 경영정보연구센터, 정주원
Unicode 의 코드 체계에 대한 설명을 볼 수 있습니다.
Unicode 뿐만 아니라 한글 관련 인코딩도 간단히 설명하고 있습니다.
Web 기술에서 국제화와 관련된 부분에 집중하여 정리되어 있습니다.

학술 데이터베이스의 유니코드 변환 적용에 관한 연구

2003 년 12 월, 한국교육학술정보원, 조순영 외
한국교육학술정보원(KERIS)에서 제공하는 학술연구정보서비스(RISS)를
Unicode 체계로 전환하기 위한 연구 보고서입니다.

Unicode 범위 목록 (http://ko.wikipedia.org/wiki/유니코드_범위_목록)

[4. Unicode 의 기본 개념]에서 인용했던 Unicode 범위 목록입니다.

조엘 온 소프트웨어 블로그 (<http://www.joelonsoftware.com/articles/Unicode.html>)

Unicode 에 대해 명쾌하게 정리해주고 있습니다.

Introduction to Unicode (<http://www.pan110n.net/Presentations/Afghanistan/Unicode.pdf>)

by Atif Gulzar Center for Research in Urdu Language Processing