UTF-8 and Unicode FAQ for Unix/Linux

유닉스/리눅스 사용자를 위한 UTF-8 및 유니코드 관련 FAQ

Markus Kuhn(*Markus.Kuhn@cl.cam.ac.uk* <mailto:Markus.Kuhn@cl.cam.ac.uk>) 16 March 2001

국봉관 역(*kook@hanyang.co.kr* <mailto:kook@hanyang.co.kr>) 2001년 3월 29일

이 문서는 유닉스 혹은 리눅스 환경에서 UTF-8과 유니코드를 사용하는 방법에 관한 내용을 담고 있습니다. 번역 상의 실수 혹은 오타를 발견하신 분은 메일로 연락주시기 바랍니다.

목 차

제 1 절 UCS와 ISO 10646은 무엇인가? 2

제 2 절 결합 문자(Combining Characters)란 무엇인가? 3

제 3 절 UCS 구현 레벨(UCS implementation levels)은 무엇인가? 3

제 4 절 UCS는 국제 기준으로 채택되었는가? 3

제 5 절 유니코드란 무엇인가? 4

제 6 절 유니코드와 ISO 10646 사이의 차이점은 무엇인가? 4

제 7 절 UTF-8은 무엇인가? 4

제 8 절 유니코드를 지원하는 프로그래밍 언어는 무엇인가? 6

제 9 절 리눅스에서 유니코드를 어떻게 사용하는가? 7

제 10 절 소프트웨어를 어떻게 수정해야만 하는가? 8

제 11 절 유니코드와 UTF-8을 위한 C의 지원 8

제 12 절 UTF-8 모드는 어떻게 활성화되어야 하는가? 9

제 13 절 UTF-8을 지원하는 X-term 버전은 어떻게 구하는가? 9

제 14 절 xterm은 얼마나 많은 유니코드를 지원하는가? 10

제 15 절 ISO-10646의 X11용 폰트는 어디서 구할 수 있는가? 11

제 16 절 UTF-8 터미널 에뮬레이터와 관련된 이슈는 무엇인가? 12

제 17 절 UTF-8이 가능한 지금 사용 중인 애플리케이션은 어떤 것이 있는가? 13

제 18 절 UTF-8을 지원하기 위해서 사용가능한 패치는 무엇인가? 14

제 19 절 유니코드를 다루는 사용가능한 비 상용(free) 라이브러리는 있는가? 14

제 20 절 UTF-8을 지원하기 위해 어떤 패키지가 현재 개발 중인가? 15

제 21 절 솔라리스 상에서 UTF-8을 위한 지원은 어떻게 동작하는가? 15

제 22 절 포스트 스크립트 glyph 명칭(Postscript glyph names)>은 어떻게 UCS 코드와 관련되어 있는가? 15

제 23 절 잘 정의된 UCS subset은 있는가? 15

제 24 절 X11 R6.4 스펙과 유니코드에 어떤 문제점이 있는가? 16

제 25 절 이런 문제에 관한 좋은 메일링 리스트는 있는가? 18

제 26 절 더욱 상세한 참고 자료 19

제 1 절 UCS와 ISO 10646은 무엇인가?

ISO 10646국제 표준은 **Universal Character Set(UCS)**를 정의하고 있다. UCS는 모든 다른 종류의 문자셋 표준(character set standards)의 상위에 존재하는 문자셋이다. 이것은 다른 문자셋과의 상호 호환성을 보증한다. 만약 어떤 텍스트 문자열을 UCS로 변환하고 다시 원래의 인코딩으로 변환할 경우 어떤 정보도 손실되지 않을 것이다.

ISO 10646은 공식적으로 31비트 문자셋을 정의하고 있다. 그러나 지금까지 문자들은 이러한 큰 코드 공간(of this huge code space)중에서도 오직 처음에서부터 65534번째 위치(0x0000 부터 0xFFFD까지)까지에만 위치했었다. 이러한 UCS의 16비트 서브셋은 기본적인 다국언어용 영역**(Bagic Multilingual Plane : BMP)** 혹은 영역 0(Plane 0)라고 부른다. BMP 영역을 벗어나 인코딩될 것이 예상되는 문자들은 역사적 혹은 과학적 목적으로 단지 전문가들만이 사용하는 약간 색다른 언어에 모두 포함된다(예: 상형문자). 현재 계획은 0x000000 부터 0x10FFFF까지의 21비트 코드 공간의 외부에 할당되는 문자들은 없을 것이라고 제안하고 있다. 이것은 백만개를 조금 넘는 잠재성있는 미래의 문자들을 지원할 것이다. ISO 10646-1 기준은 1993년에 최초로 제안됐으며, 문자의 구조와 BMP 영역의 내용을 정의하고 있다. BMP 영역의 외부에 인코딩되는 문자들을 정의하고 있는 두번째 파트인 ISO 10646-2는 준비 중에 있으나, 그것이 완성되기까지는 수년이 걸릴지도 모른다. 기반 문자에 이어 끊임없이 새로운 문자들이 BMP 영역에 포함되고 있지만, 현재 존재하고 있는 문자들은 결코 변하지 않을 것이며 안정성을 확보하고 있다.

UCS는 각각의 문자에 코드 번호 뿐만 아니라 공식 명칭도 할당하고 있다. UCS 혹은유니코드 값을 나타내는 16진수는 일반적으로 "라틴 대문자 A"를 나타내는 U+0041처럼 앞에 "U+"라는 접두사가 붙는다. UCS 문자 U+0000 부터 U+007F는 US-ASCII(ISO 646 IRV)에서와 같은 의미를 갖는다. 그리고 U+0000 부터 U+00FF까지의 범위는 ISO 8859-1(Latin-1)에서와 같다. U+E000에서부터 U+F8FF까지의 범위와 BMP 영역 외부의 더 큰 범위는 개인적인 용도를 위해 보존된다.

UCS 기준의 완전한 명칭은 다음과 같다.

```
International Standard ISO/IEC 10646-1, Information technology --
Universal Multiple-Octet Coded Character Set (UCS) -- Part 1:
Architecture and Basic Multilingual Plane. Second edition,
International Organization for Standardization, Geneva, 2000-09-15.
```

이것은 PDF 파일로 저장된 CD-ROM 세트로 80 스위스프랑(54 유로화, 45 미국달러, 32 영국파운드)에 *ISO로부터 온라인으로 주문* <http://www.iso.ch/cate/d29819.html> 할 수 있다.

제 2 절 결합 문자(Combining Characters)란 무엇인가?

UCS에서 몇몇 code point들은 결합 문자**(combining characters)**에 할당되었 다. 이것들은 타자기에서 공간을 차지하지 않는 액센트 키와 같다. 결합 문자는 그 자 체로는 하나의 완전한 문자가 아니다. 그것은 앞서는 문자에 더하는 액센트거나 혹은 구분 마크이다. 이런식으로, 어떤 문자에 어떤 액센트를 놓는 것이 가능하다. 일반적인 언어의 철자법에서 사용하는 문자처럼 가장 중요한 액센트를 가진 문자들은 옛 문자 셋을 가진 구 버전과의 호환성을 확보하기 위해서 UCS에서 그들 자신만의 코드를 갖는다. 미리 만들어진 문자**(precomposed characters)**라고 알려진 액센트를 가진 문자들은 자신만의 코드 위치를 갖지만, 또한 결합 문자에 뒤따르는 한쌍의 다른 문자로써 나타낼 수 있다. 미리 만들어진 문자들은 어떠한 결합 문자도 갖지 않는 ISO 8859와 같은 옛 방식의 인코딩과의 호환성을 위해서 UCS에서 사용 가능하다. 결합문자의 메카니즘은 어떤 문자에 액센트나 다른 구분 기호를 붙이는 것을 허락하는데, 이 것은 특히 기본 문자와 한가지 혹은 몇가지의 구분 기호와의 결합이 필요한 수학 방정 식과 국제 표음 알파벳과 같은 과학 표기법을 위해서 중요하다.

결합문자는 그들이 수정하는 문자를 따른다. 예를 들면, 독일 umlaut 문자 Ä는 미리 만들어진 UCS 코드 U+00C4로 나타내거나 대안적으로 "결합 부음 부호"(combin ing diaeresis)의 뒤를 잇는 일반적인 "라틴 대문자 A"의 결합으로 나타낼 수 있는데, U+0041 U+0308와 같다. 몇몇의 결합 문자는 다수의 액센트를 위에 놓거나 기본 문자의 위아래 모두에 결합 마크를 더 할 필요가 있을 때 적용할 수 있다. 타이 문자를 예로 들면, 하나의 기본 문자 위에 결합 문자가 최대 2개까지 필요하다.

제 3 절 UCS 구현 레벨(UCS implementation levels)은 무엇인가?

모든 시스템들이 결합 문자와 같은 UCS의 모든 진보된 메카니즘을 지원하리라고 기 대할 수 없다. 그러므로, ISO 10646은 다음의 세가지 구현 레벨을 명시하고 있다.

- 레벨 1 : 결합 문자와 한글 자모(두개 혹은 세개의 작은 문자들로 한글 음절을 이루는 특별하며 더욱 복잡한 한국 표음 문자의 인코딩)는 지원하지 않는다.
- 레벨 2 : 레벨 1과 거의 같으나 몇몇 서체(script)에 있어서 결합문자의 고정된 목록(fixed list)을 허용한다(예를 들면, 유태어, 아랍어, 인도어, 방글라데시어, 네팔어, 인도-아리안어, 오리사-인도어, 타밀어, 안드라어, 카르나타카어, 말레이시아 어, 태국어 및 라오스어가 있다). 이러한 문자들은 최소한의 결합 문자들을 위한 지원 없이는 UCS에서 적절하게 나타낼 수 없다.
- 레벨 3 : 모든 UCS 문자들을 지원한다. 예를 들면 수학자들은 어떤 수학 문자상의 틸데나 화살표(혹은 양쪽다)를 나타낼 수 있다.

제 4 절 UCS는 국제 기준으로 채택되었는가?

대답은 예이다. 수많은 국가들이 1993년에 ISO 10646-1:1993을 국가적으로 채택할 것을 공표했으며 때로는 예전에 국가에서 구현한 여러가지 사항과 기준에 대한 상호 참고 조항에 부가적인 조항을 붙인 후에 공표했다.

- 중국: GB 13000.1-93
- 일본: JIS X 0221-1995
- 한국: KS X 1005-1:1995 (ISO 10646-1:1993의 수정안 1-7을 포함)

제 5 절 유니코드란 무엇인가?

역사적으로, 단일 통합시킨 문자셋을 만들려는 두개의 독립적인 시도가 있었다. 하 나는 *국제 표준 기구(ISO)* <http://www.iso.ch/>의 ISO 10646 프로젝트였으며, 다른 하나는 다중 언어 소프트웨어 제조사들의(초기에는 미국회사가 대부분이었 던) 컨소시움으로 구성된 *유니코드 프로젝트* <http://www.unicode.org/>였다. 운 좋게도, 두 프로젝트에 참여했던 참가회사들 모두 1991년경에 두개의 서로 다른 통합 문자셋은 세계가 원하는 바가 아니라는 것을 깨달았다. 그들은 함께 노력했으며 단일한 코드 테이블을 만들기 위해 함께 작업했다. 양 프로젝트 모두 여전히 존재하며 그들 각자의 기준을 독립적으로 공표한다. 그러나 유니코드 컨소시움과 ISO/IEC JTC1/SC2는 호환가능한 유니코드와 ISO 10646 기준의 코드 테이블을 유지하기로 합의했다. 그리고 그들은 더욱 발전적인 확장 기능을 위해 면밀히 협력하고 있다. 유니코드 1.1은 과거 ISO 10646-1:1993에 대응했고, 유니코드 3.0은 현재 ISO 10646-1:2000에 대응한다.

유니코드 표준은 몇몇 일반적인 책처럼 *amazon.com* <http://www.amazon.com/exec/obidos/ASIN/0201616335/mgk25>으로부터 약 50 달러에 주문할 수 있다.

> The Unicode Consortium: *The Unicode Standard, Version 3.0* <http://www.amazon.com/exec/obidos/ASIN/0201616335/mgk25>, Reading, MA, Addison-Wesley Developers Press, 2000, ISBN 0-201-61633-5.

텍스트 프로세싱과 문자셋을 가지고 자주 작업을 한다면, 여러분들은 반드시 한 카 피를 구입해야만 한다.

제 6 절 유니코드와 ISO 10646 사이의 차이점은 무엇인가?

유니코드 컨소시움이 공표한 *유니코드 표준* <http://www.unicode.org/unicode/standard/standard.html>은 실제로 구현 레벨 3에서 기본 다중언어용 영역(BMP)을 포함한다. 두 표준 공히 모든 문자들은 같은 위치를 가지며 같은 명칭을 사용한다.

유니코드 표준은 부가적으로 몇몇 문자와 관련된 훨씬 많은 언어 체계를 정의하고 있으며 일반적으로 양질의 인쇄 출판 시스템 구현을 위한 더 나은 참고 자료가 된다. 유니코드는 예를 들어 라틴어와 유태어를 혼합하는 양 방향 텍스트를 취급하므로써, 몇몇 언어의 프리젠테이션 양식을 렌더링하기 위한 알고리즘과 문자열 비교를 위한 알 고리즘 및 그 외 많은 것을 명시하고 있다.

다른 한편 ISO 10646 표준은 잘 알려진 ISO 8859 표준과 비교했을 때 간단한 문자셋 테이블 그 이상은 아니다. 이것은 표준과 관련된 몇몇 기술들을 명시하고, 몇몇 인 코딩 대안들을 정의하며, ISO 6429와 ISO 2022와 같은 다른 ISO 표준과 관련된 UCS를 사용하는 방법에 관한 세부 사항을 포함한다. ISO 표준과 밀접하게 관련된 다른 것들 도 있다. 예를 들면, UCS 문자열의 정렬에 관한 ISO 14651이 있다. ISO 10646-1 표준의 훌륭한 특징으로는 그것이 다섯가지 다른 스타일로 변형시켜 한중일 국가의 glyph 예를 제공한다는 것이다. 반면 유니코드 표준은 한중일 국가의 한자를 단지 중국 식으 로만 보여준다.

제 7 절 UTF-8은 무엇인가?

무엇보다 먼저 UCS와 유니코드는 단지 정수를 문자에 할당하는 코드 테이블일 뿐이 라는 것이다. 그러한 문자 혹은 문자 각각의 정수 값의 시퀀스가 어떻게 바이트 시퀀스로 나타날 수 있는 지에 대한 몇 가지 대안들이 존재한다. 대단히 이해하기 쉬운 두 개의 인코딩은 유니코드 텍스트를 2 혹은 4바이트 시퀀스의 시퀀스(sequences of eit her 2 or 4 bytes sequences)로써 저장한다. 이러한 용어에 관한 공식적인 명칭은 각 각 UCS-2와 UCS-4이다. 다른 방법으로 명시되지 않는다면, 가장 중요한 바이트가 이들의 첫번째로 온다(Bigendian convention). ASCII 또는 Latin-1 파일은 모든 ASCII 바이트의 앞에 0x00 바이트를 삽입하므로써, UCS-2 파일로 변환시킬 수 있다. UCS-4 파일을 원한다면, 모든 ASCII 바이트 앞에 그 대신에 세개의 0x00 바이트를 삽입해야만 한다.

U-00000000 - U-0000007F:	0xxxxxxx
U-00000080 - U-000007FF:	110xxxxx 10xxxxxx
U-00000800 - U-0000FFFF:	1110xxxx 10xxxxxx 10xxxxxx
U-00010000 - U-001FFFFF:	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
U-00200000 - U-03FFFFFF:	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
U-04000000 - U-7FFFFFFF:	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

유닉스 환경에서 UCS-2(또는 UCS-4)를 사용하는 것은 매우 심각한 문제점을 불러온 다. 이러한 인코딩을 가진 문자열들은 파일명과 C 라이브러리 함수 파라미터에서 특별 한 의미를 갖는 '\0' 혹은 '/'와 같이 매우 광범위한 문자 바이트를 부분적으로 포함할 수 있다. 이에 더해, 대다수의 유닉스 툴들은 ASCII 파일을 예상하며, 큰 수정이 없이는 16비트 단어들을 문자로 읽을 수 없다. 이러한 이유 때문에 **UCS-2**는 파일명과 텍스트 파일 및 환경 변수 등에 적합한 **유니코드**의 외부 인코딩(suitable external encoding of Unicode)이 아니다.

ISO 10646-1 의 *Annex R* <http://www.cl.cam.ac.uk/~mgk25/ucs/ISO-10646-UTF-8.html>과 *RFC 2279* <ftp://sunsite.doc.ic.ac.uk/packages/rfc/rfc2279.txt>상에 정의된 **UTF-8** 인코딩은 이러한 문제점들이 없다. 이것은 유닉스 스타일의 운영 체제하에서 **유니코드**를 사용하기 위한 의심할 여지없이 좋은 방법이다.

UTF-8은 다음의 성질을 갖고 있다:

- U+0000부터 U+007F까지의 UCS 문자들은 0x00에서 0x7f 바이트까지 쉽게 인코딩된다(ASCII와의 호환성). 이것은 오직 7비트의 ASCII 문자들을 포함하는 파일 및 문자열들이 ASCII와 UTF-8 양쪽 모두에서 같은 인코딩을 갖는다는 것을 의미한다.
- U+007F보다 큰 모든 UCS 문자들은 각각 독자적인 바이트의 시퀀스로써 인코딩되며, 이것들은 각각 가장 중요한 비트셋(bit set)을 가진다. 그러므로 다른 문자의 부분에 어떤 ASCII 바이트(0x00-0x7f)도 나타날 수 없다.
- ASCII가 아닌 문자를 나타내는 멀티바이트 시퀀스의 첫번째 바이트는 항상 0xC0에서부터 0xFD 범위에 있으며, 그것은 이러한 문자를 위해 얼마나 많은 바이트가 필요한 지를 가르킨다. 멀티바이트 시퀀스의 모든 이후의 바이트들은 0x80에서부터 0xBF 범위에 있다. 이 때문에 resynchronization을 쉽게할 수 있고 국가에 구애받지 않고 인코딩할 수 있으며 바이트를 잃어버리지 않게 된다.
- 가능한 모든 $2^{31} UCS.UTF - 86,16 BMP3$.

- Bigendian UCS-4 바이트 문자열의 정렬 순서는 보존된다.
- 0xFE 및 0xFF 바이트는 결코 UTF-8 인코딩에서 사용하지 않는다.

다음의 바이트 시퀀스는 한 문자를 나타내기 위해 사용한다. 사용되는 시퀀스는 그 문자의 유니코드 번호에 따라 달라진다.

*xxx*비트의 위치는 이진 표기법에 의해 문자 코드 번호의 비트들로 채워진다. 오른쪽의 x비트는 별로 중요하지 않다. 그 문자의 코드 번호를 나타내는 오직 가장 짧은 멀티바이트 시퀀스만 사용할 수 있다. 멀티바이트 시퀀스에서 첫 번째 바이트의 왼쪽 1비트의 수는 전체 시퀀스에서의 바이트 수와 같다는 점에 주의하라.

예: "유니코드 문자 `U+00A9 = 1010 1001`"(저작권 부호)는 다음과 같은 UTF-8에 따라 인코딩된다.

```
11000010 10101001 = 0xC2 0xA9
```

그리고 문자 `U+2260 = 0010 0010 0110 0000`(저작권 부호)는 다음과 같은 UTF-8에 따라 인코딩된다.

1100000x (10xxxxxx)
11100000 100xxxxx (10xxxxxx)
11110000 1000xxxx (10xxxxxx 10xxxxxx)
11111000 10000xxx (10xxxxxx 10xxxxxx 10xxxxxx)
11111100 100000xx (10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx)

```
11100010 10001001 10100000 = 0xE2 0x89 0xA0
```

이러한 인코딩의 공식 명칭과 정확한 표기는 UTF-8이며, UTF는 **U**CS **T**ransformation **F**ormat을 의미한다. utf8혹은 UTF_8과 같은 다른 방법으로 UTF-8을 문서에 쓰지마라. 물론 인코딩 자체를 참조하지 않고 변수명에 참조할 경우에는괜찮다.

UTF-8의 디코딩 처리 순서에 있어서 중요한 점은 다음과 같다: 보안상의 이유 때문에, UTF-8 디코더는 한 문자를 인코딩하기 위해서 필요 이상으로 긴 UTF-8 시퀀스를 받아들여서는*안 된다* <http://www.unicode.org/unicode/uni2errata/UTF-8_Corrigendum.html>. 예를 들어 U+000A(라인 피드) 문자는 **오직** 0x0A 형식으로 UTF-8 스트림으로부터 받아들여야만 하며, 다음의 다섯가지와 같이 과도하게 긴(overlong) 형식으로 받아들여서는 안된다.

```
0xc0 0x8A
0xe0 0x80 0x8A
0xf0 0x80 0x80 0x8A
0xf8 0x80 0x80 0x80 0x8A
0xfc 0x80 0x80 0x80 0x80 0x8A
```

가장 짧은 인코딩을 찾기 위한 UTF-8 서브스트링 테스트를 무시하기 어떤 과도하게 긴 UTF-8 시퀀스를 남용할 수 있다. 모든 과도하게 긴 형식의 UTF-8 시퀀스는 다음의 바이트 패턴 중 한 가지로 시작한다.

정상적인 UTF-8 혹은 UCS-4 데이터상에서 코드 위치 U+FFFE와 U+FFFF 뿐만 아니라 코드 위치 U+D800부터 U+DFFF(UTF-16 대용)까지는 사용해서는 안 된다. UTF-8 디코더는 이러한 것들을 안전성을 이유로, 잘못된 형식으로 혹은 너무 긴 시퀀스로 취급해야 만 한다.

Markus Kuhn의 UTF-8 decoder stress test file <http://www.cl.cam.ac.uk/~mgk25/ucs/examples/UTF-8-test.txt>은 잘못된 형식을 갖는 과도하게 긴 UTF-8 시퀀스의 체계적인 모음을 포함하고 있으며 디코더의 강력함을 증명해준다.

제 8 절 유니코드를 지원하는 프로그래밍 언어는 무엇인가?

1993년경 이후에 개발된 최근의 프로그래밍 언어들은 이미 유니코드/ISO 10646-1 문자들을 위한 특별한 데이터 형을 가지고 있다. 이것은 Ada95의 경우 Wide_Character이며 자바의 경우 Char이다.

ISO C는 또한 멀티바이트 인코딩과 와이드 문자(wide character)를 취급하기 위한 메커니즘을 명시하고 있으며, *Amendment 1 to ISO C* <http://www.lysator.liu.se/c/na1.html>가 1994년 9월에 발행되었을 때 더욱 더 많은 것들이 추가되었다. 이러한 편리한 기능은 주로 여러가지 동아시아 문자들을 인코딩하기 위해서 설계되었고 UCS를 취급하기 위해서 필요한 것보다 훨씬 더 복잡해졌다. UTF-8은 ISO C 표준이 하나의 멀티바이트 문자열과 `wchar_t` 형을 호출하기 위한 하나의 인코딩 예인데, 이것은 현재의 환경에서는 보통 32비트의 부호있는 정수이며, 유니코드 문자를 저장하기 위해서 사용할 수 있다. C 컴파일러는 `yyyymmL` 형태를 갖는 상수 정수를 `__STDC_ISO_10646__` 변수에 매크로 정의하므로써, `wchar_t` 변수가 모든 로케일에 결친 유니코드 값을 저장한다는 것을 보증하는 신호를 애플리케이션에 보낼 수 있다(199712L을 예로 들어보면, 연도와 월 수는 ISO/IEC 10646과 그것이 언제 만들어졌고 언제 수정되었는지를 나타내는 버전을 참조한다는 것을 알 수 있다).

제 9 절 리눅스에서 유니코드를 어떻게 사용하는가?

UTF-8이 발표되기 전에는 다른 지역에 살고 있는 리눅스 유저들은 여러가지 ASCII 확장 문자를 사용하였다. 유럽에서는 ISO 8859-1과 ISO 8859-2를, 그리스에서는 ISO 8859-7을, 러시아에서는 KOI-8을, 일본에서는 EUC와 Shift-JIS를 가장 널리 사용하였다. 이로 인하여 파일을 교환하는 것이 어려웠고, 어플리케이션 소프트웨어를 제작하기 위해서는 이러한 인코딩들 사이의 차이점을 걱정해야만 했다.

유니코드는 결국에는 이러한 모든 인코딩들을 대신할 것이며, UTF-8 형식이 주를 이룰 것이다. UTF-8은 아래와 같은 경우에 사용될 것이다.

- 텍스트 파일(소스 코드, html 파일, 이메일 메시지 등)
- 파일명
- 표준 입출력, 파이프
- 환경 변수
- 선택한 버퍼 내용을 자르고 붙이기
- 텔넷, 모뎀, 터미널 에뮬레이터에 연결한 시리얼 포트
- 바이트 시퀀스가 ASCII 코드로 해석되는 곳이면 어떤 곳이든지 사용된다.

UTF-8 모드에서 xterm이나 리눅스 콘솔 드라이버와 같은 터미널 에뮬레이터는 대응 하는 UTF-8 시퀀스로 모든 키입력을 전달하며, 키입력을 포그라운드 프로세서의 표준 입력으로 보낸다. 비슷한 방식으로 프로세서의 표준 출력을 터미널 에뮬레이터로 보내 고 그곳에서 출력을 UTF-8 디코더로 처리하고 16비트 폰트를 사용하여 디스플레이한다 .

경고음을 갖는 완전한 유니코드 기능은 복잡한 다중-언어 워드-프로세싱 패키지에 사용할 것으로 예상할 수 있다. 리눅스는 틀림없이 ASCII 문자를 대신하기 위한 폭 넓 은 기반 위에서 사용될 것이며, 다른 8비트 문자셋은 훨씬 단순해질 것이다. 리눅스의 터미널 에뮬레이터와 명령 라인 도구들은 처음 단계에서 바로 UTF-8로 전환할 수 있을 것이다. 이것은 ISO 10646-1으로 구현한 레벨 1이 (어떠한 결합 문자도 사용하지 않고) 사용됨을 의미하며, 어떠한 프로세싱 지원도 필요로 하지 않는 라틴어, 그리스어, 키릴어 및 많은 과학 기호와 같은 서체(script)들만 지원됨을 의미한다. 이러한 레벨에서 UCS 지원은 ISO 8859 지원에 비교할 만 하며 유일한 중요한 차이점은 현재 유용한 수많은 다른 종료의 문자들이 있으며, 문자들은 멀티바이트 시퀀스로 나타낼 수 있다는 것이다.

리눅스에서 결국 결합 문자가 지원되겠지만 미리 만들어진(precomposed) 문자들이 유용한 결합 문자 시퀀스보다 더욱 좋은 선택이 되어야만 한다. 더 명백하게 말하자면, 리눅스상에서 유니코드로 만들어진 텍스트를 인코딩하는 더 좋은 방법은 *Unicode Technical Report #15* `<http://www.unicode.org/unicode/reports/tr15/>`에서 정의한 표준 형식 C *(Normalization Form C)*가 되어야만 한다.

영향력있는 POSIX 비호환 PC 운영체제 판매 회사 중 하나는(여기서 이름을 밝히지는 않겠다) 어떤 파일에서, 사용되는 인코딩과 바이트 순서를 식별하기 위해서 모든 유니코드 파일이 폭이 없는 노브레이크 스페이스 문자(ZERO WIDTH NOBREAK SPACE: U+FEFF)로 시작하게 하자고 제안했는데, 이러한 규칙에 따르면 어떤 파일에서 사용되는 인코딩과 바이트 순서(byte-order)를 식별하기 위해서 폭이 없는 노브레이크 스페이스 문자는 signature 혹은 "바이트-순서 마크(byte-order mark: BOM)"로써 참조할 수 있 다. 리눅스는 BOM이나 signature를 전혀 사용하지 않는다. BOM이나 signature의 사용은 수없이 많이 존재하고 있는 ASCII-파일 문법 규칙을 깨뜨릴 것이다. POSIX 시스템에서, 선택한 로케일은 어떤 프로세스의 입출력 파일에서 필요로 하는 인코딩을 미리 확인한다. 또한 signature "UTF-8N" 파일 없이 UTF-8 파일들을 호출 하기 위해서 그것 을 제안했었다. 그러나 이러한 비 표준적인 용어는 보통 POSIX 세계에서는 사용하지 않는다.

제 10 절 소프트웨어를 어떻게 수정해야만 하는가?

UTF-8 지원을 위한 두 가지 접근 방법이 있다. 이것을 소프트 및 하드 변환이라 부르기로 하겠다. 소프트 변환에서 데이터는 UTF-8 형식으로 어디 서나 보존되며 단지 매우 적은 수의 소프트웨어만 변화시킬 필요가 있다. 하드 변환에서 프로그램이 읽어들이는 UTF-8 데이터는 폭이 큰(wide) 문자 배열로 변환 될 것이며, 또한 애플리케이션 내부의 어느 곳에서도 처리될 것이다.

대부분의 애플리케이션들은 단지 소프트 변환만으로도 잘 동작한다. 소프트 변환은 유닉스 상에서 UTF-8을 받아들일 수 있게 하는 것이다. 예를 들면, `cat`과 `echo`와 같은 프로그램들은 전혀 수정할 필요가 없다. 그것들은 입출력이 ISO 8859-2 이든 UTF-8이든 상관없이 완벽하게 무시한다. 왜냐하면, 그것들은 입출력을 프로세스화하지 않고 단 지 바이트 스트림을 취급하기 때문이다. 그것들은 오직 UTF-8 상에서 어떠 한 변화도 일어나지 않는 `'\n'`과 같은 제어 코드와 ASCII 문 자만 인식한다. 그러므로 UTF-8 인코딩 및 디코딩은 터미널 에뮬레이터에서 동작하는 이러한 어플리케이션에 대해서 완벽하게 이루어진다.

바이트 수를 계산하여 한 문자열 안의 문자 수를 결정하는 모든 프로그 램을 위해서 약간의 수정이 필요할 것이다. UTF-8 모드에서, 프로그램들은 0x80부터 0xBF 범위 사이의 어떤 바이트 수도 계산해서는 안된다. 왜냐하면 이러한 바이트 수는 연속 바이트(continuation bytes)이며 소유하고 있는 문자는 아니기 때문이다. UTF-8과 함께 C언어의 `strlen(s)`함 수 또한 문자열 안의 문자수를 바르게 계산하지 않을 것이다. 그 대신, 만 약 UTF-8 로케일이 선택되었다면 문자수를 계산하기 위하여 `mbstowcs (NULL,s,0)` 함수를 사용할 수 있다.

예를 들어, `ls` 프로그램은 수정해야만 한다. 왜냐하면 그 것은 디렉토리를 유저에게 보여주기 위한 테이블의 레이아웃 체계를 갖추기 위해서 문자수를 파일명으로 계산하기 때문이다. 이와 유사하게 고정된 폭 을 갖는 폰트로 출력하고 그 출력을 포맷하는 프로그램들은 그것에 알맞게 UTF-8 텍스트 상의 문자 수를 계산하기 위한 방법을 알아야만 한다. 한 문 자를 지우는 것과 같은 에디터 함수는 한 문자에 속하는 모든 바이트를 지 우기 위해서 약간 수정해야만 한다. 예를 들면, `ncurses` 라이 브러리를 사용하는 프로그램 뿐만 아니라 `vi`나 `emacs`와 같은 에디터 역시 이와 비슷한 영향을 받는다.

리눅스 커널 또한 소프트 변환만으로도 잘 동작할 수 있으며, UTF-8을 완벽하게 지원하기 위한 아주 약간의 수정만 필요하다. 문자열(예를 들어 파일명, 환경 변수 등)을 취급하는 대부분의 커널 함수들은 영향을 받지 않 는다. 다음과 같은 경우에 수정이 필요할 수 있다.

- 콘솔 디스클레이와 키보드 드라이버(하나 더하자면 VT100 에뮬레이터) 는 UTF-8을 인코딩 및 디코딩해야 하며 적어도 유니코드 문자셋의 몇몇 서 브셋(subset)을 지원해야만 한다.
- VFAT과 WinNT와 같은 외부 파일 시스템 드라이버는 파일명을 나타내는 문자 인코딩을 변환시켜야만 한다. 이전에 유효했던 변환 옵션 목록에 UTF- 8을 더해야만 하며, `mount` 명령은 유저가 실행시킨 프로세서 가 UTF-8 파일명을 볼 수 있도록 커널 드라이버에게 알려야만 한다. VFAT 및 WinNT 파일 시스템은 이미 유니코드를 사용하고 있기 때문에, UTF-8은 변환 중에 손실이 일어나지 않는다는 것을 보증한다.
- 모든 POSIX 시스템의 tty 드라이버는 원시적인 한 행 편집 기능을 사용 할 수 있게 하는 일종의 "cooked" 모드를 지원한다. 문자 삭제 함수가 적절 히 작용되도록 하기 위해서 stty는 tty의 UTF-8 모드가 0x80에서부터 0xBF 까지의 범위 안에서 연속 배열을 문자로 계산하지 않도록 UTF-8 모드를 설 정해야만 한다. *Bruno Haible* <http://clisp.cons.org/~haible/>이 제공하는 stty와 커널의 tty 드라이버를 위한 몇가지 리눅스 패치 <ftp://ftp.ilog.fr/pub/Users/haible/utf8/>가 존재한다.

제 11 절 유니코드와 UTF-8을 위한 C의 지원

GNU glibc 2.2부터 살펴보면, 현재 사용하는 로케일과 상관없이, `wchar_t` 형은 공식적으로 오직 32비트 ISO 10646 값으로 사용되는 경향이 있다. ISO C99에서 필요로 하기 때문에 `__STDC_ISO_10646__` 매크로 정의에 의해서 애플리케이션에 이것을 시그널로 보낸다. ISO C 멀티-바이트 변환 함수들(`wprintf()`, `mbstowcs()`

등)은 glibc 2.2 혹은 그 이상에서 완벽하게 구현되며, UTF-8 과정을 포함하면서 wchar_t와 로케일 독립적인 멀티바이트 인코딩 사 이에서 변환하기 위해서 사용할 수 있다.

예를 들면 아래와 같이 쓸 수 있다.

```
wprintf(L"Schone Gruße!\n");
```

그런 뒤에 소프트웨어는 이러한 텍스트를 사용자가 환경 변수 LC_CTY PE(예를 들면, en_US.UTF-8 혹은 de_DE.ISO_885 9-1)으로 선택한 로케일에 명시된 인코딩상에 출력할 것이다. 컴파 일러는 C 소스 파일에서 사용하는 인코딩에 적합한 로케일에서 동작해야만 한다. 그리고 나면 위의 문자열은 유니코드 wchar_t 문자열에 의해서 오브 젝트 파일에 정확하게 저장될 것이다. 출력시에는 런-타임 라이브러리가 wc har_t 문자열을 프로그램이 실행되는 환경의 로케일에 꼭 맞는 인코딩으로 다시 변환시킬 것이다.

제 12 절 UTF-8 모드는 어떻게 활성화되어야 하는가?

여러분이 사용하는 애플리케이션이 8비트 문자셋(ISO 8859-*, KOI-8 등) 과 UTF-8 모두를 지원한다면 그것이 UTF-8 모드를 사용한다고 가정할 수 있 는 지 없는지를 몇 가지 방법으로 알아내야만 한다. 바라건대, 몇 년 안에 모든 사람들이 오직 UTF-8만을 사용하고 있을 것이며, 여러분들은 그것을 기본값으로 만들 수 있을 것이다. 그러나 고전적인 8비트 셋과 UTF-8 모두 가 틀림없이 지원되기 전까지는 아니다.

현재 사용되는 애플리케이션들은 그들 각각의 UTF-8 모드를 활성화시키 기 위해 전체적으로 다른 명령 라인 스위치들을 사용한다. 예를 들면:

- xterm의 명령 라인 옵션 "-u8"과 X 리소스의 "XTerm*utf8: 1"
- gnat/gcc의 명령 라인 옵션 "-gnatW8"
- stty의 명령 라인 옵션 "iutf8"
- mined의 명령 라인 옵션 "-U"
- xemacs에는 UTF-8과 내부적으로 사용하는 MULE 인코딩 사이에서 변환시 키기 위한 elisp 패키지가 있다.
- vim의 'fileencoding' 옵션
- less의 환경 변수 LESSCHARSET=utf-8

특별한 명령 라인 옵션이나 모든 애플리케이션을 위해 모든 설정 메카니즘 을 기억하는 것은 몹시 지루한 일이다. 그러므로 몇 가지 표준 사항을 급히 만들 필요가 있다.

제 13 절 UTF-8을 지원하는 X-term 버전은 어떻게 구하는가?

(*Thomas Dickey* <http://dickey.his.com/>가 관리하는 *XFree86* <http://www.xfree86.org/> 4.0 혹은 보다 높은 버전에서 사용하는 *xterm* <http://dickey.his.com/xterm/xterm.html> 버전은 이미 UTF-8 지원을 포함하고 있다. 만약 당신이 XFree86 4.0을 사용하지 않는다면, 대안으로써 *가장 최신의 xterm 개발 버전* <ftp://dickey.his.com/xterm/xterm.tar.gz>을 따로 다운로드할 수 있으며 여러분이 직접 "./configure --enable-wide-chars ; make" 명령을 사용하여 컴파일 할 수 있다.

xterm의 입출력 모드를 UTF-8로 변환시키기 위해서는 명령 라인 옵션 -u8을 사용하고 UTF-8 모드일 경우 *-ISO10646-1 폰트를 사용하라. ISO 10646-1 폰트들은 ISO 8859-1 폰트들의 하위 버전과 완벽하게 호환되기 때문에 ISO 8859-1 모드일 경우 *-ISO10646-1 폰트 또한 사용할 수 있다.

제 14 절 xterm은 얼마나 많은 유니코드를 지원하는가?

XFree86 4.0.1에 포함된 Xterm은 현재 고정 문자 폭을 가지며 왼쪽에서 오른쪽 방향으로 써야하는 ISO 10646-1의 레벨 1만을 지원한다. 다시 말하 면, 터미널의 의미 연결 체계(terminal semantic)가 이제 UTF-8을 디코딩할 수 있고 16비트 문자에 액세스 할 수 있다는 것을 제외하면, 터미널의 의 미 연결 체계는 ISO 8859-1의 경우와 기본적으로 같다.

가장 최신의 xterm 개발 버전 <ftp://dickey.his.com/xterm/xterm.tar.gz>은 이에 두 가지 새로운 함수들(*Robert Brady* <http://www.zepler.org/~rwb197/xterm/>가 제공)을 추가한다.

- 한중일어권의 표의문자를 위한 두배의 폭을 갖는 폰트에 대한 자동 변환
- 간단한 결합 문자 겹쳐쓰기(simple overstriking combining characters)

선택된 일반 문자가 X霧妹冪冪賣梅麥脈驀棉輞 픽셀 크기를 갖는다면 , xterm은 이제 부가적인 2X霧妹冪冪賣梅麥脈驀棉輞 픽셀 크기의 폰 트를 로드하려고 시도할 것이다(AVERAGE_WIDTH 속성의 두 배 값을 갖는다는 점을 제외하면 XLFD와 같다). *Unicode Technical Report #11* <http://www.unicode.org/unicode/reports/tr11/>에 따르면 *East Asian Wide(W)* 혹은 *East Asian FullWidth(F)*의 폭 특성(Width property)을 갖는 모든 유니코드 문자들을 나타내기 위해서 xte rm은 이러한 폰트를 사용할 것이다.

스페이스가 없거나(nonspacing) 둘러싸고 있는(enclosing) 결합 문자들(예를 들면, *유니코드 데이터베이스* <ftp://ftp.unicode.org/Public/UNIDATA/UnicodeData.txt>안에 *일반적인 카테고리 코드* <ftp://ftp.unicode.org/Public/UNIDATA/UnicodeData.html##GeneralCategory> Mn 이나 Me를 가지고 있는 문자들) 또한 현재 사용 가능하며 이것은 기본 -문자 glyph(base-character glyph)을 두 개의 결합 문자 glyph까지 단지 겹쳐쓰므로써(논리 연산자 OR-ing) 구현할 수 있다. 이것은 기본 라인 아래 에 액센트를 받아들일 수 있게 하고, 작은 문자 위에 액센트를 받아들일 수 있게 한다. 예를 들면 겹쳐쓰기와 함께 사용하기 위해서 특별히 설계된 태 국 폰트에 대해서도 또한 잘 동작한다. 그러나 특별히 "고정되고" 집단화된 폰트와 같은 어떤 폰트 안에서 키가 큰 문자 위에 액센트를 결합하는 것은 완전히 만족스럽지 않을 지도 모른다. 그러므로 미리 만들어진(precompose d) 문자들을 사용 가능한 경우에는 그것이 더 나은 결과를 낳을 것이다.

텍스트 모드 애플리케이션 프로그래머들이 주의해야 할 점:

xterm의 출력은 한중일어권의 한자와 결합 문자를 위한 지원으로써 보다 는 비례 문자처럼 반응할 것이다. 왜냐하면 라틴/그리스/키릴 등의 문자는 하나의 세로열 위치를 필요로 하는 반면, 한중일어권의 한자는 2개의 문자 를 필요로 하며 결합 문자는 하나도 필요로 하지 않기 때문이다.

오픈 소스 그룹의 *단일 유닉스 사양* <http://www.UNIX-systems.org/online.html>은 하나의 문자가 얼마나 많은 세로열을 차지하고 있 는지를 애플리케이션이 테스트 할 수 있도록 하는 두 개의 C 함수 *wcwidth()* <http://www.cl.cam.ac.uk/~mgk25/ucs/wcwidth.html>와 *wcswidth()* <http://www.cl.cam.ac.uk/~mgk25/ucs/wcswidth.html> 를 명시하고 있다.

```
#include <wchar.h>
int wcwidth(wchar_t wc);
int wcswidth(const wchar_t *pwcs, size_t n);
```

Markus Kuhn 이 비 상용(free)으로 배포하는 wcwidth() 함수 <http://www.cl.cam.ac.uk/~mgk25/ucs/wcwidth.c>는 C 라이브러리가 아직 적합한 함수를 제공하지 않는 플랫폼의 애플리케이션에 의해 사용될 수 있 다.

xterm은 앞으로 다가올 미래에 더욱 복잡한 완전한 유니코드 렌더링 엔 진에서 당신이 기대할 지도 모를 다음의 기능들을 아마도 지원하지 않을 것 이다.

- 유태어와 아랍 문자들을 위한 양방향 출력

- 아랍의 프리젠테이션 양식을 대신할 양식
- 인도의 연결선(ligature)을 대신할 문자
- 한글 자모
- 결합 문자를 저장할 임의의 스택

그러므로 유태어와 아랍어 사용자들은 유태어와 아랍 문자열을 터미널로 보내기 전에 문자열의 방향을 반전시키고 왼쪽을 채우는 애플리케이션 프 로그램을 사용해야만 한다. 다시 말해, 양방향 프로세싱은 xterm에 의해서 가 아니라 애플리케이션에 의해서 이루어져야만 한다. 유태어 및 아랍어와 관련된 상황은 미리 만들어진 glyph과 프리젠테이션 양식의 유효성에 있어 서 적어도 ISO 8859 이후 개선되고 있다. 지금으로써는 양방향 지원이 정말 로 xterm에 내장될 것인지 이것이 얼마나 정확하게 작동할 것인지 명확하지 않다. *ISO 6429 = ECMA-48* <http://www.ecma.ch/ecma1/STAND/ECMA-048.HTM>과 *유니코드 bidi 알고리즘* <http://www.unicode.org/unicode/reports/tr9/> 모두 대안적인 시작 포인트(starting point) 를 제공한다. *E CMA Technical Report TR/53* <http://www.ecma.ch/ecma1/TECHREP/E-TR-053.HTM>을 보도록 하라.

만약 당신이 애플리케이션에서 양방향 텍스트 출력을 지원할 계획이라면 , 유니코드 Bidi 알고리즘을 비 상용(free)으로 구현한 Dov Grobgeld의 *FriBidi* <http://imagic.weizmann.ac.il/~dov/freesw/FriBidi/> 혹은 Mark Leisher의 *PretBidi 알고리즘* <http://crl.nmsu.edu/~mleisher/ucdata.html>을 보라.

최근에 비록 *Robert B rady* <http://www.zepler.org/~rwb197/xterm/>가 bidi 지원을 위한 몇몇 초기 실험적인 패치를 발표하긴 했지만 xterm은 현재 아랍어, 한글 자모 및 인도 텍스트 포맷팅 알고리즘을 지원하 지 않는다. VT100 애뮬레이터에서 이러한 것들을 지원하는 것이 알맞은지 > 혹은 바람직한지 여전히 불명확하다. 애플리케이션들은 아랍어와 한글을 포 맷하는 알고리즘을 쉽게 애플리케이션 스스로 적용할 수 있다. 왜냐하면 xt erm이 애플리케이션으로 하여금 모든 필요한 프리젠테이션 양식을 출력하는 것을 허용하기 때문이다. 인도 서체에 관해서 X 폰트 메카니즘은 필수적인 연결선(ligature)을 대체할 또 다른 형태의 연결선을 위한 인코딩을 지금> 으로써는 전혀 지원하지 않고 있다. 그래서 제공할만 한 xterm이 전혀 없다 . 인도어 출력이 필요한 애플리케이션은 xterm과 같은 VT100 에뮬레이터 대 신에 *Pango* <http://www.pango.org/>와 같은 적절한 유니코드 X11 렌더링 라이브러리를 사용하는 것이 더욱 좋다.

제 15 절 ISO-10646의 X11용 폰트는 어디서 구할 수 있는가?

지난 몇년 동안 꽤 많은 수의 유니코드 폰트들이 X11용으로 사용 가능하게 되었으며, 사용 가능한 폰트의 수는 빠른 속도로 증가하고 있다.

- Markus Kuhn은 많은 다른 지원자들과 함께 X11과 관련한 과거의 -misc-FIXED-*-Iso8859-1 폰트들을 모든 유럽 문자들(라틴어, 그리 스어, 키릴어, 국제적인 표음 문자, 수학 및 기술 부호와 아르메니아어, 그 루지아어, 카타카나, 태국어, 그외의 다른 문자와 같은 몇몇 폰트들에서 사 용하는)을 지원하는 목록 일람으로 확장시켰다. 더 많은 정보를 얻으려면 *X11을 위한 유니 코드 폰트와 툴* <http://www.cl.cam.ac.uk/~mgk25/ucs-fonts.html> 페이지를 살펴보라. 이러한 폰트들은 현재 *XFree86* <http://www.xfree86.org/> 4.0.1 혹은 그 이상의 버전과 함께 배 포된다.
- Markus는 또한 *X11R6.4 배포판 내에 포함된 모든 Adobe와 Bmp;H BDF의 폰트들의 ISO 10646-1 버전* <http://www.cl.cam.ac.uk/~mgk25/download/ucs-fonts-75dpi100dpi.tar.gz>을 준비했다. 이러한 폰트들은 이미 완전한 포스트스크립트 폰트의 목록 일람(대략 30개의 문자들이 추가 되었으며 대부분 CP1252 MS-윈도우즈에서 또한 사용된다. 예를 들면, 깔끔 한 인용부호와 대시 기호(-) 등이 있다)을 포함한다. 그러나 이러한 것들은 ISO 8859-1 인코딩 아래에서는 사용할 수 없었다. 지금은 ISO 10646-1 버 전내에서 모두 액세스 할 수 있다.
- XFree86 4.0은 ISO 10646-1 인코딩에서 모든 애플/마이크로소프트 폰트 를 X 어플리케이션에서 사용할 수 있게 하는 *통합된 트루타입 폰트 엔진* <http://www.dcs.ed.ac.uk/home/jec/programs/xfsft/>과 함께 제 공된다.

- 앞으로의 XFree86 릴리즈는 배포판으로부터 과거의 BDF 폰트들을 대부 분 제거하고 그것들을 ISO 10646-1로 인코딩한 버전으로 대체될 것 같다. X 서버는 과거의 8비트 소프트웨어가 어떤 폰트를 요청할 때 재빨리 ISO 106 46-1 폰트 파일로부터 ISO 8859-*와 같은 다른 폰트 인코딩을 생성하는 자 동 인코딩 변환기로 확장될 것이다. 현대의 소프트웨어는 ISO 10646-1 폰트 인코딩을 오히려 직접 사용해야만 한다.

- *ClearlyU (cu12)* <ftp://crl.nmsu.edu/CLR/multiling/unicode/fonts/>는 *Mark Leisher* <mailto:mleisher@crl.nmsu.edu>가 만든 3700개 이상의 문자들을 가진 X11을 위한 12개의 점과 100dpi(단위 인치당 점의 개수)에 비례하는 크기의 매우 유용한 ISO 10646-1 BDF 폰트이 다.(*이미지 예* <http://crl.nmsu.edu/~mleisher/cu-examples.html>)

- **NEW**: Dmitry Yu. Bolkhovityanov는 텍스 트 모드의 IBM PC 에뮬레이터에서 사용하기 위해서 BDF 파일 안에 *유니코 드 VGA* 폰트 <http://www.inp.nsk.su/~bolkhov/files/fonts/univga/index.html>를 만들었다.

- Roman Czyborra의 *GNU 유니코 드* 폰트 <http://czyborra.com/unifont/> 프로젝트는 비 상용(free)의 완벽한 8旕浪6/16旕浪6 픽 셀 유니코드 폰트를 모으는 연구를 하고 있다.

- *etl-unicode* <ftp://ftp.x.org/contrib/fonts/etl-unicode.tar.gz>는 *Primoz Peterlin* <mailto:primoz.peterlin@biofiz.mf.uni-lj.si>이 준비한 ISO 10646-1 BDF 폰트이다.

- George Williams는 *Type1 유니코드 폰트 패밀리* <http://bibliofile.mc.duke.edu/gww/fonts/Unicode.html>를 만들었는데, 이것은 또 한 BDF에서도 유용하다. 그는 또한 *PfaEdit* <http://bibliofile.mc.duke.edu/gww/FreeWare/PfaEdit/> 포스트스크립트와 비트맵 폰트 에디 터를 개발했다.

유니코드 X11용 폰트의 명칭은 -ISO10646-1이라는 단어로 끝난다. 이것은 현재 모든 유니코드와 ISO 10646-1 16비트 폰트들을 위한 *X 논리 폰트 기술자(X Logical Font Descriptor: XLF D)* <ftp://sunsite.doc.ic.ac.uk/packages/X11/pub/R6.4/xc/doc/hardcopy/XLFD/xlfd.PS.gz> 영역 CHARSET_REGISTRY와 CHARSET_ENCODING 을 위해 공식적으로 *등록된* <ftp://sunsite.doc.ic.ac.uk/packages/X11/pub/R6.4/xc/registry> 값이다. *-ISO10646-1 폰 트들은 전체 유니코드 문자셋의 명시되지 않은 몇몇 서브셋을 포함하고 있 다. 그리고 사용자들은 그들이 선택한 폰트가 무엇이든지 간에 필요한 문자 들의 서브셋을 포함하는지 확인해야만 한다.

*-ISO10646-1 폰트들은 일반적으로 폰트에서 유효하지 않 은 어떤 문자를 나타내기 위해서 유니코드가 아닌(non-Unicode) 특별한 gly ph을 가리키는 DEFAULT_CHAR 값을 명시하고 있다(일반적으로 0x00에 위치하고 H 크기를 갖는 대시(-)로 연결된 box이다). 이것은 사용자 들이 적어도 지원하지 않는 문자가 있다는 것을 명백하게 알아차릴 수 있도 록 한다. xterm을 위한 6x13과 같은 크기의 작은 고정-폭 폰트들(the small er fixed-width fonts)은 모든 유니코드를 지원할 수 없을 것이다. 왜냐하 면 간지(Kanji)와 같은 많은 서체들(scripts)은 유럽지역 유저들이 폭넓게 사용하는 픽셀 사이즈 보다 훨씬 더 큰 픽셀 사이즈로 나타내기 때문이다. 유럽인이 사용하기 위한 전형적인 유니코드 폰트들은 *CEN MES-3 레퍼토리* <http://www.egt.ie/standards/iso10646/pdf/cwa13873.pdf>와 같은 오직 1000내지 3000가지 문자들의 서브셋을 포함할 것이다.

여러분들은 *-ISO10646-1 폰트에서 *ASCII 문자의 인용 부호 형태* <http://www.cl.cam.ac.uk/~mgk25/ucs/quotes.html>가 표준 라인 안 에 인용 부호를 배치하고 다른 플랫폼에서도 실행하기 위해서 약간 바뀌었 다는 것을 알아 차릴 수 있을 것이다.

제 16 절 UTF-8 터미널 에뮬레이터와 관련된 이슈는 무엇인가?

VT100 터미널 에뮬레이터들은 다른 문자셋들 사이를 전환하기 위해서 IS O 2022(=*ECMA-35* <http://www.ecma.ch/ecma1/STAND/ECMA-035.HTM>) ESC 시퀀스를 받아들인다.

UTF-8은 ISO 2022의 관점에서 보면 "다른 코딩 시스템(other coding sys tem)"이다(ECMA 35의 섹션 15.4를 보라). UTF-8은 ISO 2022 SS2/SS3/G0/G1/ G2/G3이 속하는 세계의 외부에 있다. 그러므로 만약 ISO 2022에서 UTF-8로 전환하면, 모든 SS2/SS3/G0/G1/G2/G3 문은 UTF-8을 벗어나 다시 ISO 2022로 돌아가기 전까지는 의미를 잃게 된다. UTF-8은 국적이 없는 인코딩이므로, 스스로 종결시키는(self-terminating) 짧은 길이의 바이트를 갖는 시퀀스(short byte sequence)는 전환하는 문장과는 독립적으로 어떤 문자가 의미가 있는지를 완벽하게 판정한다. ISO 10646-1 안의 G0와 G1은 ISO 8859-1의 그것들과 같다. 그리고 G2/G3는 ISO 10646 내에 존재하지 않는다. 왜냐하면 모든 문자는 고정된 위치를 가지며 어떤한 변경도 일어나지 않기 때문이다 . 우연히 바이너리 파일을 터미널에 덤프한 후에 터미널이 이상한 그래픽- 문자 모드로 전환된 채 남아있는 것은 UTF-8에서는 가능하지 않다. 이것은 UTF-8 모드에 있는 어떤 터미널을 ISO 2022 모드 일때보다 훨씬 더 강력하 게 동작하도록 한다. 그러므로 터미널이 우연히 ISO 2022 모드로 돌아갈 수 없도록 그것을 UTF-8 모드로 고정시켜 놓는 것이 효과적이다.

ISO 2022 표준은 ISO 2022 모드에서 벗어나기 위한 이스케이프 문자 %의 시퀀스 범위를 명시하고 있다(다른 코딩 시스템 지정, DOCS). 그리고 그러 한 수많은 시퀀스들은 *UTF-8* <ftp://ftp.informatik.uni-erlangen.de/pub/doc/ISO/charsets/ISO-10646-UTF-8.html>을 위해서 *ISO 2375 문자 코드 셋 국제 등록부(I nternational Register of Coded Character Sets)* <http://www.itscj.ipsj.or.jp/ISO-IR/>의 섹션 2.8에 등록되 었다.

- ESC %G는 ISO 2022로 다시 돌아가도록 하는 방식으로 ISO 2022에 명시되지 않은 구현 레벨로 UTF-8을 활성화시킨다.
- ESC %@는 ESC %G를 거쳐서 UTF-8로 들어간 경우에 UTF-8에서 ISO 2022로 되돌아가게 한다.
- ESC %/G는 UTF-8의 레벨 1으로 반환값 없이 전환시킨다.
- ESC %/H는 UTF-8의 레벨 2으로 반환값 없이 전환시킨다.
- ESC %/I는 UTF-8의 레벨 3으로 반환값 없이 전환시킨다.

터미널 에뮬레이터가 UTF-8 모드에 있는 동안에 G2/G3로 전환시키는 이 스케이프 시퀀스와 같은 모든 ISO 2022 이스케이프 시퀀스는 무시된다. UTF -8 모드에서 동작하는 터미널 에뮬레이터 상의 유일한 ISO 2022 시퀀스는, UTF-8에서 ISO 2022 체계로 다시 전환시키는 ESC %@이다.

비록 UTF-8 모드가 0x80에서 0x9F까지의 범위를 갖는 바이트 공간을 사 용하지만, 여전히 CSI와 같은 C1 제어 문자들을 사용하는 것을 허용한다. U TF-8 모드에 있는 터미널 에뮬레이터는 어떤 제어 문자를 해석하기 **전에** UTF-8 디코더를 입력되는 바이트 스트림에 적용해야만 한다 는 것을 이해하는 것이 중요하다. C1 문자들은 U+007F를 넘는 다른 문자들 처럼 UTF-8 모드로 디코딩된다.

제 17 절 UTF-8이 가능한 지금 사용 중인 애플리케이션은 어떤 것이 있는가?

- XFree86 4.0 혹은 그 이상의 버전과 함께 사용되는 *xterm* <http://dickey.his.com/xterm/xterm.html>(" ./configure --enable- wide-chars; make" 명령으로 컴파일하고 xterm을 실행할 때 명령라 인 옵션 -u8을 사용하라).
- **NEW:** *Yudit 2.0* <http://www.yudit.org/>은 Gaspar Sinai가 개발한 비 상용 X11 유니코드 에디터이다.
- *Thomas Wolff* <http://www.inf.fu-berlin.de/~wolff/>가 개발한 *Mined 9 8* <http://www.inf.fu-berlin.de/~wolff/mined.html>은 UTF-8 사용이 가능한 텍스트 에디터이다.
- *Cooledit* <http://cooledit.org/>는 3.15.0버전 이후로 UT F-8과 UCS를 지원한다.

- **NEW:** *QEmacs* <http://www-stud.enst.fr/~bellard/qemacs/>는 UTF-8 터미널에서 사용하기 위한 작은 에 디터이다.
- 346버전 이후의 *less* <http://www.flash.net/~marknu/less/>는 UTF-8을 지원한다.
- *C-Kermit 7.0* <http://www.columbia.edu/kermit/ckermit.html>은 전송(transfer), 터미널 및 파일 문자셋에 관해 UTF-8을 지원한다.
- *Perl* <http://www.perl.org/>은 "use utf8;" 옵션으로 요 청했을 경우에 버전 5.6부터 몇몇 *핵심적인 UTF-8을 지원(core UTF-8 support)* <http://rf.net/~james/perli18n.html>한다. 이것은 문자열이 UTF-8로 저장되고(그리고 UTF-8로 태그화 되며) length() 함수가 바이 트 수 대신에 문자들을 반환함을 의미한다. UTF-8 지원을 강화하기 위한 많 은 연구들이 지금 순간에도 여전히 진행 중이다(*perl-unicode@perl.org* <mailto:perl-unicode-help@perl.org> 메일링 리스트를 보라). 사 용 가능한 기능과 제한되는 기능을 보려면 `perldoc perlunicode` 와 `perldoc utf8`을 읽어보라.
- *Python 1.6* <http://www.python.org/1.6/>은 현재 *유니코드 지원* <http://starship.python.net/crew/lemburg/unicode-proposal.txt>을 완성하고 있다.
- *Tcl/Tk* <http://dev.scriptics.com/>는 버전 8.1부터 *유니코드를 기본 문자셋* <http://dev.scriptics.com/doc/howto/i18n.html>으로 사용하여 가동된다. 그러나 불행하게도 Tk의 폰트 처리 코드 에는 여전히

제 18 절 UTF-8을 지원하기 위해서 사용가능한 패치는 무엇인가?

- *Bruno Haible* <http://clisp.cons.org/~haible/>은 stty와 리눅스 커널 tty 및 groff 등을 위한 *여러가지 패치* <ftp://ftp.ilog.fr/pub/Users/haible/utf8/>를 준비했다.
- Miyashita Hisashi는 Emacs 20.6과 그 이상의 버전을 위한 문자셋 변역 패키지인 *MULE-UCS* <ftp://ftp.m17n.org/pub/mule/Mule-UCS/> 를 작성했다. 이것은 Mule 인코딩(Emacs에서 내부적으로 사용하는)과 ISO 1 0646 사이를 전환시킬 수 있다.
- Otfried Cheong은 *GNU Emacs를 위한 유니코드 인코딩* <http://www.cs.uu.nl/~otfried/Mule/> 페이지에, 또다른 Emacs 문자셋으로 `utf-8`을 덧붙임으로써 모든 BMP를 지원하는 MULE-UCS에 대한 일 종의 확장기능을 제공한다. 그가 만든 페이지는 또한 MULE-UCS를 위한 짧은 설치 안내서를 포함한다.
- Tomohiko Morioka는 *UTF-8 xemacs를 위한 패치* <http://turnbull.sk.tsukuba.ac.jp/Tools/XEmacs/>를 내놓았다.
- *Edmund Grimley Evans* <http://www.rano.org/>는 이메일 프로그램인 Mutt와 curses library의 일종인 Slang을 위한 UTF-8 *패치* <http://www.rano.org/mutt.html>를 준비했다.

제 19 절 유니코드를 다루는 사용가능한 비 상용(free) 라이브러리는 있는가?

- **NEW:** Ulrich Drepper가 개발한 *GNU C 라이브러리 glibc 2.2.1* <http://sourceware.cygnus.com/glibc/> 은 UTF-8을 위한 완벽한 멀티-바이트 로케일을 지원하기 위해서 유니코드 정렬 순서 알고리즘(sorting order algorithm)을 포함하며, 이것은 다른 많 은 인코딩으로 다시 코딩할 수 있다. 대부분의 리눅스 배포판들은 가까운 미래에 glibc 2.2.1을 업그레이드할 것으로 예상되지만, 경험 많은 리눅스 사용자들은 *glibc 2.2.1* 소스 <ftp://sourceware.cygnus.com/pub/glibc/releases>를 직접 설치하려고 시도할 수 있다(이런 방법은 다루 기가 어렵고 위험하다). Bruno Haible의

제 20 절 UTF-8을 지원하기 위해 어떤 패키지가 현재 개발 중인가?

* **NEW:** 고전적인 vi 에디터의 인기있는 클 론인 Vim의 최신 *알파 테스트 버전 6.0s* <ftp://ftp.vim.org/pub/vim/unreleased/>는 현재 와이드 문자와 2개까지의 결합 문자를 지원함 과 동시에 UTF-8을 지원한다. 자세한 것은 Bram Moolenaar의 *announcement* <http://mail.nl.linux.org/linux-utf8/2000-07/msg00036.html> 를 읽어보라.

제 21 절 솔라리스 상에서 UTF-8을 위한 지원은 어떻게 동작하는가?

Solaris 2.8버전부터 그 이후로 UTF-8은 적어도 부분적으로 지원된다. UTF-8을 사 용하기 위해서는 UTF-8 로케일 중 하나를 설정하라. 예를 들면 다음과 같이 C 쉘에서 입력한다.

```
setenv LANG en_US.UTF-8
```

현재 UTF-8 텍스트를 입력 및 출력하기 위해서 dtterm 터미널 에뮬레 이터를 사용할 수 있으며, mp print filter는 포스트스크립트 프린터에 서 UTF-8 파일을 출력할 것이다. en_US.UTF-8 로케일은 지금으로써는 Mo tif와 CDE 데스크탑 애플리케이션 및 라이브러리에 의해서 지원되지만, OpenWindows, XView 및 OPENLOOK DeskSet 애플리케이션과 라이브러리에 의해서는 지원되지 않는다.
더 많은 정보를 원한다면, en_US.UTF-8 로케일 지원 웹 페이지의 Sun's *Overview* <http://docs.sun.com:80/ab2/coll.45.13/I18NDG/@Ab2PageView/10821?Ab2Lang=C&Ab2Enc=iso-8859-1>를 읽어보라.

제 22 절 포스트 스크립트 glyph 명칭(Postscript glyph names)>은 어떻게 UCS 코드와 관련되어 있는가?

Adobe사의 *Unicode and Glyph Names* <http://partners.adobe.com/asn/developer/typeforum/unicodegn.html> 가이드를 읽어보라.

제 23 절 잘 정의된 UCS subset은 있는가?

40000개의 문자를 가지는 유니코드를 완벽하게 구현하는 것은 거대한 프로젝트이다 . 그러나 전과 같이 단지 수백 또는 수천 문자를 구현하는 것과 유니코드화를 거친 단 순한 하나의 인코딩속에서 모든 필요한 문자에 접근하는 단순함을 즐기는 것도 종종 중요하다(특히 유럽 시장을 위해서). 수많은 다른 UCS 서브셋들은 이미 확립되었다.

* *Windows Glyph List 4.0 (WGL4)* <http://partners.adobe.com/asn/developer/opentype/wgl4.htm>는 8비트 MS-DOS, Windows, Mac 및 마이크로소프트가 예전에 사용한 적이 있는 ISO 코드 페이지를 모두 지원하는 650 문자로 구성된 셋이다. 모든 Windows 폰트는 현재 적어도 WGL4 전 목록을 포함한다. WGL4는 CEN MES-1(<http://www.cl.cam.ac.uk/~mgk25/ucs/wgl4.txt>= WGL4 테스트 파일">)을 모두 포함하는 셋이다.
* 세가지 *유럽용 UCS 서브셋 MES-1, MES-2* 및 *MES-3* <http://www.egt.ie/standards/iso10646/pdf/cwa13873.pdf>은 유럽 표준 위원회 CEN/TC304에 의해서 CWA 13873 안에 정의 되었다.

  - MES-1은 단지 335가지 문자를 갖는 매우 작은 라틴어 하위 문자셋이다. 이것은 정확하게 ISO 6937에서 볼 수 있는 모든 문자와 이에 더한 EURO SIGN을 포함한다. 이것은 ISO 8859의 1,2,3,4,9,10,15 부분의 모든 문자를 MES-1이 포함한다는 것을 의미한다. : $UCS, MES-1MES-1Windows125214 : U+0192, U+02C6, U+02DC, U+2013, U+2014, U+201A, U+201E, U+2020, U+2021, U+2022, U+2026, U+2030, U+2039, U+203A.]MES-21052////.(EU)8..MES-2MES-1., MES-2.[: -, MES-28WGL4 : U+2113, U+212E, U+2215, U+25A1, U+25AA, U+25AB, U+25CF, U+25E6.MES-2, 8WGL4, WGL4.$
- MES-3는 2819문자를 갖는 매우 포괄적인 UCS 서브셋이다. 이것은 단순히 유럽 사용자들에게는 잠재력이 있는 매우 유용한 UCS 모음(collection)을 모두 포함한다. 이것은 더욱더 열정적인 개발자들을 위한 것이다. MES-3은 MES-2와 WGL4를 포함하는 문자셋이다.

* JIS X 0221-1995는 일본어 사용자들을 위한 7개의 겹치지 않는 UCS 서브셋을 명시하고 있다.
  - 기본적인 일본어(6884개 문자): JIS X 0208-1997, JIS X 0201-1997
  - 일본어 비-표의 문자(Non-ideographic) 보완(1913개 문자): JIS X 0212-1990 비-간지 문자(non-kanji) 및 여러가지 다른 비-간지 문자
  - 일본어 표의 문자 보완 1(918개 문자): 몇몇 JIS X 0212-1990 간지 문자
  - 일본어 표의 문자 보완 2(4883개 문자): 나머지 JIS X 0212-1990 간지 문자
  - 일본어 표의 문자 보완 3(8745개 문자): 나머지 중국어 문자
  - 완전한 폭을 갖는 Alphanumeric(94개 문자): 호환성을 위해서
  - 절반의 폭을 갖는 카타카나 문자(63개 문자): 호환성을 위해서
* ISO 10646 표준은 그것의 전체 목록을, 서브셋을 정의하고 기록하기 위해서 사용하는 수많은 *묶음들(collections)* <http://www.egt.ie/standards/iso10646/ucs-collections.html>로 나눈다. 유니코드도 비슷하지만, 똑같지는 않은 유니코드 표준의 각 섹션에 대응하는 문자들의 *블록* <ftp://ftp.unicode.org/Public/UNIDATA/Blocks.txt> (blocks of characters)을 정의하고 있다.
* *RFC 1815* <ftp://sunsite.doc.ic.ac.uk/packages/rfc/rfc1815.txt>는 ISO 10646을 명백히 좋아하지 않으며 JIS X 0221-1995를 잘 모르는 누군가에 의해서 1995년에 쓰여진 일종의 메모이다. 그것은 14개의 UCS 묶음으로 구성된 "ISO-10646-J-1"이라고 불리는 어떤 UCS 서브셋에 관해 논의하고 있으며, 14개의 UCS 묶음의 몇몇은 JIS X 0208과 엇갈린다. 이것은 과거 1995년의 일본판 Windows NT 버전에 포함된 어떤 특별한 폰트가 우연히 만들어졌다는 것이다. RFC 1815는 오늘날 완전히 시대에 뒤떨어졌으며 적절하지 않으며, 무시하는 것이 최선이다.
* Markus Kuhn은 *ucs-fonts.tar.gz* <http://www.cl.cam.ac.uk/~mgk25/download/ucs-fonts.tar.gz>의 README 파일에서 세가지 UCS 서브셋 TARGET1, TARGET2 및 TARGET3을 정의하고 있는데 이들은 대응하는 MES 서브셋을 알맞게 확장한 것이며, xterm의 폰트 패키지를 완성하기 위한 근간이 되었다.

Markus Kuhn의 *유니셋(uniset)* <http://www.cl.cam.ac.uk/~mgk25/download/uniset.tar.gz> Perl 스크립트는 구현한 프로그램이 제대로 동작하는 지를 체크하기를 원하거나 새로운 프로그램을 만들고 싶어하는 사람들을 위하여 UCS 서브셋 위에 편리한 산술 계산 셋(set)을 사용하는 것을 허용하고 있다.

제 24 절 X11 R6.4 스펙과 유니코드에 어떤 문제점이 있는가?

X11 R6.4 릴리즈 <ftp://ftp.x.org/pub/R6.4/>(1998)는 X 컨소시움이 개발한 X11 윈도우 시스템 표준에 맞는 샘플 프로그램의 최신 버전이다. 대부분의 *현재의 X11 표준* <ftp://ftp.x.org/pub/R6.4/xc/doc/hardcopy/>과 샘플 프로그램은 유닉스 환경에서의 유니코드에 대한 광범위한 관심을 일찍부터 불러일으키고 있다.

* **UTF-8 잘라내기와 붙이기:** *ICCCM* <ftp://ftp.x.org/pub/R6.4/xc/doc/hardcopy/ICCCM/icccm.PS.gz> 표준은 선택한 UCS 문자열을 변환하는 방법을 명시하지 않고 있다. 몇몇 벤더들은 현존하는 *COMPOUND_TEXT* <ftp://ftp.x.org/pub/R6.4/xc/doc/hardcopy/CTEXT/ctext.PS.gz> 메카니즘 (CTEXT)에 UTF-8을 또 하나의 인코딩으로써 추가하였다. 이것은 적어도 다음과 같은 이유에서 좋은 해결책이 아니다.
  - CTEXT는 다소 복잡한 ISO 2022 메카니즘이다. 그러나 유니코드는 CTEXT에 또 다른 추가 항목을 제공하는 것이 아니라 끔찍한 메카니즘 전체를 훨씬 간단하고 더욱 편리하며 동등한 능력을 갖는 무언가로 대체할 기회를 제공한다.
  - 존재하는 많은 애플리케이션들은 CTEXT를 통해 선택한 것과 통신할 수 있지만, 새롭게 추가한 UTF-8 옵션을 지원하지 않는다. CTEXT 사용자는 과거의 ISO 2022 인코딩을 사용할 것인지 새로운 UTF-8 인코딩을 사용할 것인지를 결정해야만 한다. 그러나 두 가지를 동시에 사용할 수는 없다. 다시 말해서, CTEXT에 UTF-8을 신중히 추가한다고 할지라도 존재하는 CTEXT 애플리케이션의 하위 버전과의 호환성은 깨진다.
  - 현재의 CTEXT 스펙은 섹션 6에서 명확하게 다음과 같이 UTF-8의 추가를 금지하고 있다. "'다른 코딩 시스템'에 등록된 ISO는 컴파운드 텍스트에서 사용되지 않는다; 확장된 세그먼트들은 ISO 2022가 아닌 인코딩(non-2022 encodings)을 위한 유일한 메카니즘이다."

  Juliusz Chroboczek <http://www.dcs.ed.ac.uk/home/jec/>은 속성 형(property type)과 선택 목표(selection target)로 사용할 수 있는 새로운 UTF8_STRING 요소를 이용하여 UTF-8의 선택 여부를 다루기 위한 ICCM의 확장에 대해서, "*유니코드 텍스트의 내부-클라이언트 교환* <http://www.dcs.ed.ac.uk/home/jec/programs/xfsft/UTF8-selections.text>에 관한 제안 초안(Inter-Client Exchange of Unicode Text draft proposal)"을 작성하였다.
* **쓸모 없는 폰트 데이터 구조들:** 폰트의 미터 단위 길이에 관한 정보를 나타내기 위해 사용하는 Xlib API와 X11 프로토콜 데이터 구조는 드물게 사용하는 폰트들(sparsely populated fonts)을 처리할 때는 대단히 쓸모가 없다. X 클라이언트에서 어떤 폰트에 액세스하는 가장 일반적인 방법은 XLoadQueryFont() 함수를 호출하는 것이며, 이것은 XFontStruct를 위해 메모리를 할당하고 서버로부터 그것의 내용을 불러온다. XFontStruct는 각각 12바이트인 XCharStruct 엔트리(entry)의 배열을 포함한다. 이러한 배열의 크기는 마지막 문자의 코드 위치 - 첫 번째 문자의 코드 위치 + 1 이다. 그러므로 U+0020과 U+FFFD를 모두 포함하는 임의의 "*-iso10646-1" 폰트는 65502 요소를 갖는 XCharStruct 배열이 할당되도록 할 것이다(심지어는 CharCell 폰트들에 대해서도 같은 결과가 발생할 것이다). 이것은 그 폰트가 오직 1000개의 문자들을 포함하고 있다 할 지라도 786 킬로바이트의 클라이언트-사이드 메모리(client-side memory)와 데이터 전송을 필요로 함을 의미한다. 지금까지 거론해왔던 일과 관련된 몇몇 주변의 일들:
  - XFree86 4.0에 함께 제공되는 비-아시아권용 **-misc-fixed-*-iso10646-1** 폰트들은 U+31FF 이상의 영역에서는 어떠한 문자도 포함하지 않는다. 이것은 필요 한 메모리 량을 153 킬로바이트까지로 제한한다. 이것은 여전히 좋지 않은 일이지만, 그렇게 나쁜 일도 아니다.(BDF 파일에서 나타나는 U+31FF 이상의 영역에서 유용한 문 자들이 실제로 존재하며, 이러한 문제가 해결될 그 날만을 기다리고 있다. 그러나 그 들은 현재는 -1로 인코딩되며, 따라서 X 서버에 의해서 무시된다.)
  - Bruno Haible은 서버에서 클라이언트로 압축시켜 XCharStruct를 전송하고, 같은 폰트를 로드하는 다수의 사용자를 위한 Xlib의 공유 메모리를 사용하는, XFree86 4.0을 위한 BIGFONT 프로토콜 확장자(extension)를 작성하였다.

  이러한 주변의 일들은 XFontStruct가 드물게 사용하는 폰트들에게 적합하지 않다는 잠재적인 문제를 해결해 주지 않는다. 그러나 그들은 API 혹은 클라이언트 소스 코드 를 변경할 필요 없이 중요한 효율 향상을 제공한다. 한가지 진정한 해결책은 XFontStr uct를 배열과는 대조적으로 문자를 저장하는 정렬 리스트나 해시 테이블을 포함하는 훨씬 유연한 무언가로 확장 혹은 대체하는 것이 될 것이다. 이러한 XFontStruct의 재 설계는 또한 결합 문자와 인도 문자의 연결선을 동시에 사용하기 위해 급히 필요한 대 비책이 될 것이다.

* **Keysyms:** 지금까지 정의된 바로는 keysyms는 단지 아주 작은 Unicode 목록 만을 포함한다. Markus Kuhn은 U-00000000 부터 U-00FFFFFF까지의 범위내의 모든 UCS 문자는 0x01000000 부터 0x01ffffff까지의 범위내의 keysym 값으로 나타낼 수 있다고 제안하였다(그리고 xterm에서 이것을 구현하였다). 이것은 널리 인정되고 있는 것처럼 전체 31비트의 UCS 공간을 지원하지 않는다. 그러나 이것은 UTF-16과 그 이상의 모드에서 나타낼 수 있는 U-0010FFFF까지의 모든 문자들을 지원하며, ISO가 더 큰 영역의 UCS 코드들을 할당할 것 같지는 않다(사실 미래에는 ISO 10646으로부터 U-0010FFFF를 넘는 코드 공간을 없애자는 제안이 있다). 따라서 유니코드 문자 U+ABCD를 얻기 위해서는 keysyms 0x0100abcd를 직접 사용할 수 있다. 또한 고전적인 keysyms와 UCS(X11 표준에 들어가야만 할 것 중의 하나이다) 사이를 변환시키기 위해 제안된 테이블에 관한 소스 코드인 xterm의 *keysym2ucs.c* <http://www.cl.cam.ac.uk/~mgk25/ucs/keysym2ucs.c> 파일을 보라. Markus는 또한 X 프로토콜 표준 부록 *A: KEYSYM 인코딩* <http://www.cl.cam.ac.uk/~mgk25/ucs/X11.keysyms>의 개정판 초안 을 작성했다. 그것은 UCS 상호 참조 테이블을 추가한 KEYSYM 인코딩(*PDF* <http://www.cl.cam.ac.uk/~mgk25/ucs/keysyms.pdf>)이다.
* **결합 문자:** X11의 세부 사항을 보면 결합 문자를 어떤 식으로도 지원하지 않는다는 것을 알 수 있다. 폰트 정보에는 액센트를 자동으로 붙이는데 필요한 데이터가 부족하다(예들 들면, TeX에는 모든 폰트에 액센트를 자동으로 붙이는 기능이 있다). 다양한 사람들이 잉크를 가지고 문자의 왼편에 폭이 없는 문자(zero-width characters)를 사용하는 가장 간단한 결합 문자 겹쳐쓰기(simplest overstriking combining characters)를 구현하기 위해서 실험을 해왔다. 그러나 이것을 구현하는 방법에 관한 세부 사항은 실제로 명시되지 않고 있다. (예를 들면, CharCell과 모노스페이스 폰트(monospaced fonts)는 폭이 없는 문자를 허용하지 않는다.) 그러므로 이것은 아직 폭넓게 확립된 기술이 아니다.
* **연결선:** 인도 서체는 연결선 치환을 지원하는 폰트 파일 포맷을 필요로 한다. 이것은 지금으로써는 결합 문자처럼 X11의 세부 사항에 포함되어 있지 않다.
* **UTF-8 로케일:** X11 R6.4 샘플은 지금으로써는 UTF-8 로케일을 위한 어떠한 지원도 포함하지 않고 있다. 과거의 UTF 로케일은 있지만, 그것은 불완전하며 현재에는 쓸모없는 UTF-1 인코딩을 사용한다. UTF-8 로케일을 구현하기 위해서는 일반적인 인코딩 변환 과정이 필요할 뿐만 아니라 여러가지 키보드 등록(entry) 방법, 존재하는 ISO 8859와 keysym 키보드를 UCS로 매핑하여 배치하기, 합성 키 (compose key)를 위한 극도로 확장된 지원 및 한글과 한자 등록 지원(entry support)을 입력하기 위한 임의 문자의 *ISO 14755* <http://www.cl.cam.ac.uk/~mgk25/volatile/ISO-14755.pdf> 16진수 등록이 필요하다.
* **샘플 구현(sample implementation):** xterm, xfontsel, 윈도우 매니저등과 같은 고전적인 표준 툴을 위한 유니코드 지원 뿐만 아니라 수많은 광범위한 유니코드 표준 폰트들이 샘플 구현에 추가되어야만 한다. 이러한 부분에 관한 몇몇 연구들이 이미 XFree86 내부에서 이루어지고 있으며 그 밖의 일들은 이전의 중요 사안들이 아직 해결되지 않았다는 사실로 인해 현재 연기되고 있다.

이러한 이슈들을 연구하는 작업이 진행 중인가? 모르겠다. 나는 몇 일에 걸쳐 X 컨 소시움을 공식적으로 계승한 단체이자 X11 표준과 샘플 구현의 관리자 역할을 하는 Op engroup인 *X.Org* <http://www.x.org/>에 접촉하려 시도했지만, 그들로부터 아직 쓸모 있는 어떠한 대답도 얻지 못했다(X.Org의 한 멤버인 XFree86에도 접촉했지만, 역시 대답을 얻지 못했다).

제 25 절 이런 문제에 관한 좋은 메일링 리스트는 있는가?

반드시 *unicode@unicode.org* <http://www.unicode.org/unicode/consortium/distlist.html> 메일링 리스트에 가입해야만 한다. 이것이 표준 작 성자와 많은 다른 도사들의 의견을 듣기 위한 최선의 방법이다. 기부금을 내고자 한다 면, *unicode-request@ unicode.org* <mailto:unicode-request@unicode.org>에 "subscribe"라는 제목과 함께 "subscribe *YOUR@EMAIL.ADDRESS* unicode" 라는 내용으로 메시지를 보내면 된다.

또한 GNU/Linux 시스템에서 일반적으로 사용하는 애플리케이션을 위한 더욱 향상된 UTF-8 지원책을 연구하기 위해서 쓰이는 **linux-utf8@nl.linux.org** 메일 링 리스트가 있다. 기부금을 내고자 한다면, "subscribe linux-utf8"이라는 한 줄의 내용을 적어 *majordomo@nl.linux. org* <mailto:majordomo@nl.linux.org>로 메시지를 보내라. 또한 *linux-utf8 archive* <http://www.linux.eu.org/lists/linux-utf8/>를 다운받을 수 있다.

Xlib와 X 서버에서의 유니코드 지원에 관한 적절한 메일링 리스트는 *fonts@xfree86.org* <http://XFree86.Org/mailman/listinfo/fonts>와 *i18n@xfree86.org* <http://XFree86.Org/mailman/listinfo/i18n>가 있다.

제 26 절 더욱 상세한 참고 자료

* Bruno Haible의 *Unicode HOWTO* <ftp://ftp.ilog.fr/pub/Users/haible/utf8/Unicode-HOWTO.html>.
* *The Unicode Standard, Version 3.0* <http://www.amazon.com/exec/obidos/ASIN/0201616335/mgk25>, Addison-Wesley, 2000. 폰트와 문자에 관계된 일을 하고자 한다면 반드시 이 문서의 복사본을 구해야 한다.
* Ken Lunde의 *CJKV Information Processing* <http://www.amazon.com/exec/obidos/ASIN/1565922247/mgk25>, O'Reilly & Associates, 1999. 동 아시아 문자셋에 관심이 있다면 틀림없는 가장 유용한 책이다.
* *Unicode Technical Reports* <http://www.unicode.org/unicode/reports/>
* Mark Davis의 *Unicode FAQ* <http://www.unicode.org/unicode/faq/>
* *ISO/IEC 10646-1:1993* <http://www.iso.ch/cate/d18741.html>
* *Frank Tang의 Internationalization Secrets* <http://people.netscape.com/ftang/i18n.html>
* *IBM's Unicode Zone* <http://www.ibm.com/developer/unicode/>
* *Unicode Support in the Solaris 7 Operating Environment* <http://www.sun.com/software/white-papers/wp-unicode/>
* *introduction of UTF-8 under Plan9* <ftp://ftp.informatik.uni-erlangen.de/pub/doc/ISO/charsets/UTF-8-Plan9-paper.ps.gz>에서 Rob Pike와 Ken Thompson이 작성한 The USENIX paper는 1992년에 이미 UTF-8(그 시기에는 UTF-2로 불렸다)로 완벽하게 수정한 첫 번째 운영체제에 대해서 보고하고 있다.
* *Li18nux* <http://www.li18nux.net/>는 리눅스를 위한 유니코드 지원을 강화하기 위해서 몇몇 리눅스 배포업자들에 의해서 시작된 프로젝트의 일종이다. 그 프로젝트는 최근에 공공 비평을 위해서 *Li18nux 2000 Globalization Specification* <http://www.li18nux.net/root/LI18NUX2000/li18nux2k_draft.html>을 발행했다.
* *Online Single Unix Specification* <http://www.UNIX-systems.org/online.html>은 wcwidth()와 같은 확장 함수을 포함한 모든 ISO C Amendment 1 함수를 포함하고 있다.
* Open Group의 *ISO C Amendment 1* <http://www.unix-systems.org/version2/whatsnew/login_mse.html> 요약서.
* *GNU libc* <http://sourceware.cygnus.com/glibc/>
* *The Linux Console Tools* <http://lct.sourceforge.net/>
* 유니코드 컨소시엄 *character database* <ftp://ftp.unicode.org/Public/UNIDATA/>와 *character set conversion tables* <ftp://ftp.unicode.org/Public/MAPPINGS/>는 유니코드 관련 도구를 개발하려는 사람에게는 필수적인 자료이다.
* *Microsoft* <http://www.microsoft.com/globaldev/reference/WinCP.asp>와 *Keld* <ftp://dkuug.dk/i18n/WG15-collection/charmaps/>s WG15 archive">의 변환 테이블도 또한 유용하다.

* Michael Everson의 *ISO10646-1 archive* <http://www.indigo.ie/egt/standards/iso10646/pdf/>는 더욱 최신의 ISO 10646-1 amendments의 온라인 버전에 더해 그 밖의 많은 유익한 내용을 포함하고 있다. 또한 그의 *Roadmaps to the Universal Character Set* <http://www.indigo.ie/egt/standards/iso10646/ucs-roadmap.html>을 볼 수 있다.
* *The Universal Character Set (UCS)* <http://www.stri.is/TC304/guidecharactersets/guideannexb.html>에 대한 소개.
* Otfried Cheong's essay on *Han Unification in Unicode* <http://www.cs.uu.nl/~otfried/Mule/unihan.html>
* *AMS STIX* <http://www.ams.org/STIX/> 프로젝트는 Unicode 4.0과 ISO 10646-2를 위한 수학 문자를 수정 및 확장하는 연구를 하고 있다.
* Jukka Korpela의 *Soft hyphen (SHY) - a hard problem?* <http://www.hut.fi/~jkorpela/shy.html>은 U+00AD를 둘러싼 논쟁에 관한 멋진 토론 자료이다.
* James Brigg의 *Perl, Unicode and I18N FAQ* <http://rf.net/~james/perli18n.html>.
* Mark Davis는 *Forms of Unicode* <http://www-4.ibm.com/software/developer/library/utfencodingforms/>에서 UTF-8, UTF-16 및 UCS-4(현재 정치적 이유로 UTF-32로도 불린다) 사이의 협정에 관해 논의하고 있다.
* Alan Wood는 *Unicode and Multilingual Support in Web Browsers and HTML* <http://www.hclrss.demon.co.uk/unicode/>에 좋은 내용을 실었다.
* *ISO/JTC1/SC22/WG20* <http://anubis.dkuug.dk/jtc1/sc22/WG20/docs/projects>은 *International String Ordering (ISO 14651)* <http://anubis.dkuug.dk/jtc1/sc22/WG20/docs/projects/n731-fdis14651.pdf> 과 *Cultural Convention Specification TR (ISO TR 14652)* <http://anubis.dkuug.dk/jtc1/sc22/WG20/docs/n690.pdf> (예를 들어, 와이드 문자 출력의 문자 변환을 지원하는 POSIX 로케일 포맷의 확장)과 같은 다양한 유니코드 관련 표준을 만들어 냈다.
* *ISO/JTC1/SC2/WG2/IRG* <http://www.cse.cuhk.edu.hk/~irg/> (Ideographic Rapporteur Group)
* *Letter Database* <http://www.eki.ee/letter/>는 언어와 문자셋 및 명칭에 관한 질문에 답변을 제공한다.
* 새로운 중국의 코딩 표준 <ftp://ftp.oreilly.com/pub/examples/nutshell/cjkv/pdf/GB18030_Summary.pdf>GB 18030은 UCS를 완전히 보장하는 단순화된 중국어를 위해 폭넓게 사용되는 GB 2312 인코딩의 구 버전과 호환 가능한 확장 기능이다.

나는 이 문서에 새로운 내용을 매우 자주 추가할 것이다. 그러므로 규칙적으로 이 문서를 체크하거나 당신에게 이메일로 통지해 주는 *Netminder* <http://www.netmind.com/URL-minder/new/register.html>를 이용하여 문서의 변화를 체크하라. 더욱 향상된 UTF-8 지원을 위한 freeware community에서의 광고 뿐만 아니라 *제안* <mailto:Markus.Kuhn@cl.cam.ac.uk>은 매우 환영한다. 리눅스에서 UTF-8을 사용하는 것은 매우 낯선 일이다. 그러므로 다음 몇 달 후의 이 문서에 많은 진전이 있을 것이다.

Ulrich Drepper, Bruno Haible, Robert Brady, Shuhei Amakawa와 가치 있는 비평을 해준 다른 많은 이들과 지원을 해준 SuSE GmbH, Nürnberg에게 특별한 고마움을 표한다.

Markus Kuhn <http://www.cl.cam.ac.uk/~mgk25/> <Markus.Kuhn@cl.cam.ac.uk>
created 1999-06-04 -- last modified 2001-02-06 --http://www.cl.cam.ac.uk/~mgk25/unicode.html