Subject: comments on I++ DME Spec version 1.1 To: horst@cme.nist.gov
Content-Type: TEXT/plain; charset=us-ascii
Content-MD5: Ppq2uhJdcc4ObBNziTTTfQ==
Hi John -
Three files with my comments on the I++ DME Spec are attached in PC text
format:

1. editorial comments on version 1.09
2. technical comments on version 1.09
3. technical comments on version 1.1

The technical comments on version 1.1 are in addition to the technical comments on version 1.09 (there were only minor differences between those two versions). All the comments on version 1.09 apply also to version 1.1.
Tom K.

Content-Type: TEXT/plain; name="comments_edit_1.09.txt"; charset=US-ASCII; x-unix-mode=0644
Content-Description: comments_edit_1.09.txt
Content-MD5: nLTioLQ1qk/s9+4vNnrwGw==
Minor Editorial Comments on Release 1.09 of the I++ DME Interface
T. Kramer
June 14, 2002

These comments are only minor editorial comments (spelling, punctuation, grammar), all of which are expected to be completely uncontroversial. A secretary could make all the suggested changes in a day or less. Line numbers refer to the line of the printed page. Blank lines are not counted.

General
The body of these comments is arranged by section, and specific suggestions are made. Recurring editorial problems are described here. Most of the specific suggestions are instances of these problems.

1. Many periods are missing at the end of phrases or sentences.
2. Many phrases starting with "that" have been set off by commas but should not be set off by commas.
3. The word "send" should be "sent" most (but not all) of the places it is used.
4. Daemon is often misspelled "deamon".
5. Incorrect double quotes are often used. Open and close quotes should be raised above the line, not lowered. Open quotes should look like 66 and close quotes should look like 99.
6. The terms "section" and "chapter" are used as synonyms. One of the two should be used consistently. In the specific suggestions "section" is used.
7. The same spelling of all terms should be used consistently, but this practice is not followed. For example, the terms "info port" and "application port" are introduced with

those spellings, but are often spelled with a dash between the two words instead of a space.

Table of Contents
1. For section 5.4, delete ", see header files in subdirectories chapter 9".
2. After section 8.1, delete the entry beginning "F1: Error ...".
3. After 9.5.5, add a line for section 10, Multiple arm support.
4. What is currently called section 10, should be called Appendix A,
and "10" should be changed to "A" on all 26 lines where "10" appears.

Section 1.3
line 2 - "workout" should be "work out".
Section 1.4
line 6 - "availavle" should be "available".
line 7 - "Scenarios" should be "scenarios".

Section 1.5
2nd bullet, second line - Fix the quotes.
5th bullet, third line - Delete comma before "that".
next-to-last line - "developments, this" should be "development. This".
last line - "equipments" should be "equipment".

Section 2
line 1 - "for helping to" should be "to help".

Section 2.1
line 5 - "While PC" should be "While the PC".
line 5 - "provide" should be "provides"
line 5 - Insert a comma after "interface" near the end of the line.
line 9 - "system" should be "systems".

Section 2.2
line 2 - Delete the comma before "that".
line 3 - Delete the comma before "that" and the comma after "protocol".

Section 2.3
In Picture 4, "Applicatio" should be "Application".

Section 2.4
line 1 - Fix the quotes.

Section 2.4.1
last two lines - "to change" should be "changing".

Section 2.4.2
line 5 - "examples" should be "example,"

Section 2.4.3
line 2 - "setup" should be "set up".

Section 2.4.4
line 4 - "info-port" should be "info port".
line 5 - "application-port" should be "application port".

Section 3.2.1
In the grey box, "cycles are possible" should be "cycles is possible".

Section 3.2.2
In the grey box "information information" should be "information".

Section 3.2.4
In the box under "StartSession", "occures" should be "occurs".

Section 4
line 4 - "diagramm" should be "diagram".
line 4 - "chapter" should be "section".
In the diamond in Picture 11 - "coloumn" should be "column".
In the box at the right in Picture 11 - "deamons" should be "daemons".

Section 4.1
line 2 - "as" should be "than".

Section 4.3
line 2 - "daemons" should be "daemon".
line 2 - "StopDeamon" should be "StopDaemon".
line 2 - Fix the quotes.

Section 5.1
1st bullet - "of a real" should be "of real".
2nd bullet - "this" should be "these"
4th bullet - "chapter" should be "section".
6th bullet - "consistent to" should be "consistent with".
6th bullet - "chapter" should be "section".

Section 5.2
In the "server" box - "StopDeamon" should be "StopDaemon".
In the "server" box - "StopAllDeamons" should be "StopAllDaemons".

Section 5.4
In the heading - Delete ", see header files in subdirectories chapter 9" Note that documents do not have subdirectories, and chapter 9 has no header files. Appendix A is probably intended. If so, a line of explanation beginning the section would be appropriate.

Section 6.1
line 2 - "application-port" should be "application port".

Section 6.1.1
line 1 - "send" should be "sent".
line 7 - "send" should be "sent".
line 13 - "Temperatur" should be "Temperature".

Section 6.2
line 4 - "send" should be "sent".
line 4 - "called CommandLine" should be "called a CommandLine".
line 5 - "send" should be "sent".
line 5 - "called ResponseLine" should be "called a ResponseLine".

Section 6.2.1
line 8 - "Examples for tag" should be "Examples of tags".
line 18 - "independend" should be "independent".
line 21 - "Examples for tag" should be "Examples of tags".
line 27 - Insert a comma after "As for a CommandLine".

Section 6.2.2.1
line 1 - "command line" should be "CommandLine".
line 2 - "Character" should be "The character".

Section 6.2.2.2
line 2 - "Character" should be "The character".
line 3 - "Character" should be "The character".
line 8 - "meaning of character" should be "meaning of the character".
line 9 - "Character" should be "The character".

Section 6.2.3
line 5 - "Transaction can overlap" should be "Transactions can overlap".
line 5 - Delete the comma before "that".
line 7 - "transaction" should be "transactions".
line 7 - "send" should be "sent".
line 11 - "transaction" should be "transactions".
line 11 - Delete the comma before "that".
line 12 - "send" should be "sent".
line 13 - Insert a comma after "00002".

Section 6.2.3.1
1st box on right - Fix the quotes.

Section 6.2.4
line 5 - "send" should be "sent".

line 5 - "transaction complete" should be "Transaction complete"
Note: Using "TransactionComplete" everywhere would be better, but the term "Transaction complete" is what has been used before this line.
line 6 - "a event" should be "an event".
line 7 - "of th" should be "to the".
line 7 - "transaction complete" should be "Transaction complete"

Section 6.2.4.1
1st box on right - "A unsolicited" should be "An unsolicited".
Sentence between examples - Put a period at the end.
Box on left starting with "E0553" - "dis" should be "Dis".

Section 6.2.5
line 1 - Put a comma after "condition".

Section 6.3
line 2 - Delete the comma before "that".
line 2 - "that the in the" should be "that in the".
line 3 - "and therefore" would be better as "and is therefore".

Section 6.3.1
line 1 - At the end of the line, "a" should be "an".

Section 6.3.1.1
line 1 - "on TCP/IP" should be "on the TCP/IP".
line 1 - "chapter" should be "section".
line 6 - Put a period at the end.
line 8 - "Error" should be "Errors".
line 12 - Delete the comma at the end of the line.

Section 6.3.1.2
line 2 - "chapter" should be "section".
line 6 - "Error" should be "Errors".
line 7 - Delete the comma before "that".
line 7 - "send" should be "sent".
line 8 - "afterit" should be "after it".

Section 6.3.1.3
line 3 - Put a period at the end.
line 5 - "Error" should be "Errors".
line 5 - Put a period at the end.

Section 6.3.1.4
line 1 - "stop a all" should be "stop all".
line 3 - Put a period at the end.
line 5 - "Error" should be "Errors".

line 5 - Put a period at the end.
line 6 - Delete the comma before "that".
line 6 - "send" should be "sent".

Section 6.3.1.5
line 3 - Put a period at the end.
line 5 - "Error" should be "Errors".
line 5 - Put a period at the end.

Section 6.3.1.6
line 1 - Put a period at the end.
line 3 - Put a period at the end.
line 4 - Put a period at the end.
line 5 - "Error" should be "Errors".
line 5 - Put a period at the end.

Section 6.3.1.7
line 1 - Put a period at the end.
line 3 - Put a period at the end.
line 5 - "Error" should be "Errors".
line 5 - Put a period at the end.
Examples, 1st box on left - "1000)" should be "1000))".
Examples, 3rd box on left - "300)" should be "300))".
Examples, 9th box in middle - Fix the quotes.
Examples, 11th box in middle - Fix the quotes.
Examples, 20th box on right - "a error" should be "an error".

Section 6.3.1.8
line 1 - Put a period at the end.

Section 6.3.1.9
line 1 - "methods" should be "method".

Section 6.3.1.10
line 1 - "methods" should be "method".

Section 6.3.1.11
line 1 - "methods" should be "method".

Section 6.3.1.12
line 1 - "methods" should be "method".
line 3 - "a object" should be "an object".
line 10 - Fix the quotes.
line 11 - Fix the quotes.

Section 6.3.1.13

line 1 - "methods" should be "method".
line 3 - "a object" should be "an object".
line 8 - "propery" should be "property".
line 10 - Fix the quotes.
line 11 - Fix the quotes.

Section 6.3.2.1
line 6 - Put a period at the end.

Section 6.3.2.2
line 1 - Put a period at the end.
line 3 - Put a period at the end.

Section 6.3.2.3
line 4 - Put a period at the end.

Section 6.3.2.4
line 4 - Put a period at the end.

Section 6.3.2.6
line 5 - "chapter" should be "section".
line 8 - "commands" should be "command".

Section 6.3.2.7
line 6 - "chapter" should be "section".
line 7 - "send" should be "sent".

Section 6.3.2.9
line 2 - "of server" should be "of the server".

Section 6.3.2.10
line 1 - "of server" should be "of the server".
line 10 - "Active)" should be "Active.".

Section 6.3.2.11
last line - "with "Get" requestable parameters" should be
"parameters requestable with "Get"".

Section 6.3.2.12
line 12 - "stop to move" should be "stop moving".

Section 6.3.2.13
line 14 - "a IJK" should be "an IJK".
line 16 - "shifted into" should be "shifted in the".
line 27 - "called end" should be "called the end".
line 28 - "to end of" should be "to the end of".

line 36 - "into IJK" should be "in the IJK".
line 40 - "If a IJK" should be "If an IJK".
line 45 - "define a IJK" should be "defines an IJK".
line 47 - "accelleration" should be "acceleration".
line 48 - Insert a comma after "move".
line 54 - "into IJK" should be "in the IJK".
line 55 - The large gap between "this" and "position" should be closed.

Section 6.3.2.15
line 5 - Put a period at the end.
line 6 - Put a period at the end.

Section 6.3.2.16
line 1 - "Method" should be "The client uses this method".
line 3 - "to search" should be "to search for" (or "for which to search").
line 3 - Put a period at the end.
line 4 - Put a period at the end.
line 5 - Put a period at the end.

Section 6.3.2.17
line 1 - "A pointer" should be "This method acts as a pointer".
line 4 - Put a period at the end.
line 5 - Put a period at the end.

Section 6.3.2.18
line 1 - "Method" should be "The client uses this method".
line 1 - "to change tool" should be "to change the tool".
line 1 - Put a period at the end.
line 3 - Put a period at the end.
line 4 - Put a period at the end.
line 5 - "..." should be ".".
line 7 - Delete the comma before "which".

Section 6.3.2.19
line 1 - "Forces" should be "The client uses this method to force".
line 1 - Delete the comma after "assume".
line 3 - Put a period at the end.
line 4 - Put a period at the end.
line 5 - "..." should be ".".
line 7 - Delete the comma before "which".
line 8 - "assumes, active" should be "assumes the active".

Section 6.3.2.20
line 1 - "Forces" should be "The client uses this method to force".
line 12 - Put a period at the end.
line 13 - Put a period at the end.

lines 14 and 15 - Delete these lines. They duplicate the
lines at the end of section 6.3.2.19 and refer to
the command in that section.

Section 6.3.2.21
line 1 - "Pointer" should be "This method acts as a pointer".
line 1 - Put a period at the end.
line 3 - Put a period at the end.
line 4 - Put a period at the end.

Section 6.3.2.22
line 1 - "Pointer" should be "This method acts as a pointer".
line 1 - Put a period at the end.
line 3 - Put a period at the end.
line 4 - Put a period at the end.

Section 6.3.3
line 9 - "XX" should be replaced by a number.
line 10 - "multiplearm" should be "multiple arm".
line 10 - Put a period at the end.
line 11 - "is here listed" should be "is listed here".
line 11 - "consitancy" should be "consistency".

Section 6.3.3.1
line 7 - Put a period at the end.
line 9 - Put a period at the end.

Section 6.3.3.2
line 4 - Put a period at the end.
line 9 - Put a period at the end.

Section 6.3.3.3
line 7 - Put a period at the end.
line 8 - Put a period at the end.
line 10 - "Definition" should be "The definition".
line 10 - "of relation" should be "of the relation".
line 10 - "in C++" should be "in the C++".

Section 6.3.3.4
line 2 - "Enumarator" should be "Enumerator".
line 7 - Put a period at the end.
line 11 - Put a period at the end.
line 12 - Put a period at the end.

Section 6.3.3.5
line 3 - Put a period at the end.

line 5 - Put a period at the end.
line 6 - "as argument" should be "as an argument".

Section 6.3.3.6
line 3 - Put a period at the end.
line 5 - Put a period at the end.
line 6 - "as argument" should be "as an argument".

Section 6.3.3.7
line 3 - Put a period at the end.
line 5 - Put a period at the end.
line 6 - "as argument" should be "as an argument".

Section 6.3.3.8
line 3 - "I,j,k" should be "i,j,k".
line 3 - Put a period at the end.
line 4 - Put a period at the end.

Section 6.3.3.9
line 2 - Put a period at the end.
line 6 - Put a period at the end.
line 8 - "as argument" should be "as an argument".

Section 6.3.3.10
line 2 - Put a period at the end.
line 6 - Put a period at the end.
line 8 - "as argument" should be "as an argument".

Section 6.3.3.11
line 2 - Put a period at the end.
line 6 - Put a period at the end.
line 8 - "as argument" should be "as an argument".

Section 6.3.3.12
line 1 - "I,j,k" should be "i,j,k".
line 2 - "I,j,k" should be "i,j,k".
line 3 - Put a period at the end.
line 4 - Put a period at the end.
line 5 - Put a period at the end.

Section 6.3.4
line 1 - "ToolChanger" should be "of the ToolChanger class".
line 2 - "-Chapter" should be " section".

Section 6.3.5
line 2 - Put a period at the end.

Section 6.3.5.1
line 1 - "Pointer" should be "This method acts as a pointer".
line 1 - Put a period at the end.
line 3 - Put a period at the end.
line 4 - Put a period at the end.

Section 6.3.5.2
line 1 - "Pointer" should be "This method acts as a pointer".
line 1 - Put a period at the end.
line 2 - "GoToPar()" should be "PtMeasPar()".
line 3 - Put a period at the end.
line 4 - Put a period at the end.

Section 6.3.5.3
line 1 - "Method" should be "The client uses this method".
line 1 - Put a period at the end.
line 3 - Put a period at the end.
line 4 - Put a period at the end.
line 5 - "Errormessages" should be "Error messages".
line 5 - Put a period at the end.

Section 6.3.6
line 4 - "this" should be "these".
line 4 - "splitted" should be "split".
line 11 - "associated to" should be "associated with".
line 12 - "chapter" should be "in section".
line 12 - "method" should be "methods".

Section 6.3.7
line 7 - "this" should be "these".
line 7 - "splitted" should be "split".
line 14 - "chapter" should be "in section".
line 14 - "method" should be "methods".

Section 7.1
1st box on right - "boot" should be "booted".

Section 7.4
title - "axis" should be "axes".
7th box on right - "axis" should be "axes".

Section 8.1
last line - Put a period at the end.

Section 8.2
box for error 0512 - "activ" should be "active".

Section 9.1
line 2 - "try outs" should be "tryouts".
line 2 - ".." should be ".".
line 5 - "proprietory" should be "proprietary".
line 9 - "proprietory" should be "proprietary".
line 11 - "proprietory" should be "proprietary".

Section 9.5.2
last line - Put a period at the end.

Section 9.5.3
last line - Put a period at the end.

Section 9.5.4
line 3 - Delete the comma before "that".
line 3 - ".." should be ".".
line 6 - Delete the comma before "that".

Section 9.5.5
line 3 - Delete the comma before "that".
line 6 - Delete the comma before "that".

Section 10
line 5 - "to to build" should be "to build".
line 10 - "for a building" should be "for building".
line 10 - "transfomation" should be "transformation".
line 10 - Put a period at the end.
line 11 - Put a period at the end.
line 12 - "articact" should be "artifact".
line 12 - Put a period at the end.
line 13 - Put a period at the end.
line 14 - Put a period at the end.
line 15 - "articact" should be "artifact".
line 15 - Put a period at the end.
line 16 - Put a period at the end.
line 17 - "transfomation" should be "transformation".
line 17 - Put a period at the end.
line 19 - Put a period at the end.
line 20 - Put a period at the end.
line 22 - Put a period at the end.
line 23 - Put a period at the end.

Appendix A
In this appendix, many of the comments are placed too far right, so that they wrap around to the next line. These should be moved to the left so they do not wrap around.

Section A.1
heading - "1" should be "A.1" and should be aligned left.

Section A.2
heading - "2" should be "A.2" and should be aligned left.

Section A.2.1
heading - "1" should be "A.2.1" and should be aligned left.
line 13 - "disconnent" should be "disconnect".
line 13 - Move comment left.
line 16 - Move comment left.
line 18 - Move comment left.
line 20 - Move comment left.

Section A.2.2
heading - "2" should be "A.2.2" and should be aligned left.

Section A.2.3
heading - "3" should be "A.2.3" and should be aligned left.
line 16 - "confirme" should be "confirm".
line 16 - Move comment left.
line 21 - "confirme" should be "confirm".
line 21 - Move comment left.
line 24 - Move comment left.
line 25 - "wong" should be "wrong".
line 35 - Move comment left.
line 37 - Move comment left.
line 51 - Move comment left.

Section A.3
heading - "3" should be "A.3" and should be aligned left.

Section A.3.1
heading - "1" should be "A.3.1" and should be aligned left.

Section A.4
heading - "4" should be "A.4" and should be aligned left.

Section A.4.1
heading - "1" should be "A.4.1" and should be aligned left.
line 29 - Move comment left.
line 31 - Move comment left.

Section A.4.2
heading - "2" should be "A.4.2" and should be aligned left.
line 13 - Move comment left.

line 22 - Move comment left.
line 26 - Move comment left.
line 27 - "calcalate" should be "calculate".
line 28 - Move comment left.
line 29 - "matriz" should be "matrix".
line 39 - Move comment left.
line 42 - Move comment left.
line 44 - Move comment left.
line 46 - Move comment left.
line 57 - Move comment left.

Section A.4.3
heading - "3" should be "A.4.3" and should be aligned left.
line 20 - Move comment left.

Section A.5
heading - "5" should be "A.5" and should be aligned left.

Section A.5.1
heading - "1" should be "A.5.1" and should be aligned left.

Section A.5.2
heading - "2" should be "A.5.2" and should be aligned left.

Section A.5.3
heading - "3" should be "A.5.3" and should be aligned left.

Section A.5.4
heading - "4" should be "A.5.4" and should be aligned left.

Section A.5.5
heading - "5" should be "A.5.5" and should be aligned left.

Section A.5.6
heading - "6" should be "A.5.6" and should be aligned left.

Section A.5.7
heading - "7" should be "A.5.7" and should be aligned left.

Section A.6
heading - "6" should be "A.6" and should be aligned left.

Section A.6.1
heading - "1" should be "A.6.1" and should be aligned left.

Section A.6.2

heading - "2" should be "A.6.2" and should be aligned left.
line 7 - "angel in degree" should be "angle in degrees".

Section A.6.3
heading - "3" should be "A.6.3" and should be aligned left.
line 8 - Move comment left.
line 16 - Move comment left.

Section A.6.4
heading - "4" should be "A.6.4" and should be aligned left.

Section A.6.5
heading - "5" should be "A.6.5" and should be aligned left.

Content-Type: TEXT/plain; name="comments_tech_1.09.txt"; charset=US-ASCII; x-unix-mode=0644
Content-Description: comments_tech_1.09.txt
Content-MD5: Cm7eIMfNUVNCT7CDW5S5XA==Technical and non-Minor Editorial Comments on Release 1.09 of the I++ DME Interface
T. Kramer
June 14, 2002

This document gives technical comments and non-minor editorial comments. Minor editorial comments are in a separate document.
1. Error Handling, Generally
The document is generally weak on error handling. A number of the comments below concern specific aspects of error handling. Addendum A to these comments is a proposed revision of part of Section 8, Error Handling.

2. Syntax Specification, Generally
For a specification that includes messaging (as this one does) to support plug compatibility, the syntax of messages must be completely and unambiguously specified. This is not an abstract issue. It is a problem that arises, almost inevitably, when an effort is made to establish a working interface between a client and a server that have been built by developers who are not in close communication. If the syntax is not tightly specified, or if either party fails to implement the specification correctly, message exchange will fail.

If, for example, it is not specified that error numbers cannot have plus signs before them, someone building a server will have his system put plus signs on error numbers, and someone building a client will have his system be unable to read plus signs on error numbers. Both systems will be in compliance with the specification, but the two systems will not work together.

The easiest way to produce a tight syntax specification is to use a syntax specification language such as EBNF. This is not essential, however. It is possible to produce a tight specification in natural language. The specification is currently far from being tight. Several of the comments below go into specific details of syntax.

3. Case Sensitivity
The document does not state anywhere whether method names and error texts are case sensitive. The document appears to intend that method names be case sensitive, since the names used contain a mixture of upper and lower case letters. The document gives no clues as to whether error text is case sensitive. In the remainder of these comments, it is assumed that case sensitivity applies.

Whatever is intended by the authors regarding case sensitivity, it should be explicitly stated.

4. The Space Character
The document does not state anywhere what use may be made of space characters, except for provisions in Sections 6.2.2.1 and 6.2.2.2, which apply only to positions 6 and 8 of messages. The easiest to implement requirement on space characters is to allow them only in strings. The next step up is to add allowing a single optional space after any comma. This helps readability for humans. The next step up is to add allowing any number of optional spaces before or after any punctuation. The most extreme is to allow any number of spaces anywhere, with the proviso that the message means exactly the same thing if all the spaces outside of strings are removed. The RS274 language is this way. Whatever decision about spaces is made, it needs to be explicit.

5. Data Types
Data types are inadequately described in several parts of the document. In Section 6, data types of arguments to methods are not given. They should be given explicitly in Section 6. In several cases, it is not currently possible to tell what the data type should be.
For example, in Section 6.3.1.3, an EventTag is the argument to StopDaemon. Is the EventTag to be written with quotes or without quotes? In Section 4.3 quotes are used. In Section 6.2.4.1, quotes are not used. As another example, in Section 6.3.3.1, SetCoordSystem can take one of four identifiers as an argument. Is the argument to be enclosed in quotes or not? The examples in Sections 7.2 and 7.4 show the argument without quotes. Is this correct? As a third example, is it incorrect to write an argument to the X, Y and Z methods without a decimal point?

6. Command String Format
The syntax of command strings to be transmitted in socket messages is not adequately described. Requirements on the first six characters of these strings are well-described in Section 6.2.2.1. The examples are very helpful, but some of the practices followed in the examples are either not described in the text or differ from what is described in the text. For example, in Section 6.3.2.11, the last line of page 41 indicates that Tool().A() may be used as an argument to the Get method. This syntax (a period between two methods) is not explained anywhere in the document. The syntax is presumably normal object model syntax meaning run the Tool() method to get the currently active tool and then run the A() method on that tool. In the dialog examples, however a different syntax is used

consistently, in which all but the last set of parentheses is omitted. Specifically, in Sections 7.3 and 7.5, Tool.A() is used. In Section 7.7 the form A.B.C() is used everywhere rather than the form A().B().C(). The A.B.C() form is simpler and easier to parse, so that form is used in Addendum B to these comments.

## 7. Response String Format

The syntax of response strings to be transmitted in socket messages is not adequately described. Requirements on the first eight characters of these strings are well-described in Section 6.2.2.2, but the description of the syntax of the remainder of response strings is very inadequate.

a. For the methods which take enumerations as the argument list (see the comment on Enumeration below), there is no description of requirements on the return data. For example, it is not stated:
i. whether on not it is an error to return data in a different order from that in the command (or from a some other specified order).
ii. whether or not it is an error to omit requested data from the return message.
iii. whether or not it is an error to put un-requested data in the return message.

b. The description of error response strings in Section 8.1 is especially poor. A proposed revised version of Section 8.1 is given in Addendum A to these comments. In addition to issues of case and space characters already described, Section 8.1 has the following shortcomings.
i. The first line of Section 8.1 appears to be intended to describe the format of an error message, but there is no text saying that. The line itself "0003 ! Error(F1, F2, F3, Text)" is a mixture of example, literal text, and fields intended to be replaced with appropriate characters in actual messages. The "0003" is presumably an example of a CommandTag. As an example, it is incorrect because it has only four characters. All the punctuation and "Error" are presumably literal text. F1, F2, F3, and Text are presumably fields. The document explains none of that.
ii. It is not clear whether error numbers must be put in with four digits (including leading zeros), or whether leading zeros are optional or forbidden. It is also not stated whether a plus sign, a decimal point, or trailing zeros after a decimal point may be used. Currently, even scientific notation for error numbers appears to be allowed, as described on page 28.
iii. It is not clear whether the "Text" should be considered a string, so that it must be enclosed in quotes, or whether it should be put into the error message without quotes.
iv. For field F3 there is no description of what the data type is and there is no requirement for what the field should contain. There is only a recommendation that the name of the method that caused the error be put into the field. If a method name is to be used, it is not clear whether the name is considered to be a string.

c. Not one of the examples of error messages in the document conforms to the requirements for an error message. The examples should be changed to be correct.
i. In the example on page 31, the severity code is wrong (2 should be 3), the error number is wrong (1 should be 2500), the error message is wrong (only the first word should begin with an upper case letter), and the error text is not enclosed in quotes.

ii. In the example on page 32, the error number is wrong (9 should be 0500), the method name is meaningless (HealthCheck is nowhere defined), the error message text is wrong (should be "Emergency stop"), and the error text is not enclosed in quotes.
iii. In the four examples on page 36, the first three fields of the error message have been omitted, all four error texts use upper case letters in places where lower case letters should be, and only the first two examples put quotes around around the error text.

8. Number of Tags Available
In Section 6.2.1 on page 29, it is provided that there is a single tag counter for ordinary commands and event commands, and the counter produces numbers from 00001 to 09999. This allows a total of 9999 tags, which is not a large number in computer operations. With the total 9999, it must be anticipated that rollover will occur frequently. If allowing a larger total number of tags would not affect command transmission adversely, it would be useful to increase the total allowed by a factor of 100 or more. Regardless of the total number of tags available, it will avoid confusion if the manner of rollover is clearly prescribed.

9. Purpose
The document does not adequately explain what its purposes are. If the document does not say what capabilities the interface is intended to provide: (a) users will not know what it is supposed to be good for, and (b) it will not be possible for anyone but the authors to determine if the interface provides the capabilities it is intended to provide. Currently, only the sentence "The spec allows to access all basic DME functionality." in section 1.2 describes a purpose. A leading CMM application developer has stated that it is important to his customers (and therefore to him) that they have to learn how to use a only single user interface. If the I++ DME-Interface does not provide the capability to run the DME manufacturer's calibration routines from the application's user interface, it will not be possible for the application developer to meet his users' needs. The document should state explicitly whether it intends to support running the DME manufacturer's calibration routines via the I++ DME-interface. An important capability for system builders is plug-compatibility (also called plug and play). The idea of plug-compatibility is that it should be possible (a) to have a given server be able to run the same software regardless of what the client application is, and (b) to have a given application be able to run the same software regardless of what the server is. The I++ DME-interface does not say whether it intends to support plug-compatibility. It should explicitly say whether it does or not.

10. Wording in Section 2.4
On page 13, Section 2.4 is a little confusing because:
a. The term "DME-Interface" is used to name both hardware and software.
b. The term "driver" appears without having been defined and the driver is given responsibility for starting TCP/IP ports. The following text is suggested to replace the first five lines of Section 2.4 The DME-Interface is a piece of software that runs on a PC based or "black box" based piece of hardware. This hardware connects to all subsystems (Picture 3). The DME-Interface handles all subsystems and provides TCP/IP sockets for communication. When the hardware is powered up and the DME-Interface is started, the DME-Interface will create up to 4 TCP/IP ports:

11. Wording in Section 2.4.4

On page 13, in Section 2.4.4, the first sentence is confusing because it is not clear whether the info is obtaining information or providing information. The following text is suggested to replace the first sentence of Section 2.4.4 The info is a piece of software that runs on a client computer. The info obtains information from the DME-Interface through the info port and provides the information to the client.

12. Character Set

The character set to be used is described in the first sentence of Section 6.1.1 as "7 bit ASCII characters". It would be better to describe them as 8-bit ASCII characters with the 128's bit set to zero, since a byte is 8 bits. If they are described as 7-bit characters, some implementation is going to conduct hidden communications using the eighth bit.

13. Characters Per Line

In Section 6.2 on page 29, is the sentence "The maximum number of characters in a line should not exceed 65535." It is probably intended to allow (65536 = 2 ** 16) characters on the line including the line terminator, but it is not stated whether the line terminator is included in the count, and the line terminator is two characters, not one. It is suggested that the sentence be changed to either:  The maximum number of characters in a line, including the <CR><LF> at the end, should not exceed 65536. Or The maximum number of characters in a line, excluding the <CR><LF> at the end, should not exceed 65534.

14. Tag Numbers

The material in Section 6.2.1 on page 29 describing tag numbers is not clear.
a. It is not clear whether incrementing the tag number is optional or required. It only says the client is responsible for incrementing.
b. The amount of the increment is not specified.
c. It is not specified what tag number should be given to the command following the command with tag number 09999 or E9999.
d. The fact that there are only 9999 numbers is somewhat obscured by using five-digit numbers for CommandTags.
e. The fact that there is a single counter for CommandTags and EventTags is buried in the text.
f. It is not specified whether the first counter number used in a session must be a specific number, such as 0001.
g. It is not stated that the first set of examples are CommandTags and the second set of examples are EventTags. A proposed revised version of Section 6.2.1 is included as section 6.2.3 of Addendum B to these comments.

15. Client and Server Terminology

The term "server" is frequently used to mean the DME-Interface, but "server" is never defined, and it is never explained that it means the DME-Interface. A sentence should be added somewhere (perhaps near the beginning of Section 2) explaining that "server" is a synonym for DME-Interface. Also, "client" is often used as a synonym for "application", but this is not explained, either. Section 2.4.1 says "The application is a piece of software that runs on the client computer"; thus, Section 2.4.1 would be a good place to explain that "client" is often used as a synonym for "application".

## 16. Out-of-range Characters

The the following requirement in Section 6.1.1 is excessive. Any character sent outside of this range will cause a "Fatal ServerError". An out-of-range character is a syntax error and should be in severity class 2, as are several other syntax errors. It is suggested that this error be called "Illegal character" and have Error No. 0007. There is currently no error message in Section 8.2 for an out-of-range character, despite what Section 6.1.1 says.

## 17. Normative vs. Non-Normative Material

It is not clear how much of the document is intended to be normative. For an interface description, only the methods used by the client (to which the server responds) are required. The internals of the server need not be specified. The methods to which the server responds are all included in Section 6.3, entitled "Method Syntax". This Section would appear to be normative. This Section matches the model shown in Section 5.2 (except for several ABC methods), so Section 5.2 is also normative. Section 5.2, however, does not include the argument names and types or the return values, as shown in Sections 5.3 to 5.10. Section 6.3 does not include that information, either. That information should be added to Section 5.2 and to Section 6.3. Sections 5.3 to 5.10 duplicate the information of Section 5.2 as well as containing much additional information. The additional information is not needed for an interface protocol, so Sections 5.3 to 5.10 should not be normative. It would be better to move them to an appendix.

## 18. ABC

Commands involving ABC are included in the object models of Sections 5.2 and 5.3, and in Appendix A. These appear to be meant for rotary axes. There is, however, no textual description of what they mean, and they are not included in Section 6.3, where all the other methods are described. It is suggested that all the ABC material be removed from the document until later versions, in which text describing what is intended is present along with the models.

## 19. Section Numbers Missing

In Section 6.1.1 (Character set), there are subheadings for Units, String formats, and Number formats. None of these is a subtopic of character set. Each should have a section number.

20. Define Enumeration

The word "enumeration" is used with a consistent meaning in Sections 6.3.1.9, 6.3.1.11, 6.3.2.6, 6.3.2.7, 6.3.2.11, 6.3.2.12, and 6.3.2.13, but it is never defined. It is suggested that the following definition be included at some point in the document before "Enumeration" is first used to describe arguments. An enumeration is a list of zero to many items from a specified list of candidate items. Each item may appear in the enumeration list at most once, and the order in which the items appear on the enumeration list is not significant. For example, if the candidate list is (a, b, c, d, e):

1. (a, b, d) and (d, a, b) are the same enumeration, and both are legal.
2. 2. (c, b, c) is illegal because c appears twice.
3. (c, e, h) is illegal because h does not appear in the candidate list.

21. Parameters and Arguments

The document uses the word "parameters" in two senses: (a) the parameters of a method call, and (b) system or tool parameters. The document uses the word "argument" as a synonym for the first meaning of parameter in Sections 6.3 (first paragraph), 6.3.1.9, 6.3.1.11, 6.3.2.3, 6.3.2.11, 6.3.2.12, 6.3.2.13, and 6.3.3.5 through 6.3.3.12. It is confusing to have two meanings for "parameter". It is also confusing to have two words that mean "argument". The best way to fix this would be to use "parameter" only for system or tool parameters, and to use "argument" only to mean the arguments of a method. To implement this, it would be necessary to change "parameter(s)" to "argument(s)" in several dozen places. This would not be hard to do. For example, in Section 6.3.1.12, the third line, which currently is "Parameters A pointer to a object, e.g. parameter block", would be changed to "Arguments A pointer to an object, e.g. parameter block". As another example, the third line of Section 6.3.1.9, which is now "Parameters An enumeration of one or more of the following methods can be argument." would be changed to "Arguments The argument list is an enumeration of one or more of the following methods.". It is particularly hard to tell in the error messages defined in Section 8.2 which sense of "parameter" is intended. It will probably be useful in several cases to have two different error messages (one with "parameter" and one with "argument") where there is now only one error message.

22. Notation Indicating Arguments

In Sections 6.3.1, 6.3.2, and 6.3.3, for methods that have arguments, the document uses three different notations in the model method calls. The first notation does not indicate that the method has arguments. It is simply the method name followed by "()". This notation is used at:
Section 6.3.1.6, line 2 - GetErrorInfo()
Section 6.3.1.9, line 2 - GetProp()

Section 6.3.1.11, line 2 - SetProp()
Section 6.3.1.12, line 2 - EnumProp()
Section 6.3.1.13, line 2 - EnumAllProp()
Section 6.3.2.6, line 3 - OnPtMeasReport()
Section 6.3.2.7, line 4 - OnMoveReportE()
Section 6.3.2.11, line 3 - Get()

The second notation indicates that arguments exist by putting "(..)" after the method name. This notation is used at:
Section 6.3.2.12, line 2 - GoTo(..)
Section 6.3.2.13, line 3 - PtMeas(..)
The third notation puts argument names in the parentheses following the method name. This notation is used at:
Section 6.3.1.3, line 2 - StopDaemon(EventTag)
Section 6.3.2.2, line 2 - IsHomed(Bool)
Section 6.3.2.16, line 2 - FindTool("Tool")
Section 6.3.2.18, line 2 - ChangeTool("Tool2")
Section 6.3.2.19, line 2 - SetTool("Tool2")
Section 6.3.2.20, line 2 - AlignTool(i1,j1,k1, alpha)
Section 6.3.3.1, line 2 - SetCoordSystem(Param)
Section 6.3.3.3, line 2 - GetCsyTransformation(Enumerator)
Section 6.3.3.4, line 2 - SetCsyTransformation(Enumarator,X0,Y0,Z0, Theta, Psi, Phi)
Section 6.3.3.9, line 1 - X(x)
Section 6.3.3.10, line 1 - Y(y)
Section 6.3.3.11, line 1 - Z(z)
Section 6.3.3.12, line 1 - IJK(i,j,k)
The third notation is the best because it is the most informative. Whatever notation is used, it should be the same notation for all of the sections listed above.

23. X, Y, Z

The X(), Y(), and Z() methods each have two versions: one with no argument, and one with one argument which is a target value for a coordinate.
a. The Remarks in the no-argument versions (identical in Sections 6.3.3.5, 6.3.3.6, and 6.3.3.7) refer to OnReport as a method. OnReport is not a method. What appears to be intended is that OnReport means either OnPtMeasReport or OnMoveReportE. The remark should be changed from:
"This method can only be invoked as argument of a Get or OnReport method." to
"This method can only be invoked as an argument of a Get, OnPtMeasReport, or OnMoveReportE method."
b. The first sentence of the Remarks in the no-argument versions (identical in Sections 6.3.3.9, 6.3.3.10, and 6.3.3.11) omits the PtMeasIJK method.
The sentence should be changed from:
"This method can only be invoked as argument of a GoTo or PtMeas method." to
"This method can only be invoked as an argument of a GoTo, PtMeas, or PtMeasIJK method."

24. Length, not Dimension

In Section 6.1.1, the first line under the "Units" subheading says "Dimensions mm". This should be "Length mm".

25. StartSession

In Section 6.3.1.1, it is not clear what should happen (a) if the client fails to send a StartSession command and starts sending other commands, and (b) if the client sends a StartSession command during the middle of a session. Presumably, these are both errors, but no appropriate error message has been defined for them. It would be useful to define error 0008 "Protocol error" with severity 2. The following paragraph is suggested as a replacement for the first paragraph of Section 6.3.1.1. After the client and the server are connected on the TCP/IP level (see Section 9.2), the StartSession method initiates a session between the client and the server. If the server receives any command other than StartSession while a session has been not been started, or if the server receives a StartSession command while a session is in progress, the server must send an error response using error 0008 "Protocol error".

26. AbortE

In Section 6.3.1.5, the description of what AbortE does is missing important details. The following paragraph is suggested as a replacement for the Remarks regarding AbortE. On receiving an AbortE command, the server must: (a) stop all motion as soon as possible, (b) stop executing any currently executing commands, (c) not start any pending commands (those for which an Ack has been sent but for which execution has not yet started), and (d) stop sending data responses for any currently executing commands. For currently executing commands, the server must send either a TransactionComplete (for all event commands and any other commands that are completed) or an error "Transaction aborted" for non-event commands that are not complete. For pending commands, the server must send an error "Transaction aborted". The AbortE command itself is not to be reported complete until the responses just described have been sent. After sending an AbortE command, the client must not send any other commands until a TransactionComplete has been received in response to the AbortE. The next command sent by the client must be a ClearAllErrors command. If the server receives any other command following an AbortE, it must send an error response using error 0008 "Protocol error".

27. ClearAllErrors

In Section 6.3.1.7, the ClearAllErrors method is not provided with any errors. It is likely that some errors will be impossible to clear during a session, so some method of reporting that the ClearAllErrors method was unsuccessful is needed.

28. Error Handling Reference

At the end of Section 6.2.5, add "Further details regarding error handling are given in Section 8.".

29. Tool Handling

In the object model, the toolchanger has an EnumTools method (see page 26), and Appendix A includes EnumTools, but Section 6.3 does not include EnumTools. Assuming only those methods listed in Section 6.3 are in the normative part of the document, there is no way for the client to ask what tools are are available in the server. The EnumTools method needs to be added to Section 6.3.

30. GetProp and SetProp Arguments
Sections 6.3.1.9 and 6.3.1.11 should have either (a) a complete list of methods that may be used as arguments, or (b) a reference to the section of the document in which a complete list can be found.

31. EnumProp and EnumAllProp

The EnumProp and EnumAllProp methods are described in Section 6.3.1.12 and Section 6.3.1.13, respectively. The descriptions of these methods are quite confusing. The first example in Section 7.7 is an EnumProp example. The example does not conform to the description given in Section 6.3.1.12, because it uses the string "Value" where the section says the string should be one of "Number", "String" or "Property". Towards the end of the description of EnumAllProp, the text says "See example 7.7". There is no example of EnumAllProp in example 7.7 (or anywhere else in the document). The existence of these methods seems pointless, anyway, because the properties of objects used in the interface are fixed, and the client knows what they are. If variable properties were allowed in the server, with semantics unknown to the client, these methods would be useless because all the return values reveal is the name and type of the property; the client would have no idea what they mean.

32. OnMoveReportE Arguments

In Section 6.3.2.7, the description of the parameters to OnMoveReportE is hard to understand. It is suggested that the following be used instead: Arguments Time(s), Dis(d), ..., where ... is an enumeration of zero or more of the arguments used with command Get(), Section 6.3.2.11. If the enumeration is empty, no reports are sent; use an empty enumeration to turn reporting off. In addition, this section uses the methods Time() and Dis() which are not defined in the document. These should be defined as methods. Section 6.3 near the definitions of X(), Y(), and Z() would be a good place to define them.

33. Implicit

The description of most methods includes sections headed Parameters, Data, Errors, and Remarks. In five sections (6.3.2.12, 6.3.2.13, 6.3.3.9, 6.3.3.10, and 6.3.3.11), a heading named "Implicit" appears. It is not clear what this heading means, and the heading is explained nowhere. Moreover, in the last three sections in which it appears as "Implicit Tool.GoToPar" (describing the X(x), Y(y), and Z(z) methods) it is inappropriate because X, Y, and Z are just coordinates and have nothing to do with GoToPars. The "Implicit" entries should be deleted in all five sections. In sections 6.3.2.12 and 6.3.2.13, text should be added explaining that GoToPars are used in the execution of those commands (and for that reason they may be considered to be implicit arguments of the command).

34. PtMeas Arguments

In Section 6.3.2.13, the arguments are described as "one or more of" X(..), Y(..), Z(..), IJK(..). It is not clear what values should be used for the coordinates of the target point if one or more of X(..), Y(..), Z(..) is missing from the argument list. Reasonable guesses would be (a) for the missing coordinates, use the coordinates of the point at which the machine is before execution of the command begins, or (b) for the missing coordinates, use the coordinates of the last point that was PtMeas'd. The best solution to this would be to require X, Y, and Z and make IJK optional. The description of the command appears to assume that this is what the requirement is. Also, it should be an error to omit IJK and have the probed point be the same as the current point, since then IJK cannot be computed and the method as described will not work.

35. PtMeas End of Search

In Section 6.3.2.13, in the description of the action of PtMeas when no IJK is given, the second bullet says "The values of X, Y, Z define the end of search position.". This may not be what is intended. It is more likely (and certainly more appropriately) intended that the end of search position should be found the same way in this case as when IJK is given, namely by moving opposite the computed IJK direction from X,Y,Z a distance given by the value of Tool.Search().

36. FindTool Data

In Section 6.3.2.16 the text says "Data FoundTool". The example in Section 7.7 is inconsistent with this. In the example, the FindTool Command is reported complete with no data having been returned. Either the text in the section should be changed to "Data None." or another line should be added to the example showing data being returned.

37. Error Documentation

In Sections 6.3.2.15, 6.3.2.17 6.3.2.21, 6.3.2.22, 6.3.5.1, and 6.3.5.2, the Errors heading has not been completed. The text does not list any errors, and it does not say "Errors None." An entry of some sort should be included after the Errors heading in each of these sections.

38. FoundTool

In Section 6.3.2.17, in the description of the FoundTool method, it is not clear (a) what happens if there was no previous call to FindTool (b) what happens if there was a previous call to FindTool but it returned an error. It is suggested that the first two lines of the section be replaced by the following. There are equally reasonable alternatives to the paragraph below. It is not of great importance which is used, as long as items (a) and (b) above are covered. The client uses this method as a pointer to a tool. It is intended to be used after a call to FindTool. If there has been a call to FindToolat any time during the session that executed without error, FoundTool provides a pointer to the tool named in the most recent such call. If there has been no such call, FoundTool returns error 1502, "Tool not found".

## 39. Pointers

Six methods are defined that act as pointers, namely: Tool, FoundTool, GoToPar for DME, PtMeasPar for DME, GoToPar for tool, and PtMeasPar for tool in Sections 6.3.2.15, 6.3.2.17, 6.3.2.21, 6.3.2.22, 6.3.5.1, and 6.3.5.2, respectively. It is not explained anywhere that the only use of these methods (outside of EnumProp and EnumAllProp) is in constructing arguments for other methods and for constructing responses. Also it is not explained that these methods do not get a pointer as data. To the contrary, in Sections 6.3.2.21, 6.3.2.22, 6.3.5.1 and 6.3.5.2, it is incorrectly stated that a pointer is returned as data. Sections 6.3.2.15 and 6.3.2.1 correctly state that no data is returned. There is no pointer data type that can be placed in a ResponseLine, so there is no way to return a pointer as data. Somewhere toward the beginning of Section 6 the use of pointers should be explained. Also, the beginning of the description of each of these methods should be changed to "This method acts as a pointer ...".

## 40. Block Methods

In Sections 6.3.6 and 6.3.7, GoToPar Block and PtMeasPar Block are briefly described and references are given to other sections of the document where more information may be found. Since Section 6 is (presumably) the normative section of the document, the descriptions of GoToPar and PtMeasPar in Section 6 should be expanded to list and describe all the methods that may be used.

## 41. Euler Angles

In Section 6.3.3.4, the error "Theta Out Of Range" may occur, but the correct range of theta is not stated; it should be stated. Also, no range is given for Psi and Phi; ranges should be given for them. If it is possible for Psi and Phi to be out of range, the error messages "Psi out of range" and "Phi out of range" should also be defined. Also, although calculations relating Euler angles to transformation matrixes are included in Appendix A, the meaning of Theta, Psi, and Phi are explained nowhere, and no reference is provided to a book providing an explanation. It would be helpful if either an explanation or one or more references were provided. "Mathematical Methods of Physics". page 376, calls the Euler angles Alpha, Beta, and Gamma. "The Mathematics of Physics and Chemistry", page 286, says "Unfortunately, the Euler angles have been defined in several different ways in the scientific literature and great confusion occurs when one attempts to compare the results of various writers". In view of this, it is not reasonable to assume everyone has the same understanding of Euler angles.

## 42. Out of Limits

The term MoveOutOfLimits is used in sections 6.3.3.9, 6.3.3.10, and 6.3.3.11 but is never defined. In addition, in Section 8.2 error message 2500 is "Machine limit encountered [Move Out Of Limits]". It seems pointless for the error message to say the same thing twice. The error message should be either "Machine limit encountered" or "Move out of limits". Possibly, both error messages are needed and they mean different things. The meaning of the message should be defined and the behaviour should be described more clearly. There are two likely meanings (a) the machine has been moving and it moved to a limit; in this case "Machine limit encountered" would be appropriate, (b) the server has detected before starting to execute a move that if it moved to the target position, it would be out of limits, so the server has decided not to move at all; in this case "Move out of limits" would be appropriate, but "Target position out of limits" would be better.

## 43. Error Severities

The error severities in Section 8.2 do not all seem to be appropriate. Whenever an error can be detected before any action is taken, it should be possible for the client to repair the error, and, hence the error should have severity 2. For example, "Incorrect parameters" and "Illegal command" fit that description but currently have severity 3. As another example, "Parameter out of range" has severity 1 (only a warning), but in many (perhaps most) cases of Parameter out of range, it will be impossible to execute the command as intended (since it is impossible to tell what is intended). Thus, Parameter out of range should have severity 2. The severities in Section 8.2 should all be reviewed.

## 44. Errors of Methods

The Errors subsection of the method descriptions should be more complete in almost all cases. For example, every method is susceptible to having incorrect arguments. Thus, "Incorrect parameters" should be included in the Errors subsection of every method.

## ADDENDUM A -- PROPOSED REVISION OF SECTION 8.1
### 8.1 Error Messages
#### 8.1.1 Syntax of a ResponseLine Reporting an Error
The following line is an example of a ResponseLine reporting an error. 00003 ! Error(1, 1007, "SetCsyTransformation", "Theta out of range") A ResponseLine reporting an error must conform to all provisions regarding such lines given in Section 6.2. The first five positions of a ResponseLine reporting an error must contain a ResponseTag. If the server can determine what command caused the error, the ResponseTag must be the tag of that command. If not, the ResponseTag must be E0000. Positions six, seven, and eight must contain a space (ASCII code = 32), an exclamation point (ASCII code = 33), and another space.  Positions nine to thirteen must contain "Error" (without the quotes). No other

spellings, such as "ERROR" or "error", may be used. Position fourteen must contain a left parenthesis (ASCII code = 40). The ResponseLine must end with a right parenthesis (ASCII code = 41) just before the closing <CR><LF>. Between the left parenthesis and the right parenthesis, there must be four fields separated by single commas. Each comma may, optionally, be followed by a single space. The first field must contain a single digit representing the severity classification.
The second field must contain four digits representing the error number.
The third field is a string. If the server knows what command caused the error, this string should contain the name (only) of the command. If not, this string must be the empty string (two double quote characters).
The fourth field is a string containing the text of the error message exactly as shown in Section 8.2. 8.1.2 Error Severity and Number The error severity character must be one of the following five digits:
0: Information.
1: Warning.
2: Error, the client should be able to repair the error.
3: Error, user interaction is necessary to repair the error.
9: Error, the server is dead or unreliable and must be restarted to be usable.
For the server to continue after receiving an error message with severity 2 or 3, the client must give a ClearAllErrors() command before any other command, or server behaviour is unspecified. The following error number ranges are used. 0000-4999, defined in this specification or reserved for future versions. 5000-8999, definable from server.
9000-9999, definable from client.

## ADDENDUM B -- PROPOSED REVISION OF SECTIONS 6.1 AND 6.2

### 6.1 Communication
Communication between the application (client) and the I++ DME-Interface (server) is based on the standard TCP/IP protocol using sockets as described in Section 3. It uses the application port with port number 1294.

### 6.2 Protocol Basics
#### 6.2.1 Units
Numbers that represent measures must use the following units.
Length: millimeters
Time: seconds
Angle: decimal degrees (no minutes or seconds)
Temperature: degrees Celsius
Force: Newtons
Compound measures use combinations of these units. For example, speed (which is length per unit time) must be expressed in millimeters per second.
#### 6.2.2 Character Set
All bytes received and sent on the application port of the server are interpreted as 8-bit ASCII characters. The 128's bit must always be zero. Only characters in the range from ASCII code = 32 (space) to ASCII code = 126 (~) may be used, except that the character pair Carriage Return (<CR>, ASCII code = 13) and Line Feed (<LF>, ASCII code = 10)

is used as a line terminator. <CR> and <LF> must always be sent as a pair in the order <CR> followed by <LF>. If the server receives a message containing any character outside of this range or containing a <CR> or <LF> anywhere except at the end of the message, the server must send an error response, using error 0007, "Illegal character". Upper case letters and lower case letters are regarded as different letters in this protocol. In other words, the protocol is case sensitive.

6.2.3 Numbers and Strings

Message syntax is described in this document in terms of single characters, tags, method names, numbers, and strings. Tags are as described in Section
6.2.4. Method names must be exactly as shown in section 6.3. Numbers and strings are described in this section.

Numbers are defined in the following hierarchy.

number
Integer
unsigned_four_digit_integer
decimal_point_number
no_exponent_number
exponent_number

A digit is one of the characters 0 to 9 (ASCII code 48 to ASCII code 57).

An unsigned_four_digit_integer consists of exactly four digits. Example: 0287

An integer consists of an optional plus (ASCII code = 43) or minus (ASCII code = 45) sign followed by one to sixteen digits. Example: +287741

Note that every unsigned_four_digit_integer is also an integer.

A no_exponent_number consists of an optional plus or minus sign followed by either (a) a decimal point (ASCII code = 46) followed by one to sixteen digits or (b) one or more digits followed by a decimal point followed by zero or more digits so that the total number of digits before and after the decimal point is not more than sixteen. Examples: (a) -.3090 (b) 5.31 An exponent_number consists of an optional plus or minus sign followed by one non-zero digit followed by a decimal point followed by zero to fifteen more digits followed by an E (ASCII code = 69) or an e (ASCII code = 101) followed by an optional plus or minus sign followed by exactly two digits. Example: -2.8843e02

A decimal_point_number is either an exponent_number or a no_exponent_number.

A number is either an integer or a decimal_point_number.

The values of all types of numbers are to be interpreted in the normal way as base ten numbers.

A string consists of a double quote (ASCII code = 34), followed by zero to 255 of the characters allowed by Section 6.2.1 (excluding double quote, <CR>, and <LF>) followed by a double quote.

6.2.4 Tags

There are two types of tags:

CommandTag
EventTag

All tags except the EventTag E0000 are generated by the client. The client must maintain a tag counter with four decimal digits. The tag counter counts four_digit_unsigned_integers from 0001 to 9999. There is only one tag counter. The numbers it generates are used to create CommandTags and EventTags.

The first counter number used in a session must be 0001.
The client must increment the counter by one each time it sends a line, except that after 9999, the counter must roll over to 0001.
A CommandTag is formed by putting the digit zero (ASCII code = 48) before the counter number. This makes CommandTags look like five-digit integers, but that appearance is irrelevant.
An EventTag is formed by putting an upper case E (ASCII code = 69) before the counter number. EventTags are for events, as described in Section 4.
Examples of CommandTags created by the client:
04711 // tag is OK
00020 // tag is OK
20 // error; only 2 characters
10710 // error; first character not zero
00000 // error; out of range, counter must be >=1 and <=9999
Examples of EventTags created by the client:
E4711 // tag is OK
E0333 // tag is OK
e0333 // error; illegal first character, must use upper case E
E0000 // error; out of range, counter must be >=1 and <=9999
E20 // error; only 3 characters
A4711 // error; first character not E
The EventTag E0000 may be created by the server for reporting unsolicited events to the client.
6.2.5 Messages and Lines
Messages sent from client to server or server to client each contain a meaningful line that starts at the beginning of the message and ends with the <CR><LF> pair that terminates the message.
The maximum number of characters in a line, including the <CR><LF>, must not exceed 65536.
A line sent from the client to the server is called a CommandLine.
A line sent from the server to the client is called a ResponseLine.
In the examples given in this document, and in the descriptions of CommandLines and ResponseLines in Sections 6.2.5 and 6.2.6, the terminating <CR><LF> is not included, but it must be included in messages!
In both CommandLines and ResponseLines, the first 5 characters of the line represent a tag.
6.2.7 CommandLine Syntax
A CommandLine consists of a tag followed by space followed by a method name followed by a left parenthesis (ASCII code = 40) followed by either
(a) a right parenthesis (ASCII code = 41) or (b) an argument_list followed
by a right parenthesis. An argument_list consists of one or more arguments separated by commas. A comma must not be placed after the last argument. One space may optionally follow each comma. An argument is a number, a string, or a method_identifier.
A simple_method is a method name followed by a left parenthesis followed by either (a) a right parenthesis, or (b) a number or a string followed by a right parenthesis.

A method_identifier is one of (a) a simple_method, (b) a method name followed by a period followed by a simple_method, or (c) a method name followed by a period followed by a method name followed by a period followed by a simple_method. Example: FoundTool.PtMeasPar.Speed() Only those combinations of method names and arguments defined in Section 6.3 may be used following the first six characters of a CommandLine. Method names are not enclosed in quotes.

## 6.2.7 ResponseLine Syntax

A ResponseLine consists of a tag followed by a space followed by
one of:

(a) the acceptance character, & (ASCII code = 38).
Example: 01912 &

(b) the completion character, % (ASCII code = 37).
Example: 01912 %

(c) the data character, # (ASCII code = 35) followed by a space followed by a data list. A data list has the same syntax as an argument list, the syntax for which is described in Section 6.2.6.
Example: 00004 # X(99.93), Y(17.148)

(d) the error character, ! (ASCII code = 33) followed as described in section 8.1.

During normal command processing by the server, it will use the tag received from the client as the first five characters of the ResponseLine so the client can use this tag to relate the ResponseLine to a CommandLine. A ResponseLine using the acceptance character means that the CommandLine identified by the tag has been received and the first five characters have been recognized as a syntactically valid tag. If the tag is not syntactically valid, the server must not accept the command. The server also must not accept a CommandLine with a tag that is the same as a tag for a CommandLine for which processing has not been completed. In both of these cases the server must report an error using error number 0001, "Illegal tag". The server must accept every command that has no tag error. A ResponseLine using the completion character means all processing of the CommandLine identified by the tag has been completed. A ResponseLine using the data character means the server is returning some data in response to the CommandLine identified by the tag. A ResponseLine using the error character means an error has occurred.
See Section 8.1 for further details.

When no errors occur, the server must send ResponseLines for a given CommandLine in the following order: one with the acceptance character, zero to many with the data character, one with the completion character. ResponseLines for different CommandLines may be sent interleaved (for example: accept command1, accept command2, complete command1, complete command2).

Only those data lists allowed by Section 6.3 may be used in ResponseLines.

In the following we will use

- Ack as a synonym for a ResponseLine where the 7th character is a &
- TransactionComplete as a synonym for a ResponseLine where the 7th character is a %
- Data as a synonym for a ResponseLine where the 7th character is a #
- Error as a synonym for a ResponseLine where the 7th character is a !

Content-Type: TEXT/plain; name="comments_tech_1.1.txt"; charset=US-ASCII; x-unix-mode=0644

Content-Description: comments_tech_1.1.txt
Content-MD5: Rk/9nNru747iEVdLiR5fYA==
Technical and non-Minor Editorial Comments on Release 1.1
of the I++ DME Interface
T. Kramer
October 4, 2002

This document gives technical comments that are in addition to the comments I made on version 1.09 of the spec. All the comments I made on version 1.09 also apply to version 1.1. The reference pages provided below reference both text and examples. Text references are given in parentheses. Example references are given in brackets. Other references are not enclosed. Example: Reference pages: 21 23 (35) [36] 63
This means there is relevant text on page 35 and an example on page 36.

1. Format of Responses
The formats for data responses are irregular and should be made regular.
The formats for commands all follow the pattern <command name>(<something>).
The formats for responses follow no single pattern. Rather, there appear to be three patterns plus non-patterned responses. The three patterns are:
a. response looks like command with data added. There are four instances:
GetMachineClass, GetCsyTransformation, IsHomed, and IsUserEnabled.
Example Response: IsHomed(0) for command IsHomed()
b. response looks like command with "Get" removed and data added. There are two instances: GetCoordSystem, GetErrStatus. Example Response: CoordSystem(PartCsy) for command GetCoordSystem()
c. response looks like command with command name and parentheses removed and data added. There are three instances: Get, GetProp, GetPropE
Example Response 1: X(30), Y(-8.2) for command Get(X(), Y())
Example Response 2: Tool.PtMeasPar.MinAccel(2.0) for command GetProp(Tool.PtMeasPar.MinAccel()) It would be helpful (especially for response parser builders), if responses followed a single simple pattern.
One possible pattern is make the response name for data by removing "get" from the beginning of the command name (if it starts with "get") and adding "data" at the end. The response would have the same parentheses as the command. Using this method, the four examples above might become:
IsHomedData(0)
CoordSystemData(PartCsy)
PtData(X(30), Y(-8.2))
PropData(Tool.PtMeasPar.MinAccel(2.0))
2. GetErrorInfo Response
Reference pages: 16 21 23 (35) 63
The GetErrorInfo command is supposed to return "strings". It is not clear whether this means to make a single response with several strings (presumably separated by commas) or to make one or more responses, each with a single string in it. The spec should state clearly what is intended.
3. GetProp Arguments

Reference pages: 21 23 26 (37) [56] 63 72
The spec is not clear about what should be allowed as arguments. Is it intended that only primitive types (string, bool, number) can be obtained? If this is the intent, some combinations of arguments would be illegal. For example, GetProp(Tool.PtMeasPar()) would be illegal since the value of PtMeasPar is an object, not a primitive type. If this is not the intent, how should data be returned?
The spec is also not clear about whether CanChangeSpeed and CanChangeAccel are allowable keywords. These are included on pages 22, 26 and 71 for GoToPars but are not included for PtMeasPars. None of the examples uses either of them. The spec is not clear whether properties of "param" objects defined on pages 22, 26, 72, and 73 are intended to be directly accessible using GetProp. These properties are not mentioned in section 6 and are not included in any examples. If properties of param objects are intended to be directly accessible, they would presumably be accessed using a command like GetProp(Tool.PtMeasPars.Approach.Max()). Also, if this were the case, would it be understood that a command like GetProp(Tool.PtMeasPars.Approach()) is valid and means the same thing as GetProp(Tool.PtMeasPars.Approach.Value())?
4. OnPtMeasReport
Reference pages: 21 24 (40) (43) (50) [54] [55] 66
The spec forbids having no arguments to OnPtMeasReport. With this rule, is not possible to command that nothing should be reported on PtMeas. It should be possible to command that nothing be reported. The simplest way to do this is to get rid of the rule (i.e., allow OnPtMeasReport with no arguments).
5. OnMoveReportE
Reference pages: 16 21 24 [32] (40) 66
What is the intended effect of multiple instances of this command?
The simplest and most reasonable intent would be that if one instance of the command follows another, the later one completely replaces the first. If this is the intent, then presumably the command OnMoveReportE() shuts off all OnMove reports. This is fine, but then what is left for StopDaemon to do? The only command that actually requires a persistent daemon is OnMoveReportE. Also, if this is the intent, commands with only Time and/or Dis have no effect (no way is provided to report time or distance data) and should be made illegal.
Another possible intent is that each OnMoveReportE command has no effect on existing OnMoveReportE daemons, but just creates a new, independent one. If this is the case, why does the spec say "If the enumeration is empty, no reports are sent". Does that mean it is possible to create a daemon that does nothing? If this is the intended effect, then StopDaemon (or StopAllDaemons) must be called to stop the effects of each OnMoveReportE command.
A third possible intent is that later commands modify earlier ones
For example, suppose the following four commands are given:
```
E0001 OnMoveReportE(Time(0.5), Dis(0.2), X())
00002 GoTo(X(20), Z(76))
E0003 OnMoveReportE(Y())
00004 GoTo(X(-20))
```
In this case, only the X position is reported during first GoTo, while both the X position and the Y position are reported during the second GoTo.

If this is the intent, specific rules for how later commands modify earlier ones will be required. Whatever the intent is, it should be clearly stated.

6. GetCsyTransformation and SetCsyTransformation
GetCsyTransformation Reference pages: 21 24 (48) (61) 67
SetCsyTransformation Reference pages: 21 24 (48) (49) [54] [55] (61) 67
It is not clear why MoveableMachineCsy is omitted from the list of systems to which these commands apply, particularly since both SetCoordSystem and GetCoordSystem apply to MoveableMachineCsy.

7. SetProp Return Value
Reference pages: 21 23 26 (37) [56] (60) 63 72
It is not clear what data SetProp should return. The spec says "The format is defined by the method enumerated." This is not very clear, but seems to say that data should be returned, and it should be returned in the same format as in the command, so that, for example, the data response to the command SetProp(Tool.PtMeasPar.Speed(100)) would be:
Tool.PtMeasPar.Speed(100).
Should the returned value be the value in the command, or should it be the value actually set (which might differ from the value in the command).
The example on page 56 does not show any returned data at all. This is another possibility, but then the client will have to call GetProp to find out what value was actually set.

8. SetProp Effect
Reference pages: 21 23 26 (37) [56] (60) 63 72
There are three problems with the effect of SetProp.
A. What happens if a parameter value is commanded to be set out of range?
If the commanded value is not greater than the allowed maximum value and not less than the allowed minimum value, certainly the effect should be to set the property to the commanded value. If the commanded value is out of range, there are at least two reasonable choices: (i) the value is not changed, (ii) the value is changed to be either the maximum or the minimum, whichever is closest to the commanded value. Appendix A implements the second of these (see page 75). The text should say what is intended. A third choice would be to set the value out of range, but that would be a bad choice.
B. What happens if a parameter maximum or minimum is commanded to be
set so that the current value of the parameter becomes out of range?
C. What happens if a parameter maximum or minimum is command to be set so that the maximum is less than the minimum? A useful principle to use in deciding what the effect of SetProp should be is that it should be impossible to create an out-of-range condition, but the value should be set as close to the commanded value as possible. This may be done by initializing every parameter with minimum < value < maximum and then using the following rules:
(i) If a value is commanded to be set out of range, set it to the maximum or the minimum, whichever is closest to the commanded value.

(ii) If a maximum or minimum is commanded to be set so that the current value becomes out of range, set the maximum or minimum to the current value.
Note that rule (ii) combined with correct initialization will also prevent the maximum from becoming less than the minimum.

9. Implicit DisableUser
Reference pages: 21 24 (39) (42) (44) 66
The remark at the end of section 6.3.2.4 says "The server calls this method implicitly whenever the client calls a method that physically moves the machine." Does this mean the client must call EnableUser again to enable the user, or if the user was enabled before the motion, is the user enabled again automatically after the commanded motion is completed? The text should make this clear.

10. FindTool Data
Reference pages: 21 24 26 (44) [56] 67 70 Section 6.3.2.16 says that data sent in response to a FindTool command should be "FoundTool", but the example on page 56 does not show any returned data. Is data returned or not? If so, what is the format of the data?

11. Effect of FindTool
Reference pages: 21 24 26 (44) [56] 67 70 There are two problems with the effect of FindTool.
A. Is FindTool("UnDefTool") a legal command? If so, it should be possible to get the properties of UnDefTool using GetProp(FoundTool...). But UnDefTool, presumably, has no properties so GetProp(FoundTool...) will then return an error. It would seem best to make FindTool("UnDefTool") illegal.
B. Suppose FindTool is called successfully so that there is a FoundTool, and then FindTool is called unsuccessfully. What happens to FoundTool? Does it stay unchanged, or does it become NoTool, or does something else happen?
Note that the comment at the end of Section 6.3.2.17 says that FoundTool can be a pointer to NoTool.

12. AlignTool Data
Reference pages: 21 24 26 (45) 67 72 The data in a response to AlignTool is is supposed to be "vectors". It is not clear how the data is supposed to formatted. If there are two vectors, there are at least two reasonable formats: (i1, ji, ki, i2, j2, k2) or ((i1, ji, ki), (i2, j2, k2)). Which is intended? Having the data look like: AlignToolData(i1, ji, ki, i2, j2, k2) would be nice.

13. Error Messages 0003 and 0004
Error message 0003 is "Illegal flag character at pos. 6" and 0004 is "No space at pos 7". These messages are pointless and should be deleted. There is no flag character in a command and no space at pos 7 (i.e. column 8) in a command. Flag characters are only in responses. Only responses have a space at column 8. The client, presumably, does not send error messages to the server pointing out bad responses.

14. Position number in Error Messages.

Errors 0002, 0003, and 0004 refer to "pos. 5", "pos 6." and "pos. 7", respectively (see page 57). These are called "column 6, column 7, and column 8, respectively, on page 30. It is very confusing to use zero-based indexing in the error messages and one-based indexing in the text. It would be better to use "column X" than "pos X" in the error messages.

15. EnumProp and EnumAllProp
EnumProp Reference pages: 21 23 26 27 (37) [56] 63 72 73 74 75 76
EnumAllPropReference pages: 21 (37) (38)
The descriptions of EnumProp and EnumAllProp are unclear. The difference between the the two is also unclear. The meaning of these commands should be explained better in the spec. One possible interpretation is that EnumProp returns the name of the type of a property, while EnumAllProp returns the names and types of the immediate children of a property. Under this interpretation, if the command, for example, is numProp(Tool.GoToPar()), there is one response message with data as follows: "GoToPar", "Property" If the command is EnumAllProp(Tool.GoToPar()), there are six response messages, with data as follows:
"MinSpeed", "Number"
"Speed", "Number"
"MaxSpeed", "Number"
"MinAccel", "Number"
"Accel", "Number"
"MaxAccel", "Number"
If the commands behave as described above, it is possible to traverse a property tree completely (from the root) by giving a number of EnumProp and EnumAllProp commands.
Is this the correct interpretation of these commands?

16. Effect of Home
Reference pages: 21 24 [31] (39) 58 66
The effect of the Home command is unclear. When command execution is complete, is the current position a known predetermined position (as might be given by configuration data), or may the current position vary after the command is complete? In the second case, the client would have to use a Get command to learn the position. The spec should state specifically what is expected.

17. Error Message Format
In the error message format described on page 57, it is not clear whether the F3 and Text fields are strings (with quotes) or if they are written without quotes. The spec should state explicitly which is the case.

18. Field F3 in Error Messages
In the error message format described on page 57, it says "I++ recommends to serve here the name of the error causing method". A. The spec should state what is required as well as what is recommended. What type of data is required to be in field F3? Assuming F3 must be a string, is the server allowed to put any string it wishes in F3?

B. The phrase "the error causing method" is ambiguous. There are at least two possibilities for what this might mean:
(i) "error causing method" means the name of the DME Interface command that was being executed. If this is the meaning, field F3 provides no information the client does not already know, since the error message includes the tag of the command that caused the error (if it can be identified) and the client will know what command that was.
(ii) "error causing method" means the name of the function in the server implementation that reported the error. If this is the meaning, field F3 is useful primarily in reporting problems to the server vendor.