Published: 11/10/67

Identification

The Request Handler
D. B. Wagner, K. J. Martin

Purpose

The Request Handler is a library procedure which performs certain important primitive operations for programs which work with the text of commands and requests. Among the programs which use the Request Handler are the command language macro facility, the debugging command tracer, and the library procedure dispatch_request. These are described in BX.18, BX.10.02, and BY.6.01 respectively, and the reader should refer to those sections for a better view of the context of the Request Handler.

Introduction

The Request Handler is treated as an I/O System pseudo-device. That is, an input stream may be attached to it in the same way that a stream is attached to a typewriter or card reader. For each stream attached to it, the Request Handler keeps two data bases, the Request Queue and the Substitution List. A procedure which wishes to use the Request Handler attaches some stream to it and sets up the two data bases to serve its needs. Subsequent calls to read from that stream cause the Request Handler to be invoked.

The Request Queue is essentially a list of ASCII lines where the Request Handler pseudo-device looks for text when it receives a "read" call. The user may place lines at the head of a Request Queue through direct calls to any of several entries described below. The Request Queue is in fact a push-down list.

The Substitution List is a list of pairs of ASCII strings, each specifying the name of a bound variable and the substitution to be made for it. The Substitution List is manipulated through calls to procedures described below.

When an I/O system "read" call is executed upon a stream attached to the Request Handler, the Request Handler returns the line at the head of the Request Queue for that stream; if the Request Queue is empty it reads a line from a default stream (normally attached to the typewriter). If that

default stream is also empty, this condition is reflected to the I/O system read call, and status is returned to the user indicating what has happened. In either case before returning this line it performs the bound-variable substitutions listed in the Substitution List for the stream.

Attaching to the Request Handler

See Section BF.1.01 for a discussion of attaching. The Request Handler is known to the I/O system as a pseudo-device for which local attachments may be made. (A local attachment is one which applies only to the process in which the attachment is made. A global attachment applies to every process in a process group.) The user issues the following call to attach a stream to the Request Handler:

```
call localattach (ioname,"request_handler", default_stream,
                  "r", status);
```

The I/O system attaches ioname to the pseudo-device "request_handler" in read mode (the only valid mode for the Request Handler). The I/O system calls the Request Handler at the entry request_handler$localattach. Upon receipt of this attach call, the Request Handler sets up an empty Request Queue and Substitution List for this stream if these do not already exist. It also establishes default_stream as the stream to be read when the Request Queue is empty. The I/O system returns status to the user (see BF.1.21 on I/O status).

When the user issues a read call on stream the I/O Switch calls the entry request_handler$read. Any calls to write on stream are rejected.

Manipulation of a Request Queue

ASCII lines may be placed at the head of the Request Queue through any one of several calls (these do not go through the I/O switch):

```
call request_handler$insert_line (stream, line);

call request_handler$insert_thread (stream, first, last);

call request_handler$insert_stream (stream, stream2);

call request_handler$insert_seg (stream, segname);
```

The arguments expected by these entries are declared:

```
dcl stream char(*),
  line char(*)varying,
  (first, last) ptr,
  stream2 char(*),
  segname char(*);
```

Each of these procedures allocates and places what is called a bunch of requests at the head of the Request Queue for the stream indicated. In the case of insert_line the bunch is the single request in line. In the case of insert_thread, first and last point to the beginning and end of a threaded list of requests which constitutes the bunch. Each item on this thread has the form of the structure request_bead:

```
dcl 1 request_bead based(p);
  2 back ptr initial (null);
  2 forth ptr initial (null);
  2 length fixed bin (17),
  2 line char (p->request_bead.length);
```

Insert_stream places an item into the queue which indicates that the stream stream2 is to be read when a request is needed. This "bunch" is then indefinitely long, and depending upon the nature of the stream may never end except through a call to the entry revert_queue as described below.

Insert_seg places an item into the queue which indicates that the ASCII segment named is to be read for requests. In this case the "bunch" is the contents of the segment.

The following calls place an item at the head of a Request Queue which indicates that some action is to be performed by the Request Handler before it accesses any bunch of requests which the item precedes in the queue.

```
call request_handler$insert_command(stream,command);
call request_handler$insert_call(stream,call);
```

The arguments expected are declared,

```
dcl stream char(*),
  command char(*),
  call entry;
```

Command is a command line which will be given to the Shell for execution when it is found at the top of the queue. Call is a procedure which will be called directly by the Request Handler (with no arguments) when it is found at the top of the queue.

The following call causes the item at the head of the Request Queue to be removed, so that the next item in the queue will be the next "seen" in the queue by the Request Handler.

```
call request_handler$revert_queue(stream);
```

Manipulation of a Substitution List

The following calls are used in working with the Substitution List for a stream.

```
call request_handler$push_subst(stream,level);

call request_handler$pop_subst(stream,level);

call request_handler$install_subst(stream,a,b,i);

call request_handler$delete_subst(stream,i);

call request_handler$count_subst(stream,n);

call request_handler$read_subst(stream,a,b,i,deleted);
```

The arguments expected are declared,

```
dcl stream char(*),
  level fixed bin (17),
  (a,b) char(*) varying,
  (i,n) fixed bin (17),
  deleted bit(1);
```

The first two entries manage a push-down list of generations of the Substitution List for a stream. The other entries work only with the current generation of a Substitution List (the one at the top of the push-down list).

Push_subst "pushes down" the Substitution List for the indicated stream and stores into level a level number for the pushed generation. Then if this same level number is given to pop_subst, the push-down list will be popped to that level. The level number mechanism provides some assurance that errors in pushing and popping will not propagate. A call to pop_subst indicating level 0 causes one level only to be popped.

Install_subst adds the pair (a,b) to the current generation of the Substitution List for the stream given, indicating that a is a string which may occur in a request (the name of a "bound variable") and that the string b is to be substituted for it wherever it occurs standing alone in the stream. If any other substitution for a exists in the current generation, it is deleted. A serial number (beginning with 1 and incremented by 1 for each call to install_subst for the same generation of a Substitution List) is stored into i. This serial number may be used in deleting the substitution pair later.

Delete_subst removes the i'th substitution installed in the current generation of the Substitution List for the specified stream. This i'th position is not reused.

Count_subst returns in n the number of calls to install_subst which have been made for the current generation of the Substitution List for the specified stream. Given this count, the calling procedure may use the entry read_subst to find all the "current" substitutions. Note that this count will be larger than the number of substitutions currently in effect if any calls to delete_subst have occurred.

Read_subst returns into a and b the substitution corresponding to the i'th call to install_subst for the current generation of the Substitution List for the specified stream. If this substitution pair has since been deleted using delete_subst, deleted is set to "1"b and a and b are meaningless.

Specifying Special Characters

The Request Handler has an array of special characters which it uses to find free-standing character strings in order to do the appropriate substitutions for these strings. Two of the special characters may be supplied

by the user; if he does not supply them, their functions are not performed. The two characters are a concatenation character and a character to indicate that the Request Handler should not substitute for the free-standing string immediately following.

The following calls supply the concatenation character and the don't-substitute character.

```
call request_handler$concat_char(stream, user_char);

call request_handler$no_subst(stream, user_char);
```

The arguments are declared

```
dcl stream char(*), user_char char(1);
```

The concatenate character concatenates the two free-standing strings immediately on either side of it. The don't-substitute character suppresses substitution for the free-standing string immediately on its right. No other special character may appear between the user-defined special character and the free-standing strings on which the special character is to act.

Reading through the Request Handler

A stream attached to the Request Handler is read through the following I/O system call:

```
call read(stream,elemno,workspace,nelem [,nelemt[,status]]);
```

See BF.1.12 for details of arguments; the Request Handler makes every attempt to act like a console in the element sizes allowed, status return, etc. It should not be necessary for the calling procedure to be aware that the stream may be attached to the Request Handler instead of where it expects. (The user effects an I/O read call by calling read_in - see BY.4.02.)

The Request Handler, upon receipt of such a call as the above, looks at the item at the head of the Request Queue for the specified stream and takes action as follows:

1. If this item specifies a call or command (put there by insert_call or insert_command) it removes the item from the queue, executes the action specified, and looks at the next item in the queue.

2. If on the other hand the item is a non-empty "bunch" (put there by insert_line, insert_thread, insert_stream, or insert_file), the Request Handler picks up a line in the bunch. If _this_ is the last line in the bunch, the bunch is empty; however the bunch will not be removed from the queue until the next time the Request Handler looks in the Queue.

3. If the item at the head of the queue is an empty bunch, the Request Handler removes it from the queue and goes on to the next item in the queue.

4. If the queue itself is empty, the Request Handler reads from the default stream which was indicated in the attach call for the stream specified.

In any case at this point the Request Handler has a command or request line gotten either from the Request Queue or the default stream. It now does bound variable substitution, as follows: A substring of the line is considered to _stand alone_ if it does not contain any special characters but is bounded by special characters. (A special character is any ASCII graphic, including a space, which is not a letter, a digit, or an underscore.) Each stand-alone substring of the line is checked with the current generation of the Substitution List for the stream and any matching substitution is performed (unless the substring is explicitly not to be substituted for). The substituted string is not rescanned.

Finally the Request Handler returns the line it has so laboriously generated as the line read from the pseudo-device.

Implementation

The Request Handler is one procedure segment with umpteen entries (where currently ump = seven). All of the entries use one of the two data bases, the Request Queue and the Substitution List. A copy of each data base exists for each stream which the Request knows about. Each data base is a separate segment and has a unique id (see BY.15.01) as a segment name. In any one process the request handler maintains one data structure to distinguish between the different streams which it may know about. This data structure is declared as:

```
dcl 1 road_sign (50),      /* effectively a maximum of
                              50 streams */

      2 stream char (31),  /* stream name */
```

```
        2 queue ptr,           /* pointer to the appropriate
                                  Request Queue */

        2 subs ptr,            /* pointer to the appropriate
                                  Substitution List */

        2 default_stream char  /* stream name of stream to read
          (31);                   from if the Request Queue is
                                  empty */
```

The Request Queue is declared as:

```
     dcl 1 request_queue based (q_ptr),

        2 current_bunch        /* relative pointer to the current
          bit (18),               bunch */

        2 space area ((131071));
```

Each bunch is allocated into q_ptr->request_queue.space as it is specified by one of the calls to the Request Handler, and freed (by the read or revert_queue entries) when it is no longer needed. A bunch is declared as:

```
     dcl 1 bunch based (b_ptr),

        2 prev_bunch bit (18),

          /* relative pointer to previous bunch in the queue
             (the bunch which will be current when this bunch
             is removed) */

        2 type fixed bin(17),

          /* indicates what type the bunch is:

                 1 = line

                 2 = thread

                 3 = stream

                 4 = segment

                 5 = command

                 6 = call       */
```

```
2 segment_ptr ptr,

   /* points to the segment when type is segment
      (only after the segment is initiated) */

2 (thread_begin, thread_end) ptr,

   /* if type is thread, point to beginning and end
      of threaded list of requests */

2 call_entry bit(216),

   /* if type is call, these six words contain the
      entry as placed there by fake_entry (BY.10.01) */

2 data_length fixed bin (17),

   /* length of following character string */

2 data char (b_ptr→bunch.data_length);

   /* depending on type, one of:

      1)  the line to be read,

      2)  the streamname to read from,

      3)  pathname of the segment to read,

      4)  command line to pass to shell.   */
```

The Substitution List is conceptually, a pushdown stack of individual substitution lists determining whether a string has a substitution and if so, what that substitution is. The Substitution List is hash coded. Since every string "read" by the Request Handler must be checked, a very simple and fast hash coding scheme is desirable. The hash coding scheme used is simply the length of the string, modulo 30. Thus, there are 30 hash buckets, each pointing to the first of a threaded list of substitutions. The threaded lists are arranged such that all substitutions of a given level are grouped together on the thread. The current-level substitutions are first on the thread followed by those of the next-previous level. Since substitutions are made using only the current level, when the Request Handler reaches one of a lower level, it has exhausted all pertinent possibilities. Given a stand-alone substring, the Request Handler determines its length, modulo 30. It then compares the substring against the substitutions in the appropriate hash bucket until either a match is found or a lower level is reached.

The substitutions are also threaded by level to facilitate popping of the substitution list. Figure 1 is a diagram of the Substitution List.

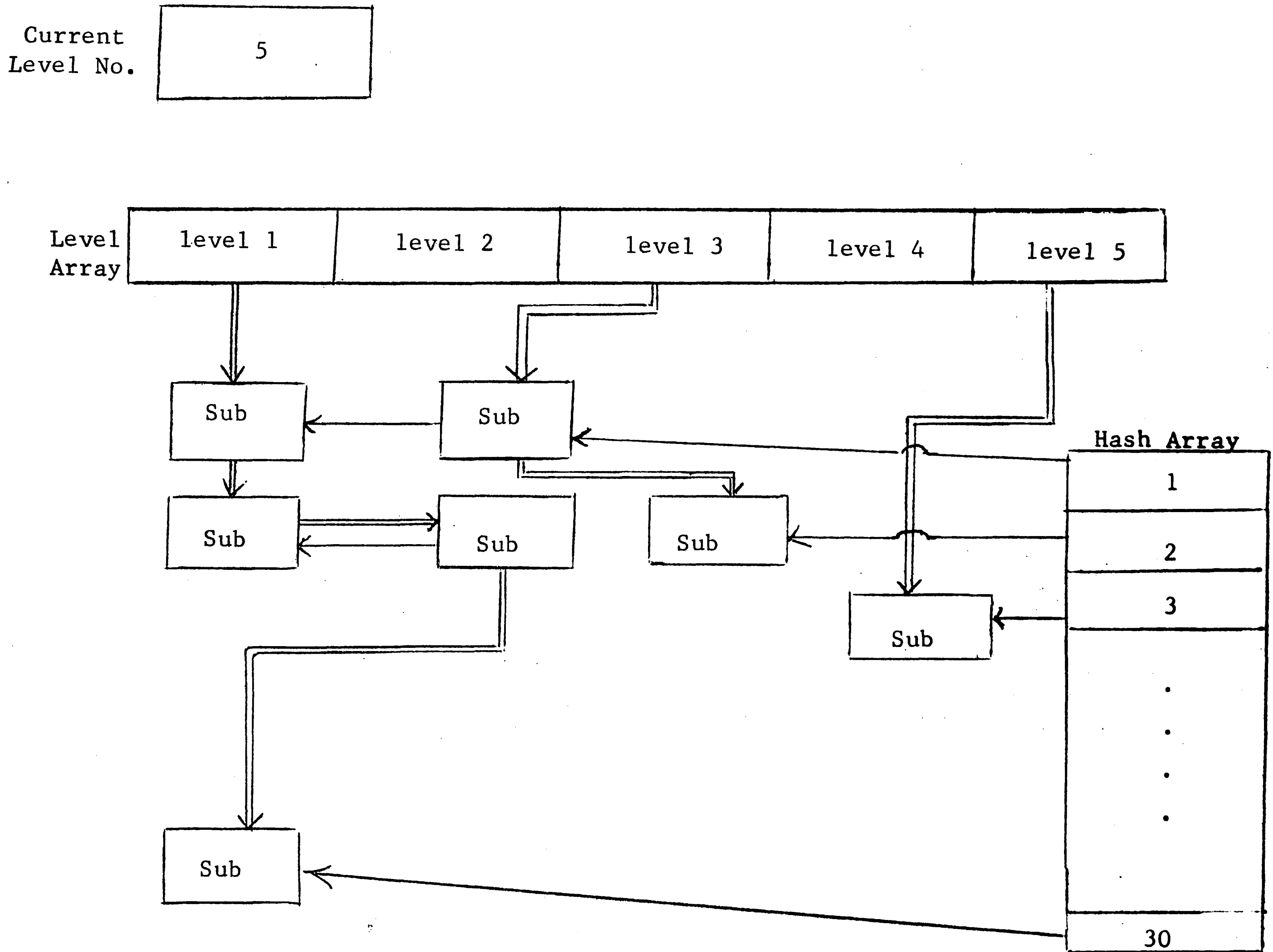

Figure 1:

The current level is 5 which has one substitution with a length of 3 characters (mod 30). Levels 4 and 2 have no substitutions. Level 3 has two substitutions, of lengths 1 and 2 characters (mod 30). Level 1 has 4 substitutions.

There are two substitutions of 1 character (mod 30), three of 2 characters (mod 30), one of 3 characters (mod 30) and one of 30 characters (mod 30).

The Substitution List is declared as:

```
dcl 1 substitutions based (s_ptr),

  2 current_level fixed bin (17),

  2 level_ptr ptr,

     /* points to level_array declared below */

  2 level_size fixed bin (17),

     /* size of level_array */

  2 hash_array (30) ptr,

     /* points to first substitution in each hash
        bucket, as declared below */

  2 space area ((131071));


dcl 1 level_array (n) based (l_ptr),

  2 thread ptr,

     /* points to first substitution on each level,
        as declared below */

  2 no_subs fixed bin (17);

     /* where n = s_ptr->substitutions.level_size
        and l_ptr = s_ptr->substitutions.level_ptr */


dcl 1 sub based (sub_ptr),

     /* pointed to by both an s_ptr->substitutions.hash_
        array pointer and an l_ptr->level_array.thread
        pointer */

  2 (level, hash_length, serial_no) fixed bin (17),

  2 (level_to, level_fro) ptr,

  2 sub_length fixed bin (17),

  2 compare_value char (sub_ptr->sub.hash_length),

  2 sub_value char (sub_ptr->sub.sub_length);
```

Level array is initially allocated in s_ptr->substitutions.space with n = 10. If the level increases to 11, the level array is re-allocated with n = 20. The size of the array is increased by 10 with each necessary increase in size. The sub structure is allocated in s_ptr->substitutions.space for each substitution installed.

When determining whether a substitution is in order for a string the Request Handler first checks the count of the number of substitutions at the current level. This count is in l_ptr->level_array (current_level).no_subs. If this count is 0, no checking for substitutions is done. If there are any substitutions at this level, the Request Handler then determines the length (l) modulo 30 of the string, and checks s_ptr->substitutions.hash_array (l) for a null pointer (which would indicate no substitutions for strings of length (l). If this pointer is non-null, the Request Handler follows it and the hash thread (sub_ptr-> sub.hash_to) until either it ends, the level of a substitution is less than the current level (no substitution to be made) or sub_ptr->sub.compare_value matches the string. If a match is found, the substitution is made.

When a match has been found or eliminated the Request Handler moves on to the next free-standing string in the input line (which was taken from the Request Queue). It is hoped that the design of the Substitution List will make the average search end quickly. It is probable that a relatively small number of substitutions (say, less than 10) will be made at each level. However, every free-standing string of the input line must be processed.