MASSACHUSETTS INSTITUTE OF TECHNOLOGY
HAYSTACK OBSERVATORY
WESTFORD, MASSACHUSETTS 01886

5 October, 1998

Telephone: 978-692-4764
Fax: 781-981-0590

To: Mark 4 Software Group
From: Roger Cappallo
Subject: Mark 4 Message Contents

In their memorandum describing the Mark 4 messaging system, dated 11 November, 1996, John Ball and Kevin Dudevoir delineate the "message delivery agents and message lengths but not (the) message content". This memorandum, along with its subsequent revisions, will define the specific message contents for all of the messages in the Mark 4 correlator system.

General Considerations

In order to maintain flexibility for unanticipated demands, and to allow the possible reception and use of messages from other systems (e.g. EVN correlator software modules), there is no restriction on the format of the contents of a Mark 4 message. Nevertheless, for normal usage within our system, it is suggested that all messages conform to either of two standardized formats. For simplicity, we will call the two formats the "ASCII format" and the "binary format".

1. The **ASCII format** will consist of an ASCII string, properly terminated by the null character. It will be directly manipulable by the C string-handling routines, for simple recognition or parsing.

2. The **binary format** will contain binary words, and possibly ASCII data as well. Its defining structure would look something like:

```
struct Binary_Message
    {
    int msg_code;
    int task_id;
    < declarations for the rest of the message fields >
    };
```

The **first full-word**, *msg_code*, will contain a binary code which identifies the specific message. In practice, the values will be set by loading *msg_code* with a symbolic constant, such as GENERATE_ROOT, that has been enumerated to have a simple integer value. Since the values will be kept low, ASCII messages will be easily distinguishable by the size

of the first word. This could be done, for example, by arithmetically comparing the first word to a symbolic constant that has been pre-loaded with four spaces; iff it is greater, it is a part of an ASCII string.

The **second word**, *task_id*, will be a unique 32-bit reference number, sequentially assigned by conductor when the task is first initiated. This will provide a simple mechanism for keeping track of tasks within the realtime software system, and for identifying the task to which messages refer. The most-recently assigned *task_id* will be kept on hard disk, so that even after a power-failure and restart of the correlator the task numbers will be unique. This will facilitate *post hoc* analysis, for example, of the correlator log.

The **rest of the message** will be formatted in a manner appropriate to the message: different structures can be used to access the rest of the message, as indicated by the value of *msg_code*. In general, there should be no more information contained in a message than is necessary, in order to keep the preparation for sending a message as simple as possible. When there are distinctly different actions to be taken as the result of an error, such as conductor's response to a failed root-generation, a different message should be used (e.g. Root_Complete vs. Root_Incomplete). This practice will result in a cleaner finite state table, in which state transitions will be more visible as table entries, instead of being buried in code that resolves the substate.

Alignment and Endian Considerations

It appears likely that the realtime correlator software system may extend over multiple computers, and perhaps multiple architectures. Since different machines have different constraints on memory accesses, the way in which data is "packed" into a C structure varies from compiler to compiler. Generally there are multiple options within one compiler, but performance may be affected for some ways of accessing the data. Thus, we will define a standard for our messages that will be easy and efficient to implement on most architectures.

The structures in Appendix A should be interpreted as they would be by an HP-UX compiler using the following pragma:

```
#pragma hp_align NATURAL
```

A complete discussion of this option may be found in Chapter 2 of the *HP C Programmer's Guide*. A quick summary is that it forces alignment to boundaries of the same size as the object being allocated. For example, **int**'s are forced to lie on 4-byte boundaries, **double**'s are on 8-byte boundaries, **char**'s on byte boundaries, and so on. Structures are aligned to the largest object defined within them.

All multiple-byte fields are assumed to be stored in big-endian order: that is, the most significant byte has the lowest address (is stored first). This implies that little-endian machines (most notably Intel-based PC's) will have to do a byte-reordering on 2, 4, and 8 byte data, before sending the message across the LAN.

Appendix B: Notes to Appendix A

The definitions for all of the Mark 4 messages are defined by a set of C structures, which are to be found in the include file *message_structs.h*. Additional notes for each of the messages can be found below:

- **Correlate_Scan**
- **Generate_Root**
- **Root_Complete**
- **Root_Failure**
- **Request_CU_Resource**
- **CU_Resource_Avail** This response indicates that the necessary slice(s) is available and has been allocated. It will be held until used by the conductor (via a Configure_Correlator message).
- **CU_Resource_Not_Avail**
- **Configure_Correlator** This command instructs the CU resource manager to perform all necessary input board and correlator board switch assignments to appropriately correlate this scan. There is a response only when the CU is unable to properly configure the correlator, which will be part of the normal unexpected-error reporting path. The sense of the bit map in *valid_channels* is that bit0 says whether or not channel 0 is valid (i.e. active), up through bit15 for channel 15.
- **Request_Tapes** This is a detailed message which requests that all of the tapes for a scan be mounted and configured, and that a good starting time within the scan be found on each tape.
- **Cant_Mount_Tape** This (single) requested tape couldn't be mounted, either because the tape couldn't be found in the database, or there were no appropriate transports available.
- **Valid_Time** As good times are found on each station, this message is sent back to the conductor.
- **Cant_Find_Time** This tape was mounted and examined, but the SU was unable to find times within the current scan.
- **Remove_Station**
- **Have_All_Times**
- **Configure_Tapes**

- **Type_100** Skeletal type 100 record with most elements filled in. The unfilled elements are [Percent_done, Qcode, start, stop, ndrec]. There will be one type 100 record for each baseline within the scan.

- **Type_101** There is one type 101 record for each channel within each baseline; these skeletal type 101 records have most elements filled in. The unfilled elements are [corr_board, corr_slot, post_mortem, block_table].

- **Start_ROT_Clock**

- **Query_Status**

- **Current_Status**