

Slide 1:

My name is Ken Lunde. I have worked for Adobe Systems for about 14 years. In that entire time, I have been in the Type Department. “Type” means fonts. All of the work I do there is related to Japanese, Chinese, and Korean. Although what I am doing is different from when I started, it's all tied into Japanese, Chinese, and Korean. So there's always a new challenge for me which is the reason why I am still there. What I am talking to you about tonight, today in Japan, is character sets and encodings, and this ties into a standard that has been published in China called GB 18030. I am going to explain to you why this standard is important, not only in China, but in other countries in the world, including Japan. If you have any questions while I am speaking, please ask.

Slide 2: Overview

I will give an overview of what I am going to cover tonight. I want to cover character sets and encodings in a very general way as an overview of why they are important and why we need to care. There is a good chance some of you in this room don't need to care. I will explain why. Next, I will cover the JIS standards only because that gives a good starting point for understanding the rest of my talk. Then, I am moving into a very important standard that has evolved over last decade or so, which is Unicode. Finally, I will wrap up by talking about the standard GB 18030.

Slide 3: Why Do We Need Character Sets?

First of all, why do we even need character sets? My feeling about this is that we need to establish a common set of characters that are commonly used and that is understood by nearly every platform. That is what interoperable means, as you go from platform to platform, whether it's Macintosh, Windows, Linux, or Unix. That is, the character set is what's understood on those platforms. A couple of interesting trends about character sets is that over the years they tend to grow, they don't shrink or get smaller. This is done in a couple of different ways. Unicode's approach is to come out with new versions. The current version of Unicode, Version 4.1, is approximately three times the size of the original, Version 1.0. Other countries, such as Japan, do this by issuing new standards. The latest one that was issued is JIS X 0213. This supplements the original standard that is now called JIS X 0208.

Slide 4: Why Do We Need Encoding?

Next, why do we need encodings? The main reason is so that our computers can process

these character sets. So you can display the text. You can send it to your friends and print it on your printer. You can do a lot of things. For this reason, because multiple platforms exist based on different architectures, there are multiple encodings, meaning that the same character set can be represented in more than one way, sometimes depending on the computer, and sometimes depending on the application you are using. The most important thing about any encoding that is ever developed is that it must be interoperable. It must be able to be converted from one encoding to another. The encoding that is considered the default, or de facto or the preferred, encoding changes over time.

Slide 5: Do I Need To Care About This?

As we talked about character sets and encoding, ask yourself a question: do I need to care about this? This actually matters to me. My job is to know this information. It really depends on the extent you use computers. Typical users, let's say, somebody who wants to write an e-mail or write a letter, they generally don't need to know this. All they care about is to type in the characters they need, that they display correctly, and that they can print them. Do the people receive their e-mail okay? That is what they care about. All of the work that the developers do, including input method manufactures, protect or insulate the users from knowing this information. People that we call power users, they tend to care about this stuff simply because they are curious people. They want to know what is actually inside the software. How does it work? These people tend to push the software to their limits. They tend to find things about the software such as what I would call undocumented features. Some people refer to these as bugs. In some cases, the bugs have a negative value, meaning that they prevent you from doing your work. But sometimes they have positive values, meaning that the software does something that it was not designed to do. It's still a bug, which I call an undocumented feature. The people who must care about this are the developers, because their job is to protect millions of users out there from knowing this information because generally they don't need to know this information.

Slide 6: JIS Character Set Standards

I think most of you are familiar with this slide in terms of what's available in Japan. The character set standards in Japan are called JIS standards, and JIS stands for Japanese Industrial Standard. Still in general use today is JIS X 0201, which is based on ASCII with the hankaku katakana attached. At this point, I think only the ASCII portion is important. The next standard that was issued in 1978 is what we now call JIS X 0208.

The last time that this standard was issued was 1997. The latest standard that was issued was JIS X 0213. That was issued originally in the year 2000 and reissued in the year 2004. This ended in a total of more than 10K characters for the combined standard. An older standard that came out in 1990 is JIS X 0212, which I consider deprecated. Deprecated simply means it is no longer being actively developed and the people who made it wish that they didn't make it. I want to cover information about how these standards are related because they do have relationships with each other. You may find it interesting.

Slide 7: Why Does JIS X 0212 Persist?

I mentioned that one of the standards, JIS X 0212, has been deprecated, meaning that it is no longer being actively developed. So the question is why is this standard still persist? Why are people like me still talking about it? Why can't I ignore it? There are a couple of reasons. First, the characters in the standard are included in Unicode and Unicode today is the most dominant encoding. All the major platforms, Windows Macintosh, use it. Unicode has many advantages. Everything has its good side and bad side. I believe that Unicode has a lot of good things going for it. The second reason why the standard still persists is because removing these characters from Unicode is a very, very difficult thing to do. It's so difficult that it won't happen. Those characters have been preserved in Unicode until Unicode no longer exist. Another reason why the standard still exists is approximately half of them still live on in the newer standard, JIS X 0213. By my calculation, 2743 kanji are common between these two standards.

Slide 8: JIS Encodings

I want to cover these very quickly. Something I wish to cover here is that many encodings have been used in Japan. These are what I refer to as legacy encodings. These are the ISO-2022-JP 7-bit encodings, and there are two 8-bit encodings, Shift-JIS and EUC-JP. The other encodings that can handle Japanese are Unicode ones.

Slide 9: Unicode Encodings

Today there are three major Unicode encodings. All these are called UTFs: UTF-8, -16 and -32. I am going to cover these a little bit more in detail in another slide. What's important is that all three of these encodings represent the same set of code points, which make up 17 planes. The first plane is called the BMP, plus 16 additional planes. Each plane can encode over 65,000 characters. If you do the math, you can see that there are over one million code points. There is no problem in terms of running out of

code points at any time soon. We have plenty to work with for many years or decades to come. What's also important about Unicode is that the dominant OSes today, Windows and Macintosh, support Unicode at the OS level. If you want to write applications that work on these platforms, you have to use Unicode or you have to work with Unicode in some way.

Slide 10: BMP Versus Non-BMP

I mentioned BMP, which stands for Basic Multilingual Plane. This is also known as Plane 0 because in the computer world, scientists like to start every thing at zero instead of one. So we call this Plane 0. These represent the first characters that were incorporated into Unicode, and because of that, they tend to be the most frequent ones used. The total number of available code points in the BMP is not the complete set of 65,000, but rather it's been reduced by roughly 2,000 because of an extension mechanism for handling one of the other encodings. Still, there is a large number of code points, 63,488 to be precise. Outside the BMP there are over one million code points available in 16 additional planes that are called Supplementary Planes. For Japanese, what's important about those planes is Plane 2. Plane 2 holds Extension B, which has nearly 44,000 kanji.

Slide 11: Which Unicode Encoding Is Best?

One question I am always asked by people is which of the three Unicode encodings is the best. Which is the best one and which one should I support? The big hint I like to give them is that one needs to support all of them. The reason is because interoperability between them is very important. You never know what sort of client you may be working with. You may receive code in UTF-8 although your application deals with UTF-16 internally, so you clearly need to interoperate with all these encodings. That's not hard. There are clear cut algorithms for moving between these different encodings and they are also published. Some of the advice that I do give for people who want to pick one for the primary encoding is if the ASCII compatibility is important to you, then UTF-8 is your friend. Any valid ASCII code point is a valid UTF-8 code point. If you care about compactness, in the case of Japanese, the best encoding is UTF-16. It provides the best level of compaction for the characters used by Japanese. What is very interesting to note is that once you go beyond the BMP, that area has more than one million code points. It doesn't matter what encoding you are dealing with, you must process four bytes per character. In UTF-8 encoding, you are dealing with four separate bytes. In UTF-16, you are dealing with two 16-bit units. Again, the total length is four

bytes. In UTF-32 encoding, you are dealing again with four bytes, but in a single 32-bit entity. What you see on your screen is the first character in Plane 2 represented by the notation U+20000, with hexadecimal representations in each encoding shown here. As you can see, they are quite different.

Slide 12: How Is JIS X 0213 Supported?

About the new standard that came out in Japan, JIS X 0213, because of the timing when it came out, I think the best way to support this standard is through Unicode. There are a couple of reasons for that. One of the next slides covers the Shift-JIS encoding issues. When they extended Shift-JIS for JIS X 0213, they didn't extend Shift-JIS encoding, but instead they used up virtually every remaining Shift-JIS code point in order to support the new standard. Unicode now, as you know, has more than one million code points.

Slide 13: JIS X 0208 Versus JIS X 0213

In essence, JIS X 0213 fills all the gaps, or fills virtually all the gaps, left behind by the previous standard, JIS X 0208. If you look at the Shift-JIS code space, there is total of 11,280 available code points. There exists what I call the original block and an extended block. If we add the characters in the two standards together you find out that only 47 code points are left unassigned. Most of them are reserved and you cannot touch them.

Slide 14: JIS X 0208 + JIS X 0213 Sample

Just to give an example of how these two standards are merged, what you see here is a Shift-JIS encoding table for both standards. This represents rows 47 and 48. The portion before the yellow highlighted section represents row 47, and is also the very end of JIS Level 1. The portion after the yellow highlighted section represents row 48, and is the very beginning of JIS Level 2, with a total of 94 characters. What you have in between, highlighted in yellow, are the characters that are added by JIS X 0213. This is from the original 2000 version of JIS X 0213. In the year 2004, they added 10 additional kanji. On the left is shown one of the ten kanji. It was assigned to the reserved code point between JIS Level 1 and the yellow highlighted section. On the right is shown another of the ten additional kanji, and it was assigned to the other reserved code point, at the end of the yellow highlighted section. The function of these reserve code points was to provide some sort of separation, something visual, between the two standards. But when they wanted to add 10 additional characters to the standard, they had to start using up these reserve code points. So, for example, if you look at the 2004 version of the standard, you will see these characters at the code points that were once reserved.

Slide 15: JIS X 0212 Versus JIS X 0213

I mentioned before in a previous slide that almost half of the kanji in JIS X 0212 are shared by JIS X 0213. There are also some kanji in these standards that are unique to each of them. There are still 3,058 kanji unique to JIS X 0212. Almost 1,000 are unique to the newer standard, JIS X 0213. Among the 2,743 kanji that are common, I have detected at least 32 minor glyph differences. In that, if you look at the glyphs in both standards, although they share the same Unicode code point, the way they are rendered or shown in the standards is different. The best example I found is what I have here. What you have on your left is from JIS X 0213, and what you have on your right is from JIS X 0212. I consider the JIS X 0213 form to be more correct. If you cannot see the difference, the difference is in this element (upper-right). In some cases, the differences are even more subtle than you see here. This is one that is more obvious than others.

Slide 16: Must I Support Unicode?

One of the first things I would like to say about Unicode here is to ask whether I need to support the Unicode. I can say “yes” for three reasons. One is that Unicode provides maximum interoperability, where you want to be able to work with different operating systems that other people are using. You want to have maximum interoperability, and you want maximum stability. Unicode has established policies on stability, meaning that once the character gets into Unicode, it will not be removed. This is the reason why JIS X 0212 kanji will never be removed from the standard. I have found at least one character that got into Unicode in error. They won't correct it because of the stability policy. It happened to be a Japanese character (see U+332C).

Slide 17: JIS Standards Versus Unicode

Here we cover Unicode in the context of Japan. It is very important to know that all the JIS standards, meaning that all of the JIS character sets are covered completely in Unicode. The standards JIS X 0208 and JIS X 0212 were there from the start, at the very beginning of Unicode. Support for the newer standard came a little bit later. It wasn't until Unicode version 3.2 that JIS X 0213 was fully supported. The way Unicode supports JIS X 0213 is outside the BMP for 303 of its kanji. If you want to be JIS X 0213 compliant in a Unicode environment, you have to go outside the BMP.

Slide 18: Introducing GB 18030-2000

The standard I would like to introduce to you is a standard that was developed in Mainland China, also called the People's Republic of China. What's important about this standard is that the standard is government-mandated, meaning that the Chinese government requires any software sold inside its border to be completely compliant with the standard. A lot of people, I think, are guessing what does compliance actually mean. That is actually one of the things I am taking on at Adobe. I can share some of my perspective here. A very important point about this new standard is that it's completely code point compatible with Unicode, meaning that any Unicode code point can be represented in GB 18030 although its own encoding is slightly larger. Anything you have done for Unicode support immediately translates to GB 18030 support. I will explain why in the next few slides. In terms certifying software for sales in that country, there is already a well established certification process. I will tell you a little more about that. But first, how many people here have heard of this standard? How many people know what this standard is?

Slide 19: Should I Care About GB 18030?

The first question is “do I really need to care about this standard?” The answer is yes. The reason for that is because Mainland China is one of the fastest growing markets on the planet. So, if you want to tap into that growing market, you need to support this standard there. Basically, no GB 18030 support translates to no market entry into that country. GB 18030 support doesn't guarantee success in that market. It only lets you enter the market. I will come back to talk about how the certification process has been done in terms of grading. Among the good points of GB 18030 is that if you do get GB 18030 certification, it confirms that you have done a very good job at supporting Unicode. In order to be certified for the standard, you really need to have proper Unicode support. If you don't already support the GB 18030 standard, doing so in the context of Unicode is the right way, and has other benefits. By forcing companies to comply with the standard, it's leading the companies to properly support Unicode. There are a lot of benefits. How many people here have experience certifying software for the GB 18030 standard? We have done this at Adobe, and I believe the first software we had certified was Photoshop Version 7.

Slide 20: GB 18030 Character Set Details

If you look at this standard in terms of its character set, and if you are familiar with some of the older standards used in Mainland China and how Unicode is related to them, we can describe it as starting off with the character set called GBK, and we add in what

is called CJK Unified Ideographs Extension A, which is often just referred to as Extension A. Extension A includes about six and a half thousand additional Chinese characters, added to Unicode. They also added support for minority scripts. I am going to cover those in later slides. How many characters does GB 18030 roughly have? That is about 30,000, with the support of extension B. An interesting thing happened recently. Unicode Version 4.1 was issued, and as a result it became possible to support the GB 18030 standard without using the Private Use Area (PUA). These are the characters that are not in Unicode, the PUA may be referred to as the “gaiji” area of Unicode. This is a rather important development.

Slide 21: GB 18030 Encoding Details

In terms of encoding, any Unicode encoding can be considered a valid GB 18030 encoding. This is simply the best way to support this standard. The standard itself does have its own encoding, but doesn't really have a name, but I refer to it as its native or legacy encoding, and its a mixed 1-, 2- and 4-byte encoding. The one-byte portion is simply ASCII. The two-byte portion is identical to the previous standard, GBK, with almost 24,000 code points. The four-byte region handles everything else, such as Extension A and the minority scripts. All of that is handled in this new four-byte region that has over one and a half million code points, which is 50 percent larger than Unicode.

Slide 22: GB 18030 & Minority Scripts

I mentioned minority scripts. The four minority scripts that should be supported for products that handle GB 18030 are Mongolian, Tibetan, Uyghur, and Yi. One thing that many people don't know is that there is some history here that has been overlooked. The next slide covers that.

Slide 23: GB Standards For Minority Scripts

Many people believe that GB 18030 is the first time when these minority scripts were supported in the context of a Chinese standard. As you can see by the standards that I have on the slide, that is not the case. In fact, one of them dates back to 1987, for Mongolian. So, the advantage of GB 18030 is that it puts these minority scripts into a single standard. It is much easier to support these scripts in a single encoding, such as Unicode, than to support them in their own legacy encodings. Shown on the slide are the first pages of the legacy standards for these four minority scripts.

Slide 24: Minority Script Examples

These are examples of what the minority scripts look like. Note how Mongolian script is vertical, but is from left to right, not right to left. (Someone from the audience asked about Korean.) That is a good point to bring up, because Korean is a minority script. I am often wondering why Korean, specifically hangul, doesn't come up in GB 18030 discussions, because Mainland China has issued a GB standard for Korean, GB 12052-89. It is completely incompatible with any standard put out by South Korea or North Korea. Yes, Korean is a minority script.

Slide 25: GB 18030 Recommendations

Some of the recommendations I usually give to people when I am asked about GB 18030 is that for now you can support GB 18030 by supporting only the BMP, but at some point in the future, and it's not really understood when it will be, is that Extension B characters in Plane 2, of which there are nearly 44,000, may become part of the GB 18030 requirement. The interesting thing about that is the moment you start talking about Extension B, you have exhausted the number of glyphs you can put into today's font formats. All the major font formats of today, meaning TrueType, OpenType, and CID, have a very hard and fixed limit of 64K (65,536) glyphs. So, supporting Extension B goes well above that barrier. Another recommendation is to support the minority scripts, if possible. We have found that Yi characters are very similar to Chinese. So, you can handle them pretty much as you do for any kanji. The other minority scripts, Mongolian, Tibetan, and Uyghur, require complex layout. If you want to support these scripts correctly, you have to handle very non-trivial things in your fonts and in your applications to support those correctly. All of the implementations of GB 18030 that currently support minority scripts do not do proper layout for Mongolian, Tibetan, and Uyghur. Adobe is currently researching these minority scripts. We are going to support proper layout for them. This is a real challenge because these are quite different than anything we have ever encountered before.

Slide 26: GB 18030 Testing

In terms of testing, obviously before you submit your application to China for certification you want to do your own testing. Make sure that you feel that your software is going to be GB 18030 compliant before you actually do the certification. The process of testing is to identify problems. When you identify them, it is important to fix the deficiency if you can. If you cannot, you must document the deficiency. We have learned that the certification agency much prefers that they find the deficiency in

documentation they read rather than finding a deficiency during their own testing. The reward for that is that you demonstrate to the certifying agency that you know your product and the GB 18030 standard. A couple of other recommendations are as follows. If your application does not have its own fonts, use the OS fonts to your advantage. Use the fonts that Microsoft Windows bundles with its GB 18030 support package. Also use the OS level APIs to your advantage. For example, if you want to interoperate with the legacy encoding, you don't need to do so on your own, and instead let the OS do the work. They have already become certified. They have done a lot of work for you. Take advantage of that.

Slide 27: GB 18030 Certification

The actual certification process is done by an organization called CESI. It stands for the Chinese Electronics Standardization Institute. If you are really curious about it, you can go to their web site. The certification process itself is performed on Windows. And, as I mentioned before, this agency prefers to find documented deficiencies. They don't want to see undocumented deficiencies. This demonstrates to them that you have done your work, you know your our product, and you know their standard. The original certification process resulted in a letter-based grade. So, for example, an application could get a letter grade such as A or A+. At Adobe, when we have discussed the certification process, I told everybody that A is okay. Some people said that we need A+. They said that we must have A+. I kept questioning why. Why do we need A+? A is still passing, right? What does the A+ grade give you? Bragging right? Who are we going to brag to? Does anybody care? The answer is no. What I learned recently, which confirmed my instinct, is that they changed the actual grading from a letter-based grade, as I thought meant nothing in the real or practical world, to a simple pass or fail grade. Your application or OS now passes or fails, in terms of GB 18030 certification. Like I said, GB 18030 certification is not a guarantee of success in that market. It simply lets you get in the door. That is it. Nothing more. It happens to be a door that is heavily guarded, locked, and with one way in. But it's still a door.

Slide 28: Unintended Consequences

Some of the unintended consequences of supporting this standard come first as merely a barrier. As I mention many times, certification is not a guarantee of success. I am saying that many times for a good reason. How your software performs in user's hands is more important than your software is certified, in terms of measuring success. Your success depends on your product's quality, and also how you market it. As I mentioned, Unicode

greatly aids the process of certification. At the same time, certification encourages Unicode support. They reinforce each other.

Transcribed by Kazuo Yana, Hosei University

Copyright © 2005 Kazuo Yana, Hosei University. **All rights reserved.**