CODED CHARACTER SET CONVERSIONS AND DATA LOSS: PROVIDING INTEROPERABILITY WHILE PREVENTING LOSS

1. INTRODUCTION

As interest has grown in providing code set interoperability for DCE and other OSF technologies, potential solutions have focused on how to send data between systems using different encodings and less on what happens once the systems get it. I believe we need to address this second part of the puzzle. This paper describes the issues, recommends a course of action, and describes the effects of that recommendation on various OSF technologies.

2. PRELIMINARIES

This section provides information about character encoding terminology, and also briefly describes code set conversions. For other context, see [RFC 13.0] and [RFC 23.0].

2.1. Definitions

This paper uses the following terms.

(a) CHARACTER SET — A group of characters without any associated encoding. Examples of character sets are the English alphabet, Japanese Kanji, and the characters needed to write European languages.

(b) CODED CHARACTER SET, or CODE SET — A mapping of the members of a character set to specific numeric code values. Examples include ASCII, ISO 8859-1 (Latin-1), JIS X0208 (Japanese Kanji).

(c) ENCODING METHOD — In order to use computers in some countries, it may be necessary to combine multiple code sets in a data stream. An encoding method provides the rules for combining sets and recognizing the set to which a given character belongs. Examples are EUC (Extended UNIX Codes), Japanese SJIS, and Taiwanese Big 5.

(d) PORTABLE CHARACTER SET (PCS) — A minimum group of characters guaranteed to be supported on all compliant systems. The most commonly used PCS includes all the graphic characters in ASCII — that is, letters A-Z and a-z, digits 0-9, and common punctuation. The PCS may be encoded in multiple ways.

Although code sets and encoding methods are different, they are treated nearly the same with respect to interoperability issues. Unless otherwise stated, the term ‘‘code set’’ refers to both concepts. The distinction between character and code sets, however, *is* important.

2.2. General Description of Conversions

Computer systems today handle data encoded in many different ways. Data may be encoded in an official or de facto standard code set (for example, ASCII, Latin-1, AJEC [Japanese EUC], a form of ISO 10646), or in a vendor-specific encoding (IBM’s EBCDIC-based code sets, HP’s ROMAN8, different implementations of Japanese SJIS). Heterogeneous networks and distributed computing models make it increasingly likely that multiple systems will process a given set of data, and that the various systems will need or want the data’s encoding to change. DCE 1.0 addresses part of the issue because it includes logic that automatically converts data between the two major American code sets: ASCII and U.S. EBCDIC (code page 500). DCE’s existing American-only solution, however, clearly does not meet worldwide needs.

A code set converter maps the encoded value of characters in a source code set to the encoded values of the equivalent characters in a destination code set. For example, the encoded value of uppercase A is `0x41` in ASCII and `0xc1` in U.S. EBCDIC. An ASCII to EBCDIC converter therefore converts any `0x41`’s in the input data stream to `0xc1`’s. `iconv()` is the standards-defined tool for doing conversions. Users provide `iconv()` with source and destination code sets, and the name of the file to be converted. In most cases, `iconv()` is run manually, although it can be part of a command script.

Converters work well when the source and destination code sets encode the same character set. If the character sets don't match, however, there is a possibility (or probability) of data loss. Consider a Latin-1 to ASCII conversion. Both code sets contain the PCS, but Latin-1 also contains a group of letters with diacritics used in Western European languages. If the input Latin-1 data contains any of the letters-with-diacritics, they are lost during the conversion because they do not exist in ASCII. (Most converters map unsupported characters to a substitute character like a question mark, space, or some other character.)

Data is not always lost when converting between code sets with mismatched character sets. It depends on the input data. Japanese SJIS and Latin-1 obviously encode different character sets, but both contain the PCS. Therefore, a data stream that only contains portable characters can be converted successfully between these two sets. However, it is incorrect to assume that because conversion works without loss under very restricted circumstances, it works in the general case.

While data loss is not assured when converting between mismatched code sets, it may occur even when the code sets do match. Many Asian code sets include only a subset of the ideographs used in a given language. Because users may need ideographs that aren't in the standard sets, Asian encodings often include a reserved area for user-defined characters (UDCs). There is no standard way to identify individual UDCs, and the group that gets assigned to the UDC area in one code set most likely differs from the group assigned to the same area in another code set. A conversion typically either loses any UDCs in the input data, or misinterprets them.

3. CODE SET INTEROPERABILITY PROPOSAL

OSF has been asked to provide code set interoperability in DCE 1.1. This interoperability is defined as being the ability to convert data when necessary from its client representation to the representation that a server understands (and, after processing, from the server's back to the client encoding). Any required conversions should be transparent to the user.

The requested default behavior is to convert source data to a universal encoding and then to convert the universally-encoded data into the destination encoding. This default is said to provide full code set interoperability. However, regardless of the way interoperability is provided, or of the use of a universal encoding, a conversion can lose data. Consider these examples:

(a) SJIS to Latin-1.

(b) SJIS to universal to Latin-1.

Assuming the SJIS stream includes Japanese characters, such characters are lost when the data is converted to Latin-1. The loss occurs regardless of whether or not the data makes an intermediate stop in the universal representation.

Automatic code set conversions are fundamentally different from the existing internationalization (I18N) model. The global locale model processes data according to the rules of the current locale. If I set my locale to French and process a file `foo`, the data is treated as if it were French. If I change my locale to Japanese and reprocess `foo`, it is treated as if it were Japanese. The data doesn't change; the way the system processes it does. With code set conversions, however, the data changes according to the rules of the converter modules. That introduces the possibility of data loss.

4. IS DATA LOSS A PROBLEM?

Just because code set interoperability adds the possibility of data loss does not automatically mean we have to worry about it. The following sections describe the reasons for and against worrying about data loss.

4.1. Reasons Not to Worry About Data Loss

Arguments against providing a way to prevent data loss include:

(a) Sites are homogeneous, so data loss won't happen (or will happen so rarely it isn't worth worrying about).

(b) Vendors will convert all server processes to ISO 10646, and 10646 can handle anything.

(c) Any solutions to data loss are too hard to implement or have an unacceptable impact on performance.

It certainly is true that most sites are relatively homogeneous with respect to character sets — a site in the U.S. tends to have local systems running ASCII (or Latin-1) or U.S. EBCDIC, while a Japanese site has SJIS- and AJEC-based systems, and a Western European site may have any of Latin-1, ROMAN8, pc850, or other relatively compatible code sets. In most cases, conversions among these systems will work just fine.

However, even in these apparently-homogeneous examples, conversions may lose data. SJIS and AJEC typically are considered homogeneous encodings, but AJEC contains approximately 6,000 more Kanji ideographs than does SJIS. An AJEC to SJIS conversion would lose any of those ideographs.

And what about other possibilities? It's not farfetched to imagine a site with systems running Latin-1 and Latin-2, or ASCII-only and Latin-1 systems. Other scenarios such as a network running a mixture of Taiwanese EUC and Latin-1 processes are uncommon today, but should become more common given the expected growth of global networks.

If sites are not necessarily homogeneous, there's another possibility that may eliminate the data loss issue. Some predict that in the future, all servers will run ISO 10646 (UCS, Universal Coded Character Set), and so be able to handle virtually all characters. However, if all servers are not UCS-based, data loss remains a possibility.

So how likely is it that all servers will go UCS? Microsoft (with NT) and Apple seem to be moving to all UCS-systems, so it seems likely that their servers will be UCS-based. Other vendors are treating UCS as one of a group of supported code sets, so they may not be as inclined to designate UCS as the only acceptable encoding for servers.

That is in part because there are disadvantages to a UCS-only server. While it may be able to take in virtually all characters a client sends, it probably has to do something with them, which means the server must have a UCS-based locale or locales. If there is a single UCS-based locale, it must be generic instead of being tuned to specific user requirements. For example, a ‘‘universal’’ locale would include one collation order. Individual languages/territories, however, have specific and often contradictory requirements that a single collation order cannot meet.

While a single UCS-based locale may not meet all needs, vendors may also balk at storing multiple UCS-based locales on all servers. In OSF/1 1.1, an average Latin-1-based locale object is 40-45KB, while the larger Asian-based objects vary between 0.5 and 0.8 MB. UCS-based locales would almost certainly be larger, since they probably would define more characters. As noted earlier, sites do tend to be relatively homogeneous, so while some vendors might favor the uniformity of an all-UCS system, others are likely to choose something that more closely meets their specific needs.

Given that many vendors are treating UCS as one of many supported encodings, I believe we cannot assume all servers will be UCS-based, or that we should design code set interoperability support around that assumption.

As for the other argument against worrying about data loss — that potential solutions are too hard — we need to look at some solutions to determine whether they really are too hard. See the section ‘‘Proposed Solution’’ for one to consider.

4.2. Reasons to Worry About Data Loss

Although I don't agree with the arguments against dealing with data loss, there have to be good reasons *for* dealing with it. The main reasons are:

(a) Users.

(b) Users.

(c) Users.

In general, the worst thing a system can do is lose or corrupt data; it often is a fatal error to do so. Admittedly, loss or corruption on a system level typically refers to bits that get garbled or written over or otherwise accidentally lost, while loss in a code set conversion is a planned-for event (`if outside_range map_to_substitute_char`). However, try explaining that difference to a user who just lost all the ‘‘foo’’ characters in his/her file because the server that processed the data happened to be running ‘‘bar’’. The subtleties are likely to get lost amid all the shouting.

Furthermore, when conversion happens automatically and transparently, even users who want to guard against data loss have no realistic way to do so. They must rely on the hope that clients and servers are using encodings that include the same characters. Of course, the conversion will work in many cases, but the only way to find out when it doesn't is after the data is lost. Since the distributed nature of applications often is invisible to users, they may not be aware that they're ''taking a chance.'' This is different from the mostly-manual way `iconv()` gets used today.

I believe it is unacceptable to create the possibility of data loss while providing neither a warning to users that the loss is about to occur, nor a way to prevent that loss.

5. PROPOSED SOLUTION

In order to provide code set interoperability, OSF has been asked to create code set tags. These tags would be passed between clients and servers to determine whether conversion is necessary, and if so, what kind of conversion to perform (i.e., should the conversion go directly from, say, X to Y, or from X to universal to Y?). There are varying opinions about whether OSF should formally register tag values, or adopt a looser tagging model that defines the tag's structure, but not the values.

I recommend that OSF formally register code set tag values and that it also register character set values. (See rationale below.) In the proposed registry, each code set would be associated with one or more character sets. All code sets are required to include the PCS (Portable Character Set), and OSF will choose a universal encoding (probably a form of ISO 10646) that is assumed to be the concatenation of all other character sets. In between, there would be other, smaller character set values like these:

```
/* The character set names shown are just placeholders!
 * They will change as this proposal is refined.
 */
Character Set           Example Encodings
-------------           -----------------
PCS                     ASCII, U.S. EBCDIC,
                            [...  all other code sets]
Western Europe          ISO 8859-1, pc850, ROMAN8
Eastern Europe          ISO 8859-2
Japanese1               SJIS, UJIS
Japanese2               AJEC
Taiwanese1              eucTW, Big 5
  .  .  .                 .  .  .
```

Once code sets and their associated character sets are registered, DCE and other software can use both to determine whether a client-server connection is appropriate. The evaluation might work like this:

```
if (codeset1 == codeset2)
    accept server;
    send data as is;
else if (charset(codeset1) == charset(codeset2))
    accept server;
    convert codeset1 data to universal or
        other accepted encoding;
    send data;
    convert from universal or other accepted
        encoding to codeset2;
else /* charset(codeset1) != charset(codeset2) */
    reject server;
```

An argument against worrying about data loss is that it may adversely affect performance. Therefore, OSF should not require that an application use the character set information. In such a case, the simplified logic might work like this:

```
if codeset1 == codeset2
    send data as is;
else
    convert codeset1 data to universal or other accepted
        encoding, knowing data loss is possible;
    send data;
    convert from universal or other accepted encoding
        (if necessary) to codeset2;
```

5.1. Limitations of Character Set Evaluation

Using character set information as an evaluation criteria isn't a perfect defense against data loss for several reasons. The most important is that character set divisions are often arbitrary, rather than being easily definable and universally accepted. The result is that code sets only rarely encode exactly the same character set.

Many character sets contain the characters from one or more ''scripts''. A script is a system of writing, such as Latin, Cyrillic, and Kanji. Some scripts are used for multiple languages, and some languages use multiple scripts. An example of the former is the Latin script, which is used to write many languages including English, German, Polish, and Vietnamese. Languages that use multiple scripts include Japanese, which uses Kanji, hiragana, katakana, and Latin letters. A ''Japanese'' character set contains the four scripts, while a ''Cyrillic'' character set would support the languages written with that script (Russian, Bulgarian, Serbian, others).

It may seem intuitive that there be a single ''Latin'' character set, and that it contain all characters for all languages that use the Latin script. But it's not that easy. Instead, Latin letters typically are subdivided into smaller, regional character sets including American, Western European, Eastern European, and so on. The Western European set, for example, contains the characters needed to write Western European languages like French, German, and Spanish.

The decision to define any character set typically is a matter of practicality and economics. Computer markets in Western Europe are well-developed and somewhat inter-related, so it makes sense to have a single character set for this region. There is less (although increasing) inter-relation between Western and Eastern European markets, so there are separate character sets for these two regions.

The result is although there is no linguistic justification for a ''Western European'' character set, ISO and many computer vendors have created Western European code sets. However, all have chosen to encode slightly differing character sets. Examples are Latin-1 and ROMAN8, both of which contain characters not in the other. Here's the list:

(a) IN LATIN-1, BUT NOT IN ROMAN8:

- (i) No break space (NBSP).
- (ii) Broken bar.
- (iii) Copyright sign.
- (iv) NOT sign.
- (v) Soft hyphen.
- (vi) Registered trade mark sign.
- (vii) Superscript two.
- (viii) Superscript three.
- (ix) Cedilla.
- (x) Superscript one.
- (xi) Multiplication sign.
- (xii) Division sign.

(b) IN ROMAN8, BUT NOT IN LATIN-1:

- (i) Lira sign (with double bar).
- (ii) Florin sign.
- (iii) Capital letter S with breve.
- (iv) Small letter s with breve.
- (v) Capital letter Y with diaeresis.

(vi) Em dash (?).
(vii) Filled box.

Even though an evaluation based on char sets would show Latin-1 and ROMAN8 as encoding the same set, because each contains some unique characters, conversion from one to the other could lose some data. This problem exists with other code sets that may encode the ''same'' character set. However, keep in mind that:

(a) Although there often are mismatches between versions of char sets, code-set-unique characters tend to be low use and so are not common in data streams.

(b) Despite the fact that some character set divisions are linguistically arbitrary, they are based on practical, everyday business use. If OSF registers character sets, it should favor practical needs over seemingly more elegant groupings.

Using character set as an evaluation criteria has two other disadvantages: it is slightly slower to evaluate than to send data without checking, and evaluation may occasionally produce the ''wrong'' result. Remember the earlier example of a SJIS to Latin-1 conversion in which the data happens to contain only portable characters. This conversion would be rejected. However, this situation is likely to be extremely rare.

One other point — evaluation based on char set does not solve the problem of Asian UDCs. Characters assigned to the UDC areas of Asian code sets are opaque; there's no standard way to determine what character a given UDC code represents. Registering code and character sets does not solve this problem. Rather, converter modules need specific information about the UDCs in source and destination sets to convert the characters properly, and that requires more information than will be in the registry.

5.2. Advantages of Character Set Evaluation

Using character set as an evaluation criteria provides a mechanism for preventing most data loss. It removes the blind connection to servers that cannot adequately process a client's data. It also provides a way to protect users who are not aware they're using a distributed application from unexpected and seemingly unexplainable data loss.

While it would be nice to provide an air-tight way to avoid data loss, doing so would be significantly more complex and probably would impact performance unacceptably. Consider, too, that while evaluation based on character set is not a perfect solution, it is a significant enhancement over a model which provides no way to recognize inappropriate conversions and avoid data loss.

6. REGISTRY DETAILS

It is with some trepidation that I recommend having OSF create a code set and char set registry. This task is not as simple as it may appear, and requires careful planning and execution. Among the details to be determined are:

(a) *Form of registered names.* We could choose among a UUID, an integer, or a string. We're leaning toward an integer because it's easy to pass, easier to define, and can be used by multiple technologies. In any case, even if OSF does not register code and char set values, it must at a minimum provide the format of code set values. An OSF-defined format must exist in order to address code set interoperability.

(b) *Method for associating code sets and char sets.* There must be a way to specify that a given code set consists of multiple char sets. AJEC and SJIS are both Japanese, but AJEC contains approximately 6000 more characters than are in SJIS. Taiwanese Chinese faces a similar situation, with previous versions of CNS 11643 containing about 13,000 characters, and the 1992 version containing 48,000. Character set designations should allow an application to make judgements like these:

(i) SJIS → AJEC always okay.
(ii) AJEC → SJIS maybe not.

(c) *Registration criteria.* Should OSF allow any vendor to register all its proprietary code sets, or should we only register standard encodings like ISO 8859-1? In order for the registry to be useful, we probably will have to allow a vendor to register as many sets as it wants, but we should realize that some vendors have hundreds or even thousands of sets (as Dave Barry would say, I'm not making this up). The sheer numbers mean that the registry is more than a do-in-your-spare-time task.

There is also the issue of determining the character set to which a code set belongs. OSF would need to supply a list of character sets (naturally, we would work with the members to construct this list), and then leave it to the vendors to decide the char set(s) to which each of their code sets belongs. There's no way OSF can individually research each code set, so we would have to rely on the vendor's information.

(d) *Define the registry's uses.* Actually, this is more a matter of setting expectations. In creating a registry, OSF would be providing a mechanism for promoting code set interoperability, and making it possible for a client to determine whether to establish a connection with a server. We would define the registered values that clients and servers could exchange. However, we would not guarantee that either could necessarily handle a registered encoding. Suppose, for example, that the registry contains a value for IBM's pc850. Although servers would be required to recognize the value, they would not be required to have converter modules into or out of pc850. The set of modules on any given server would be implementation-defined.

(e) *Publication details.* Once the registry is set up, how should OSF distribute the information? Should it go with a particular OSF technology, or should the list be available with all technologies? It certainly should be available for OSF members and licensees, but should it also go to other companies or organizations? These details need to be worked out.

While defining code and char set values is important, OSF should not require applications to use this information. Some organizations might not want to register their code and char sets but still want to use DCE's interoperability functionality. Therefore, OSF should define a range of ''private use'' values within the registry. Vendors or applications would be free to use these values without contacting OSF. Of course, interoperability would be more limited for clients and servers using ''private use'' encodings. Only the systems knowing the local values and having the appropriate converter modules would be able to interoperate.

6.1. Registry Rationale

Although a registry involves work to set up and maintain, I believe it's the only effective way to allow code set interoperability. If OSF defines the format but not the contents of code set tags, we will be making the same mistake that standards groups did when inventing locales. X/Open defined a format for locale names, but it said the contents were implementation-defined. ISO and POSIX didn't even define a format. The result was that vendors chose incompatible naming schemes, hampering locale interoperability. Now efforts are underway to standardize locale names.

OSF should learn from the locale experience and provide registered, readily accessible code set values. Then, in order to provide a mechanism for preventing data loss, we also should provide registered character set values.

6.2. Relationship to Other Registries

It would be nice if OSF could avoid the job of registering code and char sets completely, and just point to some other registry. Unfortunately, little exists in this area.

(a) *ISO.* ISO 7350 apparently was designed to register code sets, but it only contains about 10 entries and currently is dormant. There is talk about resurrecting 7350 or creating another standard, but little has happened yet. Knowing ISO's snail-like pace, I don't think we can wait for anything to happen here. Also note that an ISO registry probably would be limited to international or national standard sets — that is, vendor-specific sets wouldn't get in.

(b) *X/Open.* X/Open is working on a registry of locale names and contents, and part of that involves defining string and integer values for registered locale names. These values will represent the three major parts of a locale — language, territory, and code set. Although the X/Open proposal is still a draft, integer values currently are defined as unsigned32s. The two most significant octets contain the registration authority, which can be anything from ISO to JIS to a company- or consortia-specific value. The lower two octets contain a number that represents a specific locale. X/Open will assign the integer values.

To provide code set interoperability, OSF only needs the code set portion of a locale name, and we have been asked to register (or provide a naming syntax for) only that single part. X/Open, on the other hand, is creating a registry of locale names in such a way that the code set portion cannot be picked out of a full integer value. Also note that X/Open is not registering character set values. If OSF and X/Open proceed with their registries, the possibility exists for conflict between the two systems. To minimize this, I recommend that OSF also use the upper two octets of a code set value to contain the registration authority, and that OSF and X/Open work together to define those values. I believe we can reach agreement on this.

A question still remains about the lower two octets. Should OSF simply register code sets, since that's all DCE needs, or should it register locale names? This is an open issue.

(c) *X Consortium.* The consortium has a small registry of code set names, but they are strings and (to my knowledge) do not include encoding method implementations like AJEC and SJIS. In addition, X does not register character set names (as the term is defined in this paper).

A code/char set registry is one of those ideas that comes up again and again in I18N circles, but that never seems to get done. I recommend that we work on a more complete registry proposal, reserving the right to terminate this project if we discover that the workload exceeds OSF's resources. The initial design goal will be a simple, no-frills system.

7. EFFECT ON OSF TECHNOLOGIES

If we register code and char sets, OSF technologies will be effected in various ways.

7.1. DCE

OSF will need to add code set and char set attributes to NSI for servers. Applications can then use those attributes to determine whether they want to connect to a given server. Applications are not required to make this evaluation — if they believe data loss is not an issue, they can ignore the character set attribute. In any case, they must check the code set so they can determine whether to send the data as is or convert it to another encoding. The other encoding may be the same as what the server is using, or may be an intermediate form like ISO 10646 or other region-specific encoding.

Because there are few existing tagged systems, clients and servers will typically rely on the current locale to identify the source's encoding and the encoding a server can handle. Therefore, there must be routines for converting code set names into and out of the registered values. Here's the basic idea:

```
<dce_stuff>_string_to_token (unsigned char codeset_string,
                             unsigned32 *codeset_int);

<dce_stuff>_token_to_string (unsigned32 codeset_int,
                             unsigned char *codeset_string);
```

The first routine accepts a code set string value as input and returns the equivalent registered integer value. The second routine does the reverse. OSF supplies the integer values, but each vendor must supply the mapping table from its local names to the OSF values. This allows vendors to continue to use local variations of a name — e.g., ISO8859-1, 88591, Latin-1 — and have them all map to the single OSF-registered value.

7.2. DME

As with DCE, DME and DME applications can use the registered code and char set values to determine whether to make a client-server connection. DME will use DCE's communications infrastructure, so code and character set data need to be attributes in a given DCE binding handle before DME can use them.

A DME application's evaluation logic may differ from that of a DCE application's. While DCE generally is only concerned with the way data is encoded and so only needs code set information, a DME application may care about the entire locale. The evaluation might work like this:

```
if (locale1 == locale2)
    accept server;
    send data as is;
else if (lang_terr1 == lang_terr2) &&
       (charset(codeset1) == charset(codeset2))
    accept server;
    convert codeset1 data to universal or
        other accepted encoding;
```

```
        send data;
        convert from universal or other accepted
            encoding to codeset2;
    else /* (charset(codeset1) != charset(codeset2)) ||
            lang_terr1 != lang_terr2 */
        reject server;
```

In this example, the DME application uses full locale names to determine whether a client should connect to a given server. When the names match, the client accepts the server and sends data without conversion. If the language and territory fields match, but the encodings differ (e.g., ja_JP.SJIS and ja_JP.AJEC), the application uses the DCE conversion functionality. If the language and territory fields differ (e.g., fr_CA.ISO8859-1 and de_DE.ISO8859-1), the application rejects the server. This example assumes that the server is supposed to perform locale-specific operations and that it would not be acceptable to use a German locale instead of French. If an application has more flexibility in the type of processing it can accept, it can include less restrictive evaluation logic.

DCE 1.1's planned code set interoperability functionality obviously must be in place before DME applications can do the evaluation outlined above.

7.3. Motif

X specifies a different model for code set conversions than is proposed for DCE client-server operations. In X (and thus Motif), if code sets do not match, the source data is converted to Compound Text and then from CT to the destination encoding. X mandates that implementations support conversions to and from all X-registered code sets. Data loss is a possibility with this model, but since conversions tend to be visible — as with a cut-and-paste between windows — users at least are aware if loss occurs.

Since X defines the interchange protocol, Motif may not be able to use OSF-registered code set values. This needs more study.

7.4. OSF/1

Most OS operations take place locally instead of being distributed. Therefore, OSF/1 has little need to convert data automatically from one encoding to another, and so it probably will not use the code set values very much. An exception would be `iconv()`.

REFERENCES

[RFC 13.0] A. Thormodsen, ''DCE 1.1 Internationalization Requirements'', August 1992.

[RFC 23.0] R. Mackey, ''DCE 1.1 Internationalization Guide'' (to be published).

AUTHOR'S ADDRESS

Sandra Martin
Open Software Foundation
11 Cambridge Center
Cambridge, MA 02142
USA

Internet email: martin@osf.org
Telephone: +1-617-621-8707