SMPTE RECOMMENDED PRACTICE

RP 138-1996
Revision of RP 138-1992

Control Message Architecture for Digital Control Interface

1 General

1.1 Scope

This practice defines the architecture of the control message language used within a general-purpose communications channel of an interface system which transports data and control signals between equipment utilized in the production, post-production, and/or transmission of visual and aural information.

It is intended that the language described in this practice be utilized when constructing messages used as part of an overall system, allowing interconnection of programmable and nonprogrammable equipment as required to configure an operational system with a defined function, and to allow rapid reconfiguration of a system to provide more than one defined function utilizing a given group of equipment.

1.1.1 Control message language is composed of vocabulary, syntax, and semantics expressed in terms of tokens, rules, and actions, respectively.

1.1.2 The primary intent of this practice is to define the architecture of the messages to be transmitted within the supervisory protocol of the communications channel for the purpose of controlling equipment by external means. Syntax is the set of rules which shall be applied to the vocabulary (tokens) to construct control messages. (The content of the vocabulary and its semantics, being specific to the type of generic equipment, is defined elsewhere.) This practice, or sections thereof, may be applied to the interconnection of elements within an item of equipment.

1.2 Definitions

For the purpose of this practice, the following definitions shall apply:

virtual machine: A logical device consisting of a single device or a combination of devices that responds in essence or effect as a generic type of equipment; e.g., VTR, video switcher, telecine, etc.

virtual circuit: A transparent, logical communications connection between virtual machines. The communications path, in reality, passes through other levels and is propagated over a physical medium.

2 Message structure

2.1 Architecture

The message architecture described in this practice is prepared broadly on the principals of communications levels. This architecture follows a logical structure and is defined in terms of a virtual machine. Messages are of variable length according to function. Complex functions may be divided into basic functions, transmitted as a sequence of shorter messages for execution in the virtual machine.

2.2 Virtual machine

All messages pertaining to generic types of equipment shall be defined in terms of the virtual machine. Utilization of the virtual machine concept in defining messages provides a message architecture that is independent of machine-specific characteristics.

Approved December 10, 1996

3 Control message classification

3.1 Control messages

Control messages are classified in accordance with figure 1.

3.1.1 Virtual machine messages are used to pass commands and responses between virtual machines. Virtual machine messages are those initiated by a controlling device with responses originating in the controlled device. Receipt of a virtual machine message shall result in a defined action and/or response by the virtual machine.

Virtual machine messages may be subdivided into:

3.1.1.1 Common messages whose coding is reserved to provide for functions of general application; e.g., procedures, reference time functions, and reset.

3.1.1.2 Type-specific messages are applicable to specific generic categories of equipment.

3.1.1.3 User-defined messages implement special functions which are not included in the type-specific message set.

3.1.2 System-service messages are messages other than virtual machine messages.

3.2 Virtual machine message subsets

A separate and distinct subset of virtual machine messages shall be specified for each type of virtual machine (VTR, telecine, audio tape recorder, graphics generator, etc.). Said subset, termed a dialect, shall comprise common messages, type-specific messages, and, optionally, user-defined messages.

3.2.1 Common messages: Resident machine messages which are in all virtual machine dialects but not necessarily operative in all virtual machines, whose coding is reserved to provide for functions of general applications.

3.2.2 Type-specific messages: Virtual machine messages which are defined in virtual machine dialect recommended practices.

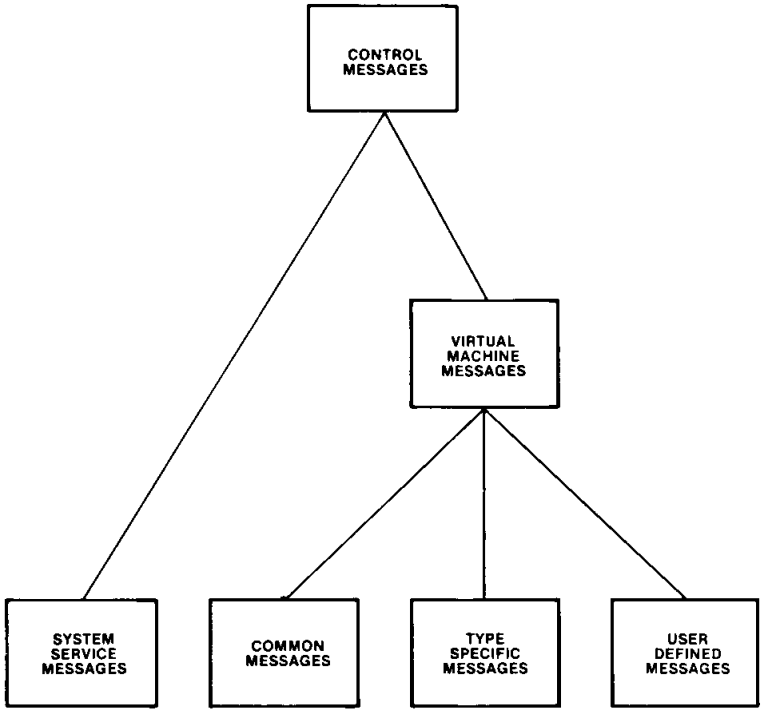


Figure 1 – Message Classification

3.2.3 User-defined messages: Virtual machine messages which are unique to the type (manufacturer, model, version, S/N, etc.) of the specific machine. Although the definition and/or documentation of user-defined messages is considered outside the scope of this practice, the structure of such messages shall conform to the message architecture as defined herein.

4 Control message construction

4.1 Syntax

System service and virtual machine messages are uniformly constructed with the following syntax:

MESSAGE = KEYWORD (+ ARGUMENT)

where the keyword characterizes the function to be performed and the argument contains the parameters, where necessary, to perform that function.

A parameter has the following syntax:

PARAMETER = (NAME +) VALUE(S)

The name may be implied with the use of specific keywords and, in such cases, is therefore not required. The length and format of the value (or values) are defined by the name (or implied name). No restriction is placed on the possible concatenation of parameter values.

4.2 Message formats

All control messages are formed as groups of integral bytes. The first byte of each message shall be the keyword. A keyword specification defines the format of its arguments; although no mathematical relationship is intended between the bit pattern of the keyword and the format. Messages are constructed in one of the following formats:

```
Format 1 Message        =   <Keyword>

Format 2 Message        =   <Keyword> <Parameter List>

        where:
<Parameter List>        =   <Parameter>
        or:
<Parameter List>        =   <Begin> <Parameter Group> <End>

        where:
<Parameter Group>       =   <Parameter>
        or:
<Parameter Group>       =   <Parameter Group> <Parameter>

        where:
<Parameter>             =   <Parameter Value> . . . <Parameter Value>
        or:
<Parameter>             =   <Parameter Name>
                        =   <Parameter Value> . . . <Parameter Value>
```

The appropriate message format can be selected by means of the decision tree given in figure 2.

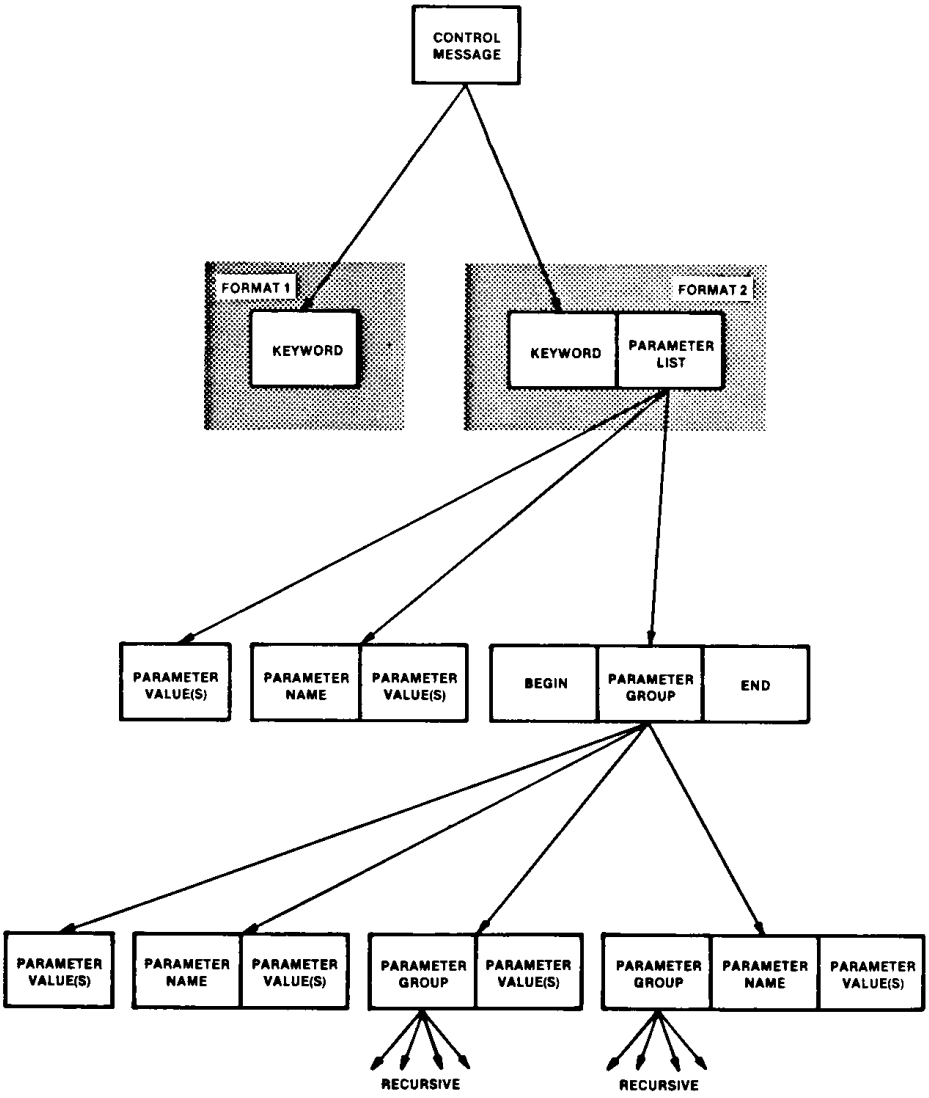


Figure 2 – Decision tree

5 Message coding

5.1 Identical or similar functions on equipment of differing generic types should be effected by the same keyword bit pattern.

5.2 Parameter values

Messages may contain parameters as an essential part. All parameters are classified as follows:

5.2.1 Logical parameter values

Parameters representing any abstract function(s) that may be expressed by a simple binary state of 1 (true) or 0 (false) such as tally on/off or yes/no.

The minimum code length for a single logical parameter is one byte. Individual logical parameters can be assembled, where applicable, into groups to form bit-specific bytes for transmission purposes.

5.2.2 Numerical parameter values

Parameters representing a numerical value and consisting of the following:

Unsigned number parameters:

Parameters representing any numerical value without polarity.

Signed number parameters:

Parameters representing any numerical value with polarity.

5.2.3 Time-code parameter values

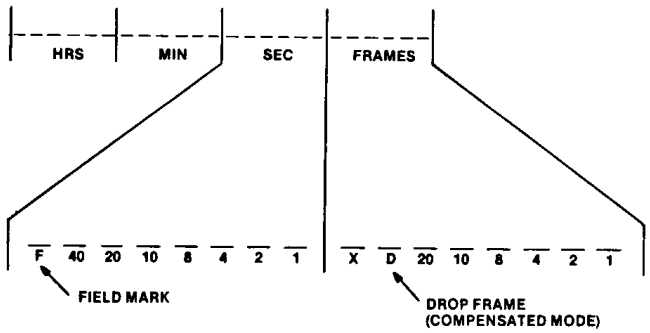
Time is indicated as a four-byte quantity. Parameters representing hours, minutes, seconds,and frames are expressed in BCD in that order. (In BCD,the least significant bit is transmitted first.) The hex “40"-bit of the frame’s byte will be set to one (1) in drop-frame compensated mode. In nondrop frame uncompensated mode and all other time code standards, this bit will be zero (0). In all standards, the hex ”80"-bit of the second’s byte will be set to zero (0) to indicate monochrome field 1 or color fields 1, 3, 5, or 7. This bit set to one (1) will indicate monochrome field 2 or color fields 2, 4, 6, or 8. Unused bits are reserved and are set to zero (0) until defined (see figure 3).

5.2.4 High-resolution time code parameter values

High-resolution time is indicated as a six-byte quantity. The first four bytes are exactly the same as time parameter values. The two remaining bytes express fractions of frames as a 16-bit binary unsigned number (see figure 3).

5.2.5 Literal parameters are parameters based, in general, on ASCII characters.

5.2.6 Raw data parameters are parameters based on a free-form data stream. Raw data parameters must provide for byte transparency to the lower layers. The first byte of a raw data parameter shall be a byte count.



TIME PARAMETER FORMAT

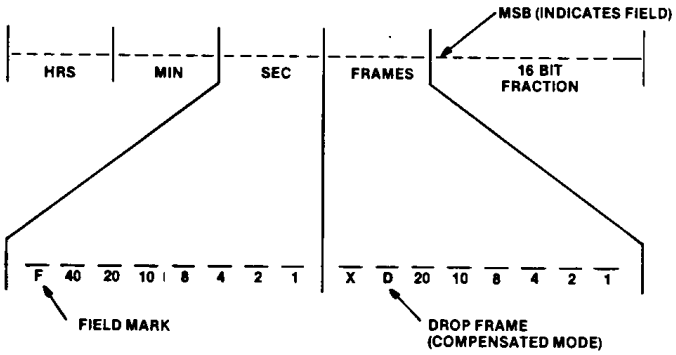


Figure 3 – High-resolution time parameter format