# Code Extension in ASCII* (An ASA Tutorial)

## Editor's Note

*The following working document was produced by a Subcommittee of the American Standards Association Sectional Committee X3, Computers and Information Processing, in its efforts to develop a proposed American Standard. In order that the final version of the proposed American Standard reflect the largest public consensus, X3 has authorized publication of this document to elicit comment, criticism and general public reaction with the understanding that such working document is an intermediate result in the standardization process and is subject to change, modification or withdrawal in part or in whole. Comments should be addressed to the X3 Secretary, Business Equipment Manufacturers Association, 235 East 42 Street, New York, N. Y. 10017.—E. Lohse, CACM Assistant Editor for Information Interchange.*

The American Standard Code for Information Interchange (ASCII) contains a number of control characters associated with the principle of *code extension*, that is, with the representation of information which cannot be directly represented by means of the characters in the Code. The manner of use of these characters has not previously been completely described.

This paper presents a set of mutually consistent philosophies regarding code extension applications, and suggests a corollary set of doctrines for the application of the code extension characters. Distinctions are drawn between code extension and such other concepts as "graphic substitution" or "syntactic representation" which are often used to meet similar requirements.

Also covered are certain topics which are not truly concerned with code extension but which are often linked with it in discussion on code applications.

The material in this paper is equally applicable in principle to the (proposed) ISO international 7-bit code for information interchange.

## 1. Introduction

In the establishment of a general purpose code such as the American Standard Code for Information Interchange (ASCII), or its international counterpart, the ISO 7-bit code, a fundamental decision must be made as to the size of the code. In making such a decision there is usually a conscious effort to avoid the most obvious problems with a code which is either too large or too small. Should the number of characters included be too small, many individual users will find their needs not accommodated, and will be forced to adopt "parochial" codes for their applications. Should the number of characters be too large, many potential users will find the standard code disproportionately costly to implement, or untenably inefficient in transmission or storage, and will again be driven to the use of some other code. Thus, either extreme in code sizing will reduce the generality of application of the code, defeating the very purpose of standardization in this field.

The 7-bit size (128 characters) adopted for ASCII is thought to be near optimum at present with respect to the above considerations. Nevertheless, there will doubtlessly be numerous applications with requirements that are not accommodated by a code of this size, or at least not by the specific characters assigned within it. Still, it is hoped that many of these applications can be served by the use of the standard code augmented in some appropriate manner. Through such an approach, the user may be able to implement much of his system with standard hardware or software. More significantly, perhaps, he will thereby be able to retain compatibility with other systems for the interchange of that information which can adequately be directly represented by the standard code.

The concept of augmenting the standard code for such purposes may be spoken of in a generic way as "code extension," although later we will restrict our extensive use of this term to a particular class of approaches.

The codes with which we are concerned contain four characters whose definitions indicate their relationship to code extension. They are:

| | |
|---|---|
| SO | (Shift Out) |
| SI | (Shift In) |
| DLE | (Data Link Escape) |
| ESC | (Escape) |
| SS | (Start of Special Sequence) |

The use of these characters is not treated in detail in the code standards. Actually, the very nature of code extension inherently limits the degree to which standards for it may be constructed: it is a means of operating "beyond the standard." Nevertheless, there are several advantages to establishing a unifying general philosophy.

First, such a philosophy can prevent undesirable conflict between independently contrived applications of code extension. For example, a code extension procedure used by a data communication terminal device should be inherently free from any hazard of conflict with a code extension procedure used in a communications system which may be called upon to serve the terminal.

Second, the availability of such a general philosophy can provide guidance to system designers to facilitate the advance inclusion of general provisions for code extension operations in information handling equipment.

Subsequent sections of this paper describe such a unifying general philosophy. It is directed at the application of code extension to those portions of a system where the use of the standard code itself would ordinarily be appropriate; that is, in what is spoken of as "information interchange."

Naturally, there are other functions within many information

* ASA Document X 3.2/368, March 11, 1966.

interchange systems for which an extremely unusual usage of the standard code, or some entirely different representation of information (e.g., the points of a character matrix), may be entirely appropriate. Such functions are often thought of as being internal to some autonomous system component. Just as the code standard is not presumed to be appropriate for such functions, it is not presumed that the philosophy outlined below is appropriate for them.

The suggested procedures presented here for the use of the five special control characters for code extension should in no way be considered to deprecate the practice of using sequences of graphic characters to represent machine instructions, graphic characters not otherwise available, and so forth. Programming languages used in data processing, for example, are based upon such an approach.

It is hoped that the material in this paper will provide background to help a system designer choose among all the various specific approaches in the face of any particular requirement.

The American Standard Code for Information Interchange consists of two general categories of characters, *graphics* and *controls*. There are 32 controls, 95 graphics, and the character DEL (Delete) which in reality is neither. The 95 graphics include both upper and lower cases of the Latin (often called "Roman") alphabet, the Arabic numerals 0 to 9, a number of punctuation marks and special symbols, and SP (space), the "nonprinting graphic."

## 2. Graphic Set Extension: Use of SO (Shift Out) and SI(Shift In)

There are a number of applications which are not adequately accommodated by the graphic set of the standard code. The most prominent examples are those of systems in which special symbols are required by some scientific discipline or commercial usage (for example, meteorology), and those requiring the use of languages which cannot be directly represented by the Latin alphabet, such as Russian. Of course, these needs could often be met through *graphic substitution*: that is, by adopting for the system a code which differs from the standard code in that certain standard characters are replaced by the special ones which are required. The displaced standard characters are, however, naturally lost to use by such a system. However, it will often be desirable for the system to have the capability of printing (or otherwise handling) such special graphics, while retaining the ability to communicate with other systems using the standard set of graphics. A suggested procedure is to use the characters SO (Shift Out) and SI (Shift In) to select which one of two sets of graphics is to be associated with the 95 "graphic" positions of the code. SO indicates that an alternate graphic set, containing the necessary special symbols (and perhaps some standard graphics as well) is to be put into force. SI brings about a return to the standard set of graphics.

The set of 32 control characters in the code and the character DEL (Delete) should not be affected by the shift operation. It is of course possible that the use of a new set of graphics may require a corollary change in the execution of a control within its standard definition. For example, if the alternate graphic set contains characters which are given a larger typographical size than those of the standard set, the character Line Feed may have to produce a larger motion when the alternate set is in use. This is of course not construed as a change in the control character set.

There is no implication that the alternate set should be entirely different from the standard set. It may contain whatever repertoire of characters are needed for operation in a particular environment. For example, the alternate set might retain the standard letters and numerals but replace certain punctuation marks with weather symbols. In another application, the lower case of the alphabet might be replaced in the alternate set by special mathematical symbols, while the upper case alphabet, the numerals, and the punctuation marks are retained. It is recommended that any symbols common to both the standard and alternate sets be assigned to the same code table position in both. It is also advisable to leave SP (space) in the alternate set whether required or not, as many printing mechanisms treat it separately, not actually behaving as if it were a graphic.

*Application to Devices of Modest Repertoire.* It should be noted that useful application of these principles may in some cases be made in devices having a relatively modest capacity for different graphics. Consider, as an example, the problem of making a terminal device to render messages in both Latin (standard) and Greek (alternate) alphabets, and requiring the conventional numerals and punctuation in connection with either. In many situations it would be satisfactory to render all letters in the upper case: that is, the receipt of the coded representation for either "A" or "a" would cause "A" to be printed.[1] Extending this principle to the special symbols coded in the same area of the code with the letters, one can see that 32 printing characters can suffice for 64 characters of the code. (Actually 63, since DEL, though coded in the graphic region, is not a graphic.) Adding provision for the 10 numerals and the 22 remaining symbols, the machine need have but a 64-character graphic capacity for its work in the Latin alphabet.

An additional 31 printing characters can serve, in the same manner, for both the upper and lower case Greek alphabets and some associated special symbols when in the alternate set. The 10 standard numerals and the 22 standard punctuation marks are used in the alternate set operation. This postulated application can therefore be implemented in this manner with a terminal device having only the 95-character graphic capacity which would ordinarily be required for full rendition of the standard set.

Such a device when in its standard mode may receive, without hazard, information containing any of the 95 ASCII graphics. If a graphic set shift were not used in this application, the bilingual capability could only be served with a 95-graphic printer by making the Greek alphabet a *graphic substitution* for the l.c. Latin letters in the code table. The device could not then be safely used for interchange of information with systems which might use the lower case Latin letters, since the receipt of these would of course cause the printing of Greek letters.

*Multiple Graphic Sets.* In many applications there will be a need for many alternative graphic sets. Applications in the graphic arts industry will often be of this class. It has been frequently suggested that, to cater to such needs, provisions should be made for the use of a suffix after the character SO to indicate which alternate set is desired. Actually, however, such a procedure appears to be neither necessary nor desirable. It is extremely attractive to reserve SO for use, by itself, to select the single alternate set in systems having but two sets,[2] and for selecting the principal alternate set in systems having several sets. The additional alternate sets, if provided, should preferably be invoked according to the following philosophy:

If there are several alternate sets to be invoked, there is, in effect, a need for the control functions SO, SO′, SO″, . . . , to put them into force. Of these, only SO is directly represented in the code, so some other means must be found to represent SO′, SO″, etc. There is, however, a general method of representing additional control functions through the use of sequences prefixed by ESC (Escape) (see the next section). Therefore, it is recommended that the additional shift controls required in a multiple-set system should be represented using Escape. This approach avoids any possible need for one device to be capable of handling two types of control-representing sequences, one type prefixed with Escape and the other with SO.

Regardless of the set currently in use, the character SI puts the standard set into effect.

---

[1] This technique is already widely in use where 64-graphic printers are used in systems which utilize all 95 graphic characters.

[2] Such systems may be of appreciable commercial significance. A prominent example is that of a message handling system in a country having a non-Latin national alphabet, where the national and standard (and therefore international) alphabets are both useful.

In some multiple-set applications there may be a need to repeatedly transfer back and forth between the standard set and a particular alternate set. The following approach could be useful in such situations, and is consistent with the philosophy already described:

The specific alternate set to be used is nominated as *"the* alternate" through the use of an associated control function (represented as an Escape sequence). SO then causes the nominated set to be put in effect in lieu of the standard one. SI, as always, restores the standard set, but does not effect the status of the nominated alternate set, which can be invoked again with SO when required. Nomination of one of the other sets as the one to be invoked by SO is done with another associated Escape sequence.

*Automatic Restoration.* It is recommended that terminal devices and other such equipment be arranged to automatically revert to the use of the standard graphic set whenever the association of the terminal with another terminal or system has been discontinued or suspended: that is, at the beginning or end of a *call, transaction, transmission,* or whatever is appropriate.

### 3. Control Set Extension: Use of ESC (Escape)

The expected requirements for additional controls beyond those assigned in the code are somewhat different from those for additional graphics. It is typical of systems requiring additional graphics, that the graphics may often be used in groups and for an extended period, such as when printing text in a foreign language. On the other hand, it is more typical controls that they appear sparsely throughout the information. For this reason, the recommended doctrine of obtaining additional controls does not provide for replacing the standard set of control characters with an alternative one, but rather for the one-at-a-time representation of additional controls by sequences of existing characters. These sequences are called "code extension sequences," and the term code extension is most directly applied to such representation of additional controls.

In order that a code extension sequence may invariably be identifiable as such, each such sequence begins with the prefix character ESC (Escape), which has no other use. (The name Escape is perhaps a little misleading in this respect: the character was initially established as a signal that subsequent operation was to be "not in the standard code.") Hence, these code extension sequences are also called "Escape sequences."

It was at one time proposed that code extension sequences should be standardized as consisting of ESC and a single-following character. While this would be adequate for many applications, there are a number of considerations which may make longer sequences desirable in many cases. One such consideration is just that of having an adequate number of sequences available for the functions required in one system, or in a number of systems requiring nonconflicting function representations. Another consideration is that it is sometimes desirable to represent a critical function by a long sequence to gain security against accidental or malicious operation. A third consideration is the desire, in some systems, to have a mnemonic relationship between the character sequence and the designation of the function to be controlled.

In many systems it may be appropriate to establish some fixed length for all Escape sequences. In other systems, it may be of great utility to have available a doctrine which allows sequences of various lengths to coexist in the same system. Paramount among the requirements for a variable-length doctrine is the need to have a simple means for a device to determine the end of each sequence which it receives: that is, how many of the characters following ESC are associated with it. This is necessary so that the device may avoid giving the normal interpretation to individual characters of a code extension sequence, even when the specific sequence is not to be recognized and acted upon.

The following approach is suggested for the assignment of sequences for those systems in which it is desired to use sequences of varying lengths:

The characters of the code are divided into two groups, called "intermediate" and "final" characters, respectively. (The basis for making a particular partition into these two groups is discussed later.) Any Escape sequence begins with ESC, continues with any number of "intermediate" characters (including none), and ends with one "final" character. Thus, any device recognizes an ESC as indicating that the characters which follow form a code extension sequence up through the next "final" character.

Thus, if we represent ESC by "E," any intermediate character by "I," and any final character by "F," the allowable sequences are of the forms EF, EIF, EIIF, EI, . . ., IF, etc.

*Partition of the Code.* There are a number of criteria affecting the way in which the characters of the code will be divided into "intermediate" and "final" groups. Among the significant ones are:

1. "Intermediates" should be distinguishable from "finals" by a simple logical test, preferably by the sense of 1 bit in the coded representation.
2. A given class of character, such as alphabetic, numeric, etc. should be entirely within one group.
3. Upper- and lower case of any specific alphabetic character should be in the same group. This allows a system designer to assign sequences so that no distinction is made on the basis of case, if desired.
4. A reasonable number of 2-character (i.e., ESC plus one "final") sequences should be available which use only letters or numerals, because such sequences are convenient for use by humans.
5. The "final" group should contain some characters which are likely to occur with reasonable frequency in a stream of data. This ensures that, should the legitimate final character of a sequence be lost or mutilated, the system will soon be restored to its normal mode of character interpretation.

Two specific partitions have recently been given serious consideration. The first meets most of the criteria, although it is not as easy to implement as the other. In this partition, columns 2 and 3 of the code table (numerals, punctuation, and special symbols) are "intermediate" characters; all others are finals. Thus, a 2-bit logical test is required to identify the class of a character. Nevertheless, this partition allows a number of attractive features:

1. A reasonable number of 2-character sequences can be made using only graphics. Even if the "special symbols" are not used, there are 26 sequences which can be formed if no case distinction is desired, or 52 if the two cases are treated separately.
2. There are 260 (or 520, again depending on the use of case distinction) 3-character sequences using only alphabetic and numeric characters.
3. Two-character sequences may be created with controls; this may be attractive where an alternative version of a standard control function is involved. Thus, ESC LF could be a sequence representing "half-line feed."
4. Longer sequences may be created with a control at the end and including numeric indices. Thus, ESC 3 5 DC2 could mean "switch device 35 to state 2."

On the other hand, in some applications alphabetic abbreviations for functions and devices are desirable, this partition does not allow a sequence to contain multiple alphabetics.

Partitions based on a single-bit difference between the two classes will probably be easier to implement in many devices. The leading candidate for such an "equipartition" is one which makes the left half of the code table (controls, numerics, punctuations and special symbols) "final" characters, and the right-hand half "intermediates." This arrangement reverses the role of alphabetic and numeric characters with respect to the features previously described, thereby somewhat diminishing the utility of the scheme.

The opposite arrangement has also been proposed. This retains the desirable relationship between alphabetics and numerics, but precludes the possibility of 2-character sequences using controls.

Further investigation of both implementation and application ramifications of these various partitions will be required before it will be possible to recommend one as being preferred.

*Restrictions.* There are some additional restrictions which are recommended in order to avoid certain potentially serious problems.

The ten communication control characters should never be used in Escape sequences. Such use could cause interference with the control logic of communication systems through which the data may be passed, unless the systems were arranged to detect the sequences and determine their lengths, an unnecessary burden. These ten characters are:

| *Designation* | *Name* | *Code Table Column/Row* |
|---|---|---|
| SOH | Start of Heading | 0/1 |
| STX | Start of Text | 0/2 |
| ETX | End of Text | 0/3 |
| EOT | End of Transmission | 0/4 |
| ENQ | Enquiry | 0/5 |
| ACK | Acknowledge | 0/6 |
| DLE | Data Link Escape | 1/0 |
| NAK | Negative Acknowledge | 1/5 |
| SYN | Synchronous Idle | 1/6 |
| ETB | End of Transmission Block | 1/7 |

Also, additional communication controls should preferably not be represented by ESC sequences, but rather by DLE sequences, as described in the next section.

The use of control characters in ESC sequences should be avoided whenever possible, unless the control function being represented is related to the control used in the sequence. For example, to use a sequence of the form ESC BEL to control a second audible signal seems unlikely to produce any difficulty.

The character DEL (Delete) should not intentionally be assigned in ESC sequences, as this character may appear unpredictably in some systems as a result of correction of operator errors in perforated tape. ESC itself should also not be intentionally included, especially since in some attractive partitions it is a "final" character; thus it would mean at once that the sequence was starting *and* ending, a paradox to be avoided. (Those partitions having controls as "intermediates" avoid this problem entirely, though that is probably not a compelling advantage.)

*"Locking" and "Nonlocking" Controls.* In many previous papers and discussions about code extension there has been an unnatural concern with the difference between "nonlocking" and "locking" Escape sequences; that is, between those whose function is of a temporary nature—"ring auxiliary bell"—and those whose function is persistent, such as "print red." There is no need for this concern as long as the control function is similar in nature to those directly represented in the code, such things as format effectors, information separators, device controls, and such. These existing controls also include both "nonlocking" and "locking" functions, a fact which has (rightly) caused little concern. Note that the suggested ways of utilizing ESC do not carry the concept that ESC by itself is a way of switching the device into a different mode for any extended period of time. The device is only in a different mode to the extent that those characters intimately associated with the ESC are given a new interpretation.

There is another concept associated with this groundless concern which is inappropriate from a more serious viewpoint. Often people speak of the use of an Escape sequence (or of Escape itself) to "put the system into another code." This concept is an abstraction which cannot be discussed unless more specific details are ascribed to it. Suffice it to say that no general provisions for such behavior can be standardized in any meaningful way. A similar problem exists with respect to certain concepts of so-called "transparency"; this matter is discussed in the section of that name.

It has been recommended that terminal devices clear themselves of any "abnormal" state when their association with another terminal or system has been discontinued or suspended. This is, of course, equally desirable whether the "abnormal" state is achieved by local control, by the receipt of a control character, or by the receipt of an Escape sequence. In addition, should such association be discontinued or suspended during an ESC sequence, the sequence should be considered aborted.

*Use of Escape for Additional Graphics.* The question is often raised as to whether it would be appropriate to use ESC to create additional graphics. There are two ways in which this might be done. One way is the use of ESC sequences to represent additional "shift" controls, as discussed above under *Graphic Set Extension*; these additional shifts indeed create representation of new graphics, but the ESC sequence has merely been used to represent additional controls, an orthodox procedure.

The other possible procedure would be to use ESC sequences to represent individual additional graphics. While there appears to be no hazard associated with such a usage, it should be kept in mind that such an operation may be extremely cumbersome to implement in many devices.

*Standardization of Escape Sequences.* It has often been proposed that provision be made for standardizing specific sequences to represent control functions which have become widely enough used to justify such action. Although this is an attractive notion, there are a number of obstacles to its implementation. Prominent among these is the need to reserve a "block" of sequences for use by the standardizing body as the need arises. Any such "reservation" would appear to intrude upon the individual user's ability to optimally solve his own problems. Attempts to devise a procedure to allocate blocks of sequences based upon the final character, or upon the character immediately after Escape, have produced unattractive results.

A recent proposal is to reserve for future standardization all sequences of some specified length or lengths. For example, it might be established that any standardization of sequences for specific functions would be done with 3-character sequences (E-I-F structure). Thus users could use any 2-character sequence for special purposes without fear of later conflict. Such short sequences are attractive for functions which, though specialized, are used frequently in the systems requiring them. Users desiring long sequences may also assign them without worry.

Adoption of such a concept could perhaps lay the groundwork for standardization of sequences at some future time.

*Printing of ESC.* The question is often raised as to whether or not typical terminal devices will print a symbol for ESC. It is expected that there will be printing devices capable of printing symbols for many or all of the control characters. Nevertheless, it is a basic concept of the code that, in the ordinary application, the control characters will be nonprinting. Control-printing devices would thus be primarily for monitoring, maintenance, and similar functions.

When a need is expressed for ESC to print, it is instructive to ask, "if there were a character in the code for this function, would we want *it* to print?" If the answer is "no," then it is reasonable to suggest that the ESC (and the rest of the sequence, for that matter) not print. The nonprinting of ESC will no doubt be inherent on most devices as it is a control character; nonprinting of the rest of the Escape sequence may be easily controlled by virtue of the simple way in which beginning and end of a sequence are delineated.

If the answer is "yes," then either:

1. The application calls for a "monitoring type" printer anyway, and when one is provided the ESC *will* print; or
2. The function is in the nature of a program instruction, or

an abbreviation for something, and should be represented by the syntactic use of graphics anyway, as is traditional in programming languages.

As usual, the establishment of clear relationships between the purpose of Escape sequences and that of directly represented control characters will generally make the answer to such dilemmas obvious.

### 4. Code Extension for Communication Controls: Use of DLE (Data Link Escape)

Recommendation of any specific doctrine for the use of DLE falls within the jurisdiction of ASA Task Group X3.3.4, Communication Control Procedures. Their recommendations to date are contained in a number of tutorial papers[3] issued by that group. The subject is discussed in this paper only to show the relationships of this use to other aspects of code extension.

It is necessary for a communication system to be able to readily distinguish between the communication controls which are of concern to it and other controls with which it is not concerned. The assignment of specific communication control characters in the code provides this distinction under ordinary circumstances. It is necessary that this distinction be preserved when additional controls of one type or the other are represented by code extension sequences. The character DLE (Data Link Escape) is provided for use in lieu of ESC as the first character of code extension sequences used to represent additional communication controls. Thus, communication link control logic may ignore ESC entirely, passing it and the characters which follow as any other "text" characters. Code extension sequences of concern to the communication control logic can invariably begin with DLE, to which the logic may be made sensitive.

The prohibition previously expressed against the use of communication control characters in ESC sequences is intended to prevent direct interference with the communication control logic.

### 5. Concept of "Special Sequences": Use of SS (Start of Special Sequence)

The concepts previously described involve changing the "meaning" of characters in the code. That is, the coded representation which ordinarily denotes "plus" may denote "radical sign" when preceded by SO; the representation for "Z" may denote "print red" when prefixed by ESC.

There is another related type of special behavior which may be usefully distinguished from the above-mentioned types. In this class the *meaning* of the characters is, in effect, not to be changed but merely the action which is to be taken upon their receipt. (Keep in mind that the code standard does not prescribe the action to be taken upon the receipt of a character. This is a function of the application of the code in a particular device.)

A homely analogy may be drawn from the world of the theater. The word "enter" as part of a stage direction calls for a different action on the part of the actor than the same word in the lines to be spoken; yet, the letters e-n-t-e-r do not really change meaning.

In the information processing field there are numerous similar examples. Commands to the time-sharing executive program of a computation system need to be distinguished from program or data statements. Device controls used to represent functions such as "punch off" often need to be flagged as not applicable to the peripheral device through which they pass at a certain time—otherwise one would perhaps be unable to cause an output punch to punch "punch off". (This is equivalent to the bypass/restore concept often used in perforated tape systems.)

The character SS (Start of Special Sequence) is recommended for use to indicate the beginning of a collection of data requiring some such special handling. Due to the extremely diverse nature of the possible applications, it is impossible at this time to propose a more detailed standard doctrine for such applications. In particular, the extent of the data over which the connotation of special handling extends (the length of the "special sequence") is not prescribed, nor is any particular character to mark the end of such a sequence. Appropriate arrangements must be made in the face of the requirements of a particular application.

### 6. Transparency and Its Relationship to Code Extension, If Any

When topics such as those above are discussed there almost invariably arises some reference to the concept of *transparency*. This term broadly refers to those properties of an information handling system which allow it to handle data which it could not handle if it did not have so much transparency. For example, if one has a communications system which will only pass 118 of the ASCII characters (because the other ten are reserved for internal use by the system), and one wishes to send text including some of the forbidden characters, it is said that (more) transparency is needed. If one has a paper tape system which can only handle 7 bits of information per frame (perhaps because the eighth track is committed to parity), and one wishes to record 8 bits per frame, again it is said that (more) transparency is needed. Obviously there are as many different types of transparency as there are parameters of an information handling system.

Many workers in this field have suggested that any doctrine for code extension should include "provision for transparency." The concept of transparency which is inherent in this suggestion is an abstraction which is not susceptible to specific treatment. Only when specific problem components are identified and described can solutions be synthesized, and the part played in such solutions by code extension may then be determined.

In general, the relationship between "transparency" and code extension is no more nor less than this: If there is a need in an application for some specific type of "transparency" it must be built into the system involved. If additional control functions are needed to manage the application of transparency, or to control the system in the face of its new-gotten transparency, perhaps these can be obtained through the use of code extension.

An example of this may be found in the doctrine proposed by Task Group X3.3.4[4] for the control of communication systems possessing a certain type of transparency in order to meet certain specific objectives.

In this doctrine, DLE sequences are used to represent controls to invoke the particular transparent properties which the system may have, and to represent the additional control functions required to unambiguously control the system while it has these properties.

### 6. Conclusion

It is felt that the philosophies and recommended doctrines described herein form a useful and mutually consistent structure for handling a number of related problems. It is not immediately apparent in what way standards may be written to facilitate the consistent application of these concepts. It is hoped that the publication of this paper will elicit comments on the relationship between these proposals and known applications that will give guidance in that respect.

---

[3] A summary will be found in: *Comm. ACM 9* (Feb. 1966), 100 *et. seq*; see especially Section 6.

[4] *Comm. ACM 9* (Feb. 1966), 100 *et seq*, Section 7.
*Comm. ACM 8* (April 1965), 203 *et seq*.