Published 2/28/68

Identification

The Type Table and Type-Table Maintainer
D. A. Levinson

Purpose

The IOSW forwards attach calls to the Not-Founder. The Not-Founder creates the Attach Table entry for the given ioname. Some of the information for the entry is obtained from the arguments of the attach call. Two key items for the entry are obtained from the Type Table (TT) via the Type-Table Maintainer (TTM):

1) the name of the outer module associated with the type,
2) the names of the driving tables associated with the outer module.

When a user (group) issues its first outer call, the Type Table is initialized with default entries for all types known to the I/O System. Calls, described in detail below, are provided for editing the Type Table, changing existing entries, and adding and deleting entries.

The Local Extension of the Type Table

The attach call establishes an ioname for any process of a user group. Thus, this call establishes a global ioname. As described in Section BF.1.00, ionames may be attached locally, that is, such that the ioname is known only to the attaching process. The default Type Table for localattach calls is the global Type Table. The local extension of the Type Table provides the capability of editing the Type Table on a per-process basis. Thus, when an ioname is locally attached, and the Not-Founder references the Type Table, the local Type Table is searched first, for a match on the type specified in the second argument of the localattach call, and only if this search fails, is the global Type Table searched. This is not true of the (global) attach call for which the search is always restricted to the global Type Table. To edit for global attachments one simply edits the global Type Table.

The Type Table

The global Type Table is allocated in the Attach Table segment, and the local extension in the local Attach Table segment. Figure 1 is the declaration of the Type Table with brief comments on the items. Here the items will be described more fully:

1) nextrelp - The Type Table grows dynamically in increments corresponding to an array (subtab in FIgure 1) of entries. The arrays are threaded

together by nextrelp which is a relative pointer to the "next" piece.

2) type_name - This is the name given to the device or pseudo-device type (see Section BF.1.01).
3) module_segment name - This is the segment name of the outer module which services the given type.
4) module_flag - This flag indicates if the module is system-standard for the given type or user supplied, and determines a suitable search algorithm for purposes of linking.
5) name - This is the name of a driving table (at most 3, possibly none) for use by the outer module.
6) flag - indicates if the specified driving table is system -standard or user supplied, and determines search alogorithms according.
7) copy_sw - indicates if a copy of the driving table is to be provided or the original.
8) offset - More than one driving table may be contained in the segment specified by 5). Offset is the offset in words relative to the zero of the segment.
9) override_flag - When a Type Table entry is edited, the entry prior to the edit must be preserved in tact for those ionames that were attached when it was in effect. This is accomplished by setting the override flag to "1"b when an edited entry is to be made. The edited version is used for attach calls subsequent to the edit.

Primitives

```
ttm$change_outer_module(type, table, name, dir, cstatus);

        dcl type char (*),
            table char (1),
            name char (*),
            dir bit (1),
            cstatus bit (18);
```

This call changes the outer_module associated with type in table(= "1" if local, "g" if global) to name. If dir is "0"b the I/O System's directory is searched for name, else if dir is "1"b then the user's working directory is searched for name. If an entry does not already exist for type this call will create it. Finally, cstatus is set to "0"b if an entry already existed, "1"b if not, and "01"b if an error is encountered.

```
ttm$change_dtab(type, table, dtabn, name, dir, copy_sw, offset, cstatus);

        dcl type char (*),
            table char (1),
            dtabn fixed,
            name char (*),
```

```
        dir bit (1),
        copy_sw bit (1),
        offset fixed,
        cstatus bit (18);
```

This call (not to be confused with atm$change_dtab, see BF.2.13) is used to change driving table n (= 1, 2 or 3) associated with type in table (= "1" if local, "g" if global) to name in dir ("0"b = IOS directory, "1"b = user's working directory). The copy_sw if "1"b specifies a copy of the segment name, if "0"b the original. Offset is the offset of the driving table, in words, from the zero of segment name. Finally, cstatus is "0"b if the call is successfully executed, "1"b if an error is encountered.

```
ttm$delete_type(type, table, cstatus);

dcl type char (*),
    table char (1),
    cstatus bit (18);
```

This call deletes the entry for type in table ("1" if local, "g" if global). If an error is encountered cstatus is set to "1"b else to "0"b.

```
dcl 1 tt based (p), /*type table*/
 2 nextrelp bit (18),          /*relp to next subtable*/
 2 subtab (20),                /*subtable array*/
  3 type_name char (32),       /*type name*/
  3 module_name char (32),     /*outer module name*/
  3 module_flag bit (1),       /*1=user set,0=system default*/
  3 dtabs (3),
   4 name char (32),           /*driving table name*/
   4 flag bit (1),             /*1=user set,0=system default*/
   4 copy_sw bit (1),          /*1 = copy of dtab, 0=original*/
   offset fixed,               /*word offset of dtab*/
  3 override_flag bit(1);      /*1 = not active, 0 = active TT entry*/
```

Figure 1 - Type Table Declaration