X11/1998-27

MUMPS Development Committee

Extension to the MDC Standard
Type A Release of the MUMPS Development Committee

Pattern Match String Extraction, v. 3

June 28, 1998

Produced by the MDC Subcommittee #13
Data Management and Manipulation

Art Smith, Chairman
MUMPS Development Committee

Dan Bormann, Chairman
Subcommittee #13

1. Identification of the Proposed Change

1.1 Title

Pattern Match String Extraction, v. 3

1.2 MDC Proposer and Sponsor

Arthur B. Smith, Emergent Technologies
A353 Clydesdale Hall
379 East Campus Drive
Columbia, MO 65251 USA
Voice: (573) 882-2666 (weekdays)
Voice: (573) 642-8802 (weekends/evenings)
FAX: (573) 882-5444
e-mail: SmithAB@missouri.edu *–or-* Emergent@sockets.net

Subcommittee 13 Task Group 2
String Handling
David Whitten, Chair

1.3 Motion

This document (as X11/SC13/1998-8) was approved as MDC Type A at the June 1998 meeting in Boston, MA.

1.4 History

August, 1998	X11/1998-27	<This document> Final form (updated history, motion and discussion only).
June, 1998	X11/SC13/1998-8	Proposed for MDC Type A status. History & discussion updated. Passed 11:1:7
Mar. 1998	X11/SC13/TG2/98-1	Proposal modified to use Longest Match disambiguating algorithm per SC recommendation. Proposed for replacement Type B status, motion amended in SC to raise to SC Type A status (amendment: 13:1). Passed 13:2:4
Sep. 1997	<no document>	Subcommittee requested another version of this proposal, based on results of a straw poll proposed by the Task Group favoring a "Longest Match" disambiguating algorithm.
Mar. 1997	X11/SC13/97-2	Proposed for elevation to SC-A, details unchanged from previous version, though language strengthened in 2.1. Failed 4:3:3
Oct. 1996	X11/SC13/96-11	Proposed for replacement SC-B due to changes in formalism reflecting removal of disambiguating rules. Passed 16:7:3.
Mar. 1996	X11/SC13/TG2/96-3	Replacement SC-B due to changes in formalism (extraction into setdests rather than glvns). Passed 16:0:4

Oct. 1995	X11/SC13/TG2/95-6	Proposed for SC-A, defeated in Task Group 0:6:4 due to imprecision in the formalism.
Jun. 1995	X11/SC13/TG2/WG3/95-1	Pattern Match String Extraction v. 2 with algorithm separated out as replacement SC-B (passed 18:1:4).
Jan. 1995	X11/SC13/TG2/WG3/94-2	Pattern Match String Extraction as SC-B (12:3:3)
Jun. 1994	X11/SC13/TG2/WG1/94-1	A Comparison of Regular Expressions and Pattern Match in MUMPS, discussed in Working and Task Groups
Jul. 1993	X11/SC13/TG2/WG1/93-1	Regular Expressions discussion paper

1.5 Dependencies

None.

2. Justification of the Proposed Change

2.1 Needs

Many applications accept structured input that must be parsed after entry. Examples include peoples' full names, account numbers and mathematical expressions. Pattern match has long been used in MUMPS to validate the format of these structured inputs. With the current pattern match operator, however, input which matches the pattern must then be reparsed to obtain the component pieces since there is no way to obtain the substrings corresponding to the pattern components. This requires extra processing and introduces potential errors due to divergence of the pattern match and the parsing code.

It would be beneficial if the pattern match operator could perform the extraction of substrings in the event of a successful match. This closely parallels the ability to do string extraction using regular expressions in Unix utilities. With the incorporation of multiple patatoms in pattern match (currently an MDC Type A proposal), pattern match has become equivalent in power to regular expressions.

Historically, the MUMPS pattern match operator is only able to produce the True/False answer to the question "Did the string match the pattern?" All that was necessary to answer this was to find if at least one partitioning of the string into substrings matched a corresponding partitioning of the pattern into pattern elements.

It is often the case that more than one such partitioning is valid, which has not been the concern of the pattern match operator until one tries to match individual substrings with the component patatoms in the pattern. This version of the proposal reflects a long history of attempts to produce a set of disambiguating rules that always produces only one valid partitioning. See Section 7 (Discussion) for details. A brief history follows:

Several attempts were made to establish a set of disambiguating rules so that the actual partitioning used can be determined. It was hoped that these rules could be constructed so that they simply echoed the way all of the M implementors already ran their pattern match. All of these attempts met with difficulties, and cons such as "Ties hands of the vendors" were frequently cited. Vendor feedback was scarce and, when offered, pointed out problems without offering suggestions for solutions.

It also appears that most, if not all, "real world" uses of pattern match string extraction do not require ambiguous patterns. The ambiguities appear to arise only in easily avoided pathological cases that are of little practical use.

For this reason, it was determined that the best approach was to simply allow the ambiguity. The partitioning of the string into substrings would, in some cases, be nondeterministic, and users were guaranteed only that one of the possible parses will be consistently represented in the assignment of variables by pattern match string extraction.

This ambiguity proved too unpalatable for the Subcommittee, however, and so the proposal failed to move forward. A subsequent Straw Poll in the Subcommittee indicated strong support for a Longest Match algorithm, which is presented in this document.

2.2 Existing Practice in the Area of the Proposed Change

Current practice involves the use of the pattern match operator to determine if input is in an acceptable format, followed by separate application of string manipulation functions (e.g., $PIECE and $EXTRACT) to parse valid input into component substrings.

3. Description of the Proposed Change

3.1 General Description of the Proposed Change

Pattern match compares a string to a pattern to determine a yes/no answer to the question "does the string match the pattern?" The purpose of this proposal is to extend pattern match to allow some or all of the substrings matching the individual patatoms to be extracted into variables as a side effect of a successful pattern match. This general functionality is described in the literature variously as string decomposition, parsing, tokenization or lexical analysis.

A pattern is made up of a sequence of patatoms. If a string successfully matches this pattern, each patatom represents a (not necessarily unique) set of substrings, and the concatenation of these substrings is the string itself.

It is significant that each patatom represents a *set* of substrings. A pattern containing alternation may have patgrps containing patatoms within the alternation which are used more than once to match different substrings (if the alternation repcount is greater than one). Similarly, there may be patatoms within the alternation which do not match any part of the string, yet the pattern as a whole matches the string as a whole.

The proffered syntax for this functionality allows the programmer to place a setdestination in parentheses immediately following the patatom as currently defined in pattern match. This is not ambiguous with alternation since alternation is *always* preceded by a repcount and the setdestination can *never* be immediately preceded by a repcount, nor can it start with something resembling a repcount.

The following recursive rules are used to effect the string decomposition. Implementations are required to perform *as if* these rules were followed as specified, but may use any mechanism to achieve this result.

1. The pattern match is evaluated precisely as before to determine success or failure. The setdestinations, if any, are not evaluated at this time, i.e., indirection is not expanded, subscripts are not checked and the naked indicator is not changed.

2. If the pattern match fails, nothing further happens. The setdestinations remain unevaluated, the variables they denote are unchanged, and the naked indicator is unchanged.

3. A correspondence between substrings and patatoms consistent with the successful match is established. When more than one correspondence is possible, apply the following recursive rule set to each patatom from left to right to determine the specific correspondence to use. Note that each patgrp within an alternation requires a separate application of these rules.

   A. If the patatom is not an alternation, select the longest matching substring that produces a match in the pattern as a whole.
   B. If the patatom is an alternation, use the below rules and apply rules A and B recursively to each patatom in the selected patgrp(s) from left to right.
      1) Select the correspondence(s) that uses the smallest possible value of the alternation's repcount.
      2) If multiple correspondences satisfy 1), for each sequential application of the alternation (i.e., each value of the repcount) select the patgrp(s) within the alternation that correspond to the longest possible substring.
      3) If multiple correspondences satisfy 1) and 2), select the leftmost patgrp in the alternation.

4. Consider each patatom which has a setdestination associated with it, in order from left to right. Assign the corresponding substring(s), if any, sequentially (in the order they occur in the string) to the setdestination exactly in the manner of the SET command. Note that this is when the setdestinations are evaluated for subscript or indirection errors, and when the naked indicator may be modified. As previously noted, there may be no substrings to assign (in which case the setdestination is not evaluated at all), or there may be multiple substrings to assign. Also note that the patatom includes the repcount so if multiple repetitions of the patcode, strlit or alternation are required, the concatenation of all the substrings consumed by the repetitions is assigned, as that is the substring corresponding to the entire patatom.

Implications of these rules include:

A. Errors in the setdestination interrupt the assignments at that point. Any setdestinations to the left of the setdestination with the error that have corresponding non-empty substrings have been assigned, those to the right have not.

B. Not all setdestinations are necessarily evaluated; setdestinations not used (due to alternation) or following an erroneous setdestination are unevaluated and unchanged.

C. References to gvns in exprs in arguments or subscripts of setlefts or references to gvns in the setdestination may affect the value of the naked indicator. In particular the naked indicator is set to the last successfully referenced gvn.

D. Evaluation of the setdestinations may be dependent on the parse of patcodes evaluated earlier in the pattern. For example, in the expression

```
"12.ABC"?1.N(x)1P.E(y(x))
```

the value 12 would be assigned to variable x, and then the value "ABC" would be assigned to y(12). Note that if one assumes this functionality is desirable (and there are conceivable uses for it), then the late evaluation of the setdestinations is necessary. In the above example if the variable x was previous undefined, an early evaluation would produce an "Undefined" error (ecode=M6).

E. The same substring, or overlapping substrings may be assigned to multiple setdestinations. Consider the following example:

```
"A1"?2(1A(x),1N(y))(z)
```

In this example, x is assigned "A", y is assigned "1" and z is assigned "A1", as these are the substrings that correspond to the patatoms associated with these variables.

F. Variable repcounts must be carefully considered. Consider the five following examples:

```
(1)      "AB"?1.3A(x)
(2)      "AB"?1.3(1A(x),1N(y))
(3)      "AB"?1.3(1A,1N)(x)
(4)      "AB"?1.3(1.3A(x),1.3N(y))
(5)      "AB"?1.3(1A(x),1N(y))(z(x))
```

In the first example, the variable x is assigned "AB", as that is the (only) substring that satisfies this patatom and leads to a successful match of the pattern as a whole.

In the second example, the variable x is first assigned "A", and then assigned "B" so that it ends with the value "B".

In the third example, the variable x is assigned "AB". While the pattern and string are identical to the second example, the assignments differ because the patatom with the setdestination is now the alternation (with the multiple repcount), not the (single repcount) component patatom.

In the fourth example, x is assigned "AB" (behaving like the first example). Note that this is a case where the disambiguating rules apply. By minimizing the alternation to a single repetition (rule B1) we get this result. If the alternation's repcount were allowed to assume a value of two variable x might be assigned the value "A" and then overwritten with "B" (behaving like the second example).

In the fifth example, the variable x is assigned "A", and then "B" as in the second example. Following this z(x) (the setdestination for the alternation) is assigned "AB". Thus z("B") will have the value "AB".

As a further demonstration of the application of the disambiguating rules, consider the additional two examples:

```
(6)      "<ABCD>"?1P1.3(1.3A(x),2E(y))(z)1P
(7)      "<ABCD>"?1P3.(.2A(x),2P(y))(z)1P
```

In example (6), the alternation's repcount is first minimized to 2. In the first repetition, the longest substring is consumed, so x gets assigned "ABC". In the second repetition of the alternation, x is overwritten with "D" and finally "z" gets assigned "ABCD". Variable y is not assigned.

In example (7) the alternation's repcount is first minimized to 3. In the first repetition the longest substring is consumed so x gets assigned "AB". In the second repetition, the longest substring is consumed so x gets overwritten with "CD". In the third repetition, the longest substring is consumed so x gets overwritten with the empty string. Finally, variable z gets assigned "ABCD". Variable y is not assigned.

3.2 Annotated Examples of Use

See section 3.1, above, for more examples.

```
ER READ "Enter <item no><comma><quantity ordered>: ",X,!
   IF '(X?4N(ITEM)1","1.3N(QUANT(ITEM))) DO  GOTO ER
   . WRITE "Bad input format. ",!
   WRITE "You have selected ",QUANT(ITEM), " of ",ITEM, ".",!
```

3.3 Formalization

All modifications are to ANSI/MDC X11.1-1995.
Modify the definition of patatom in section 7.2.3 to be:

```
                         | patcode        |
patatom  ::=  repcount   | strlit         |  [ patsetdest ]
                         | alternation    |
```

and add the following definition following the definition for alternation:

```
patsetdest  ::=  ( setdestination )
```

Also, add the following text:

If more than one one-to-one order-preserving correspondence between the S_i and the pattern atoms exist the following rules are used to select the correspondence used in the two paragraphs following the rules. These rules are applied to each patatom in the pattern, from left to right and recursively in the case of alternations.

A. If the patatom is not an alternation, select the longest matching substring that produces a match in the pattern as a whole.

B. If the patatom is an alternation, use the below rules and apply rules A and B recursively to each patatom in the selected patgrp(s) from left to right.
   1) Select the correspondence(s) that uses the smallest possible value of the alternation's repcount.
   2) If multiple correspondences satisfy 1), for each sequential application of the alternation (i.e., each value of the repcount) select the patgrp(s) within the alternation that correspond to the longest possible substring.
   3) If multiple correspondences satisfy 1) and 2), select the leftmost patgrp in the alternation.

Each optional patsetdest, if any, is executed only if S?pattern is true, and only if the associated pattern atom is satisfied by one of the S_i in the selected correspondence. If these conditions hold, these (and only these) patsetdests are executed from left to right as follows:

For each of the substrings S_i of S satisfying the pattern atom in the selected correspondence, in the order in which they (the S_i) appear in the string, perform all the actions of SET setdestination=S_i as defined in section 8.2.18.

4. Implementation Effects

4.1 Effect on Existing User Practices and Investments

Existing MUMPS code will be unaffected by this proposal. This proposal extends the capabilities of the language and simplifies the code necessary to parse regular languages.

4.2 Effect on Existing Vendor Practices and Investments

It is hoped that the rules specified herein correspond identically with the algorithms used by the vendors. Simply performing the extraction process will doubtless impose some significant impact on the vendors. Vendor input on the extent of this impact is heartily encouraged, as no quantitative answers have been obtained from any vendor.

Having to alter the pattern match algorithm would unnecessarily add to their effort and may significantly impact efficiency of pattern match evaluation. If the disambiguating rules proposed in Sections 3.1 and 3.3 do not correspond to the algorithms used by the M vendors, they are requested to propose a set of rules that do match.

Note to the M vendors: The Subcommittee has indicated that this is desirable functionality, and that leaving the ambiguous cases unspecified is unacceptable (see Section 7, especially September 1997). In light of this, vendor objections that the disambiguating rules do not match their algorithm without helpful suggestions for how to modify the rules to avoid that difficulty will no longer be welcomed as constructive input by the document editor.

Furthermore, requests for vendor input on the impact of this proposal have been in every iteration of this proposal from January, 1995 to the present. Every effort has been made to quantify the vendor impact of this proposal. In the opinion of the document editor, objections based on the inadequacy of this section are a reflection on the responsiveness of the vendors, rather than the strength or weakness of this proposal.

[Despite requests only one vendor was able to supply any information, and that only at the June 1998 meeting, not before. This vendor indicated that the proposal could not be implemented cost effectively. ABS 8/98]

4.3 Techniques and Costs for Compliance Verification

The examples shown in Sections 3.1 and 3.2 must produce the effects explained in the text.

4.4 Legal Considerations

None known.

5. Closely Related Standards Activities

5.1 Other X11 Proposals Under Consideration

None known.

5.2 Other Related Standards Efforts

None known.

5.3 Recommendations for Coordinating Liaison

None.

6. Associated Documents

X11/SC13/TG2/WG1/93-1	Regular Expressions. Discussion paper.
X11/SC13/TG2/WG1/94-1	A Comparison of Regular Expressions and Pattern Match in MUMPS. Discussion Paper.
X11/94-14	Multiple patatoms Within alternation. MDC Type A

7. Issues, Pros and Cons, and Discussion

October, 1993, Dublin

Task group discussion of discussion paper X11/SC13/TG2/WG1/93-1, Regular Expressions by Art Smith and Fred Hiltz.

February, 1994, Houston

Discussion of Regular Expressions in working group as a complementary technique to pattern match. Working Group report to Task Group generated some slight interest in investigating the option to enhance pattern matching to obtain the functionality of regular expressions.

June, 1994, Reno

The issue was raised that this invokes side effects for an operator, which might better be done by a function. The working group opted to pursue the operator for now. The issue of early versus late evaluation of the glvns was discussed. Some present were concerned that this was not in keeping with the rest of MUMPS, though it was pointed out that $SELECT has the same late evaluation (and possible non-evaluation). The late evaluation provides added functionality.

The disambiguating rules were recognized as arbitrary, and differ as presented herein from those originally selected by the working group due to vendor response. It is important to the success of this proposal that it not require extreme modification of vendors' pattern match engines.

Art Smith is to write a proposal for the next meeting.

E-mail correspondence, November 1994, with Maury Pepper

It was discovered that the case of overlapping substrings was not addressed in the original form of the proposal, and that the left-to-right order was not clearly identified as substring order as opposed to patatom order. Further, the need for $MATCH was shown, and the syntax developed (another proposal was to use an empty subscript to identify the sequencing).

Because of the pathological examples discussed and the questions of order of evaluation and side effects of assignment, it was considered that perhaps the glvns shouldn't be allowed to be subscripted at all, but it was felt that this significantly weakened the proposal. To quote from one of the messages "I think the 'real' applications for this will be intuitive. The ugliness comes mostly, if not entirely, from deliberately pathological examples. Certainly this is no worse than indirection...."

Maury Pepper co-authored document X11/SC13/TG2/WG3/94-2, a predecessor to this document in acknowledgment of his contributions.

January, 1995, Albuquerque

David Whitten proposed an alternative approach to that presented in X11/SC13/TG2/WG3/94-2, Pattern Match String Extraction. The alternative approach would use different syntax to avoid the side effects present in this proposal. The working group wanted to see both proposals to decide which was preferred.

Both authors wished to have their proposals considered on their individual merits, rather than on the choice of disambiguating rules. It was decided at that time to produce a document describing the disambiguating rules, which each proposal could then reference. This would guarantee that the choice of rules does not influence the choice of proposals, and would allow the rules to be modified without affecting either proposal.

Document X11/SC13/TG2/WG3/94-2, was discussed at the working group meeting. The $MATCH svn described in that document was eliminated before the proposal was brought to the Subcommittee for elevation to SC-B status. Pros and cons and **(number of times cited)** are shown below. The motion passed 12:3:3

PRO	CON
1. Extends existing functionality **(2)**	2. Operator side-effects **(4)**
2. Functionality has been desired and is implementable **(1)**	
3. Side-effects **(3)**	

June, 1995, Chicago

Brought forward as Subcommittee Type B. Four pros and two cons were identified. Numbers in **(boldface parentheses)** following the pro/con are number of times cited in the ballot. The motion passed 18:1:4.

PRO	CON
1. Needed functionality **(3)**	1. Missing impact statement **(2)**
2. Powerful and flexible **(2)**	2. Relies on side effects **(3)**

3. Locality of reference (3)
4. MUMPSish solution (2)

Regarding Con 1, the impact statement, Section 4, was not missing, but was, perhaps inadequate. It has been further quantified in this rendition. In particular, three vendors have been consulted repeatedly on this. All have promised to get back to the author with a response each time, and none have. In one case, a name was given of whom to address the question to; a letter to this individual failed to evoke any response. Further comments from vendors *continue* to be invited.

Regarding Con 2, the author acknowledges that this introduces side effects into an operator. No other MUMPS operators have side effects. Side effects should be very closely examined to determine if the value of their effect exceeds the damage they exact in clarity. Straw polls taken in task group and working group meetings have indicated that this is the preferred approach. Pro 3 speaks to the desirability of this approach.

Pattern Match Algorithm was also broght forward as a Subcommittee Type B document and passed 16:2:4 with the following pros and cons:

PRO	CON
1) Required for Pat. Match String Extr. (8)	1) Discourages innovation in algorithm improvement. (4)
2) Results are verifiable independent of implementation. (1)	2) Currently left to implementor. (1)
3) Makes Pat. Match String Extr. determinate. (2)	3) Side effects in operator. (3)
4) Good base for further standards development. (2)	

October 1995, New Orleans

Task Group 2, in discussing this document, noted that the phrase "in the manner of the SET command." in the formalism was confusing and imprecise. The Task Group recommended that the appropriate language from the SET command be used instead. The motion to bring it forward for SC-A status failed 0:6:4. There was one Pro (Needed functionality) and one Con ("In the manner of the SET command" too imprecise).

Pattern Match Algorithm was brought forward and passed 10:4:11 with the following pros and cons:

PRO	CON
1) Needed for pattern match string extraction (4)	1) Ties hands of the vendors(7)

March 1996, Boston

In addressing the concerns of the Task Group, it became apparent that the functionality could be increased while simplifying the formalism by changing the destination of the assignment from a glvn to a setdest, thereby making the action be exactly that of a SET statement. This adds the capability, for example, to have a single pattern match accept a phone number in any of a number of formats (but only those formats) and, if accepted, convert it to a canonical form (by using the equivalent of SET $E or SET $P). Because of the significance of this change, the proposal was brought forward for consideration as a replacement type B, rather than a type A at this time. This was seen as an improvement by the task group, and this proposal was accepted as a replacement type B by a vote of 16:0:4.

David Marcus of Micronetics discovered that the disambiguating rules in the related proposal X11/SC13/96-3 (Pattern Match Algorithm) had a problem that could lead to an infinite recursion (consider "123Z"?.(.P,1N)1"Z"). After examining this, it became clear that the proposal as written had flaws, and it was not brought out of task group. Upon further reflection, and with no suggested improvements from any of the vendors, the author has decided to eliminate the disambiguating rules entirely, thereby eliminating the need for the Pattern Match Algorithm proposal. Because of this change, the proposal is once again being brought forward as a replacement Type B proposal.

October 1996, Toronto

This version represented a substantial change from previous versions. The disambiguating rules which caused such problems were summarily dropped. This results in admittedly ambiguous results in some cases, though these are generally only artificially pathological examples. The motion to make this revision (as a replacement type B) passed 16:7:3, indicating a fairly strong mandate that this version is preferable to the previous version which had flawed disambiguating rules. The following PROs and CONs were cited (number in parentheses is number of citations).

PRO	CON
1) Desirable functionality (8)	1) Results may be inconsistent across implementations (12)
2) Exists elsewhere (4)	2) Results may not be predicted (9)
3) Removes ropes from vendor's hands (3)	3) Should be a library function (7)

The vote count itself is ambiguous as to whether this proposal is better because it removes the disambiguating rules, or whether it is better because it is not flawed by broken disambiguating rules. Is it "No rules are better than bad rules", or "No rules are better than any rules"? Certainly the number of times CONs 1 and 2 were cited indicate that the lack of disambiguating rules is of concern. I believe this concern is deserved, but not damning, as the inconsistent cases appear to occur only in fairly contrived examples, and the inconsistencies can readily be removed by rewriting the pattern. To date, no real-world patterns show the inconsistency, nor has any inconsistent pattern which cannot be rewritten without ambiguity been proposed.

Regarding CON 3, this has been previously addressed, most recently in straw polls during the June 1995 meeting. The argument of the first paragraph of section 2.1 (input parsing should exactly match input error checking, without allowing divergence) addresses this, I believe.

March 1997, San Diego

A version of the proposal with no disambiguating rules (unchanged from the previous version except for tightening the wording in Section 2.1) was brought forward in Task Group with the motion to present it to Subcommitte for elevation to SC-A. This motion failed in Task Group 4:3:3 with the following (TG) pros and cons:

PRO	CON
1) Powerful	1) Ambiguity
2) Necessary	2) Overloads pattern match
	3) Unnecessary.

When the document editor equested further guidance, it was noted that the proposal was powerful and needed but not in this way. The editor was requested to investigate possible problems in pattern indirection.

A motion was made from the floor (by Fred Hiltz, seconded by Art Smith) in Subcommittee 13 to accept the document as a Subcommittee Type A proposal. The motion failed 6:7:6 with the following pros and cons:

PRO	CON
1) Adds needed functionality (4)	1) Ambiguous results substring (10)
2) Something similar has been implemented (4)	2) Evaluation requires new mechanisms (8)

Regarding the concern about Pattern Indirection, the proposal editor does not see any difficulties. Pattern Indirection would not be changed in its definition (an expratom V pattern following the ? operator). Since the definition of pattern is expanded by this proposal, the range of allowable expratom values is increased, but no change appears necessary.

September 1997, Chicago

No document was brought forward at this meeting. Question was raised in Task Group 2 about the status of this proposal. The editor indicated that after the previous meeting's discouraging results he was letting it "cool off" for a meeting before starting it moving backwards towards recision. Task Group 2 then brought a straw poll before the Subcommittee to confirm their intention. The results of this poll are shown below:

"How should Pattern Match String Extraction handle ambiguous constructs?"

1) First match (some Unix tools do this)	(0)
2) Longest match (some Unix tools do this)	(13)
3) Shortest match (some Unix tools do this)	(0)
4) No rule (keep ambiguous)	(6)
5) Produce an error	(1)
6) Let Pattern Match String Extraction die	(0)

Given these results, the proposal editor will attempt, once again, to produce a version of Pattern Match String Extraction with disambiguating rules that appear acceptable to the members of Subcommittee 13. Constructive input from vendors, especially on improving the disambiguating rules and quantifying Section 4.2, is encouraged in the strongest possible terms.

March 1998, Atlanta

The "Longest Match" disambiguating rules were formulated and added back into the document per the subcommittee's request (see above). The modified document was originally proposed (by the Task Group) as a replacement Subcommittee Type B proposal. This provoked very little discussion so the author moved an amendment (from the floor) to raise the document to Subcommittee Type A status. The amendment passed 13:1. This produced rather more discussion of the document (as was desired). The document passed (as SC-A) without further modifications by a vote of 13:2:4 with the following Pros and Cons (number in parentheses is the number of times cited).

PRO	CON
1. Adds needed functionality (8)	1. Cost of implmentation exceeds benefit (3)
2. Something similar has been implemented (4)	2. Significant vendor impact (7)
3. This version is non-ambiguous (5)	

June 1998, Boston

Brought before the full MDC for approval as an MDC Type A document. Only the history and discussion were changed from the previous version. The motion carried, 11:1:7, with the following PROs and CONs:

PRO	CON
1. Extends existing functionality (2)	1. Difficult to implement (1)
2. Function has been implemented (2)	