ISS 45

ICL

DRVFILE Technical Reference

Version 7.7

ISS45 7.7 DRVFILE Technical Reference

Date of Issue	Product Identification Number	Part Number	Brief Description
April 1995	45001/011	80316000	Version 7.0
August 1995	45001/011	80316778	Version 7.1
October 1996	45001/011	80328029	Version 7.3
February 1999	45001/011	Electronic Library 80602986	Version 7.6
August 2000	45001/011	89000057	Version 7.7 (Unchanged)

Address comments and corrections to:

ICL Retail Systems
ISS45 Program Director
2933 Bunker Hill Lane
Suite 101
Santa Clara, CA 95054

This documentation is designed for placement in an ICL binder that can be ordered separately. To order the binder, contact your sales representative. Indicate PIN 45007/002 and/or part number 80192817.

This sheet contains spine cards that can be used to identify the binder for the appropriate documentation. Cut one of the cards along the dotted lines and insert it in the binder's spine pocket. Discard the remaining cards or save them for later use.

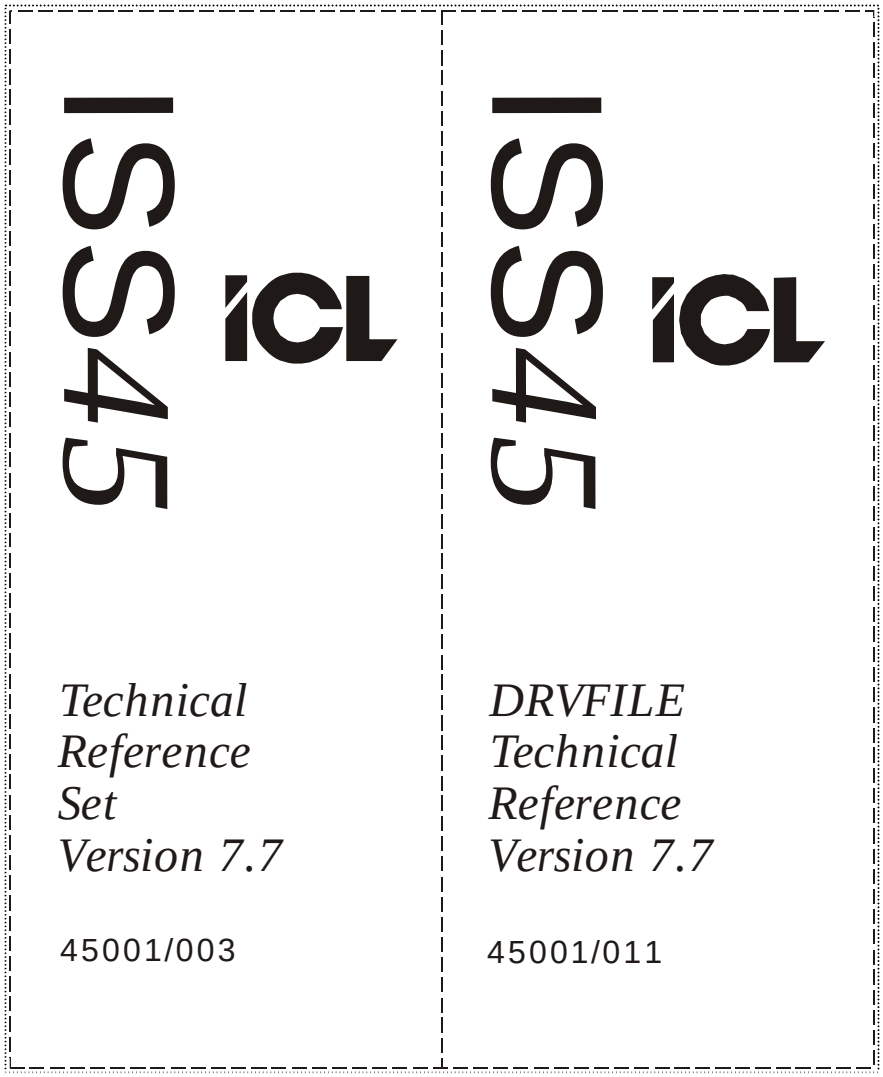

Table of Contents

Introduction

DRVFILE is the database management of the system. Its object is to create a unified interface between all applications and the shared files on the systems. Since this system is based on a networking system, the files that DRVFILE handle can be local or remote.

In addition, DRVFILE allows you to develop MACROS that perform in the background under other MS-DOS applications. This facility allows the developer to develop code that operates as a POS server without dedicating the PC to this job.

DRVFILE currently contains built in functions that can be used by the developer like GPFI (General Purpose File Interface) and POS load.

Accessing files through DRVFILE and creating a single point of interface, results in the following advantages:

- All applications access files in a predefined method. The actual interface is resident in a driver, reducing the amount of code of the application. If there is a 'bug' in the access method or disk/LAN handling, there is only one layer to fix without having the need to repair all the application software.
- Accessing information through the DRVFILE allows different configurations without having to change any of the software. For example: The data may be on the local PC disk or in a remote PC. The application software does not have to be aware of this fact and it always receives the data in a unified way.
- All the redundancy logic is held in the DRVFILE level, simplifying the application development. For example: The application writes the data to the database and, depending on the system setup, DRVFILE actually writes the data to the desired disks.
- Using DRVFILE creates a user interface to all other applications, and for developers that want to access or enhance the system. This is a form of data protection, protecting the system from 'PC hackers' changing data, damaging indexes or not using the correct logic.

- DRVFILE allows the system to grow, change its file layouts without the need to upgrade previous applications to new formats. Also, if the system is to be ported to different operating systems or equipment, there is only the need to update the low level disk access methods or LAN interface, to upgrade the total system.

  Mirroring and full redundancy: Using DRVFILE allows the possibility of mirroring data to several different disks on the system without having the application software involved in this process. This simplifies development and makes the system setup very easy and adaptable to the actual hardware available.

DRVFILE Calls

The calls to DRVFILE are separated into two different types:

- General calls
- Specific calls

General Calls

The general calls enable the user to access any QuickDex or DOS file within the network. The location and destination of the reads and writes to these files are configured in the DRVFILE setup.

This setup maintains 64 x 5 bit flags per file or Macro. Each flag represents the following information:

- **Read LAN**: The Data is not available on this disk and must be read from the server via the LAN. This bit is set on a diskless PC acting as a workstation.

- **Write LAN**: The Data must be updated to the server via the LAN. This bit is set on all stations including the Back Office.

- **Write Local Disk**: The data must be written to the local disk. This bit is set on all PCs that contain a Backup file (Back Office).

- **Redundant Backup**: The data must be written to the backup disk (mirror to Back Office). This bit is set in the PC holding the Master files (MFS1 or MFS2).

- **Send Mask Redundant**: This bit is sent as part of the parameters in the 'Write LAN', indicating to the server not to perform a Redundant Backup (the backup is performed by the sender). This bit is set in the Back Office PC.

Specific Calls

Specific calls are developed by the individual developer. These calls are considered as MACROS.

DRVFILE Parameters

The parameters of DRVFILE are read from the FILE.PRM file. The format is obtained later. It includes redirection tables for QuickDex files, etc.

Table entry format

Each table entry is formatted:

- **Read from LAN**: PC# and link type 0 means read from local disk. Link type may be STARLAN or CFS-LAN or RS232.
- **Write LAN**: PC# and link type.
- **Write Local Disk**: Yes / No.
- **Backup LAN**: PC# and link type.
- **MASK Backup LAN**: Yes /No. Yes means that when writing to LAN, the receiving PC will not backup the originator PC, even if the receiving PC has it's Backup LAN bit set.

Setup settings for various workstations

```
Possible Configuration:

========================

PC1                PC2
    standard lan
```

All the POSTs are also connected to the standard LAN.

Example:

POS-PLU file map settings (reside on every PC - 33 byte len).

	PC1-MFS1	PC2-MFS2
Read LAN	No	No
Write LAN	No	1
Write Local disk	Yes	Yes
Write backup	2	No
Mask backup	No	No

DRVFILE General Calls

PCHK

This application checks if DRVFILE is loaded.

Call from BASIC:

call pchk(ret%)

Call from C:

int ret = pchk();

Return code 0 means DRVFILE loaded into memory.

All DRVFILE calls are in this format:

call drvfile_function_name (parameters,buffer)

Where:

parameters is a structure of 30 bytes, and

buffer is a variable length record, up to 1024 bytes length.

DRVFILE reads parameters when it loads from FILE.PRM. However, DRVFILE parameters can be read or updated in run time without the need to reload, by calling these functions:

PGETPARMS

This reads setup parameters. DRVFILE returns a string that contains its parameters and formatted the same as FILE.PRM

Call from BASIC:

call pgetparms(f.param$,f.setup.parameters,ret%)

Call from C:

int ret = pgetparms(char *f_parm,char * f_setup_parameters);

Return code 0 means OK.

PSETPARMS

Updates DRVFILE setup parameters. DRVFILE reads a string that contains the new parameters and is formatted the same as FILE.PRM. Its parameters are updated accordingly.

Call from BASIC:

```
call psetparms(f.dummy$,f.parameters,ret%)
```

Call from C:

```
int ret = psetparms(char *dummy,char * f.parameters)
```

Return code 0 means OK.

DRVFILE - PLU File Interface

This section has been split into sections, due to support for the different versions
(7.3, 7.1, 6.4 and 6.3). New programs should use the new 7.3 API calls. However, DRVFILE supports the old calls (versions 7.1, 6.4 and 6.3) for backwards compatibility, i.e: old programs need not be changed in order to run on version 7.3.

A special interface for PLU files is provided through the DRVFILE extensions library. The library supports calls for PLU inquiry and maintenance. This interface is the preferred PLU interface

7.3 PLU Record Structure

```
struct  rlt_plu_rec

{

   unsigned char plu_number[7];                BCD     PLU number

   unsigned char dep[2];                       BCD     Department number

   unsigned char name[40];                     ASCII   ASCII name

   unsigned char name_pos[20];                 ASCII   POS name

   unsigned char tax;                          BITS

                                                         Bit 0- TAX rate
1

                                                         Bit 1- TAX rate
2

                                                         Bit 2- TAX rate
3

                                                         Bit 3- TAX rate
4

                                                         Bit 4- TAX rate
5

                                                         Bit 5- TAX rate
6

                                                         Bit 6- TAX rate
7

                                                         Bit 7- TAX rate
8

   struct                                      BITS

   {

      char pay_credit_foodstamp  :1;                     Payment by food
stamps

      char non_merch             :1;                     Non merchandise

      char decimal_qty_req       :1;                     Decimal
quantity

      char negative_entry        :1;                     Negative entry
```

```
      char store_coupon          :1;                    Store coupon

      char vendor_coupon         :1;                    Vendor coupon

      char wic_item              :1;                    WIC item

      char persistence           :1;                    Not Used

   } item_flags1;


   struct                                        BITS

   {

      char promotional           :1;                    Item on
promotion

      char not_for_sale          :1;                    Sale prohibit

      char default_price         :1;                    Not used

      char manual_price          :1;                    Manual price

      char force_qty             :1;                    Force quantity

      char weighed_item          :1;                    Weighed item

      char inhibit_qty_rpt       :1;                    Inhibit repeat

      char non_discount          :1;                    Discount
prohibit

   } item_flags2;


   struct

   {

      char second_price          :1;                    Not Used

      char disalow_bonus_c       :1;                    Save for later
use

      char cost_plus             :1;                    Cost plus item

      char price_verify          :1;                    Price verify

      char price_ovverride       :1;                    Price override

      char supplier_prom         :1;                    Supplier
promotion
```

```
      char save_disc_flag        :1;                    Save discount
flag

      char item_on_sale          :1;                    Item on sale

   } item_flags3;

   unsigned long price;                          AMT_D   Item price

   unsigned int  date_price;                     DATE_I  Price change date

   long center_price;                            NUM_D   Center price

   unsigned int  date_ctr_price;                 DATE_I  Center price
change date

   unsigned char cost_per_case_price[4];         BCD_D   Cost per case

   unsigned int  date_cost_price;                DATE_I  Cost per case
change date

   unsigned char unit_per_case[2];               BCD     Unit per case

   unsigned char msu_c;                          NUM     multiply sell
unit

   unsigned char mm_c;                           NUM     Mix & match

   unsigned char return_no;                      NUM     return type

   unsigned char mm_flags;                       BITS

                                                         Bit 0: save for
internal

                                                               number

                                                         Bit 1:
continuation record

                                                               of
internal

                                                         Bit 2: reserved

                                                         Bit 3: no
internal

                                                         Bit 4: reserved

                                                         Bit 5:
continuation record

                                                                of m&m
```

```
                                                        Bit 6: return
to user with
                                                               OK_MORE
                                                        Bit 7: no mix
match
   unsigned char subdept[6];                  BCD     Subdepartment
   unsigned char family_num[2];               BCD     Family number
   unsigned char user_defined[13];            UNDEF   User defind
fields
   unsigned long weight;                      NUM_D   Weight
   unsigned char place[3];                    NUM
row/column/shelf+label#
                                                         1b   1b     «b
«b
   unsigned char disc_num;                    NUM     Linked discount
   unsigned long second_price;                NUM_D   Frequent shopper
discount
   unsigned char tare              ;          NUM     Tare weight
   unsigned char  inter_num[7];               BCD     Internal number
   unsigned char  label_count;                NUM     Label count
   struct                                     BITS
   {
      unsigned char  scale             :1;             Send to scale
      unsigned char  shelf_stock       :1;             Maintain shelf
stock
      unsigned char  local_del         :1;             Marked for
local deletion
      unsigned char  host_del          :1;             Marked for host
deletion
      unsigned char  head_office_dep   :1;             Report to head
office
      unsigned char  weighed_scale     :1;             Weighed at
scale
      unsigned char  nu                :2;             Not used
```

```
   } bo_flags1;


   unsigned char freq_shop_type;              NUM     Frequent shopper
type

   unsigned      second_family;               NUM     Second family

   long          save_price;                  NUM_D   Save price


   unsigned char  pos_message;                NUM     Linked message

   unsigned char  nu_pos_messgage;            NU      Not used

   unsigned char  hierarchy_code[5];          BCD     Hierarchy code

   unsigned int   shelf_life;                 NUM     Shelf life (in
days)

   unsigned int   promo_no;                   NUM     Promotion number

   unsigned char  bucket_no;                  NUM     Bucket number

   unsigned int   extended_promo_no;          NUM     Extended
promotion

   unsigned char  extended_bucket_no;         NUM     Extended bucket

   unsigned char  coupon_no[7];               BCD     Coupon number



   unsigned char  info_nu[92];                NU      Not used


   // *** END OF INFORMATIVE PART ***


   // *** START OF SALES PART ***


   unsigned char  sale_nu[89];                NU      Not used


   long day_promo_revenue;                    NUM_D   Daily revenue
when promoted
```

```
   int  day_promo_qty;                      NUM     Daily quantity
when promoted

   unsigned int  day_date_sales;            DATE_I  Date of last sale


   long day_qty_sales;                      NUM     Daily sales

   long day_revenue;                        NUM_D   Daily revenue

   long day_discount;                       NUM_D   Daily discount

   long day_promotion;                      NUM_D   Daily promotion
revenue

   long day_qty_promotion;                  NUM     Daily promotion
count


   long p_day_qty_sales;                    NUM     Previous day
quantity sold

   long p_day_revenue;                      NUM_D   Previous day
revenue

   long p_day_discount;                     NUM_D   Previous day
discount


   long week_qty_sales;                     NUM     Weekly quantity
sold

   long week_revenue;                       NUM_D   Weekly revenue

   long week_discount;                      NUM_D   Weekly discount


   long p_week_qty_sales;                   NUM     Previous week
quantity sold

   long p_week_revenue;                     NUM_D   Previous week
revenue

   long p_week_discount;                    NUM_D   Previous week
discount


   long month_qty_sales;                    NUM     Monthly quantity
sold

   long month_revenue;                      NUM_D   Monthly revenue
```

long month_discount;	NUM_D	Monthly discount
long p_month_qty_sales;	NUM	Previous month quantity sold
long p_month_revenue;	NUM_D	Previous month revenue
long p_month_discount;	NUM_D	Previous month discount
long year_qty_sales;	NUM	Yearly quantity sold
long year_revenue;	NUM_D	Yearly revenue
long year_discount;	NUM_D	Yearly discount
long z_qty_sales;	NUM	Quantity sold since last Z
long z_revenue;	NUM_D	Revenue since last Z
long z_discount;	NUM_D	Discount since last Z
long z_promotion;	NUM_D	Promotion since last Z
long z_qty_promotion;	NUM	Promotion quantity since last Z
long pccs_save_qty_sales;	NUM	PCCS save quantity sold
long pccs_save_revenue;	NUM_D	PCCS save revenue
int day_price_override;	NUM	Daily price overrides
int week_price_override;	NUM	Weekly price overrides
long day_reduce_qty_prom;	NUM	Daily reduction quantity

```
   long day_sale_item;                          NUM_D  Daily item on
sale revenue

   long day_qty_sale_item;                      NUM    Daily item on
sale quantity

   long period_sale_item;                       NUM_D  Period item on
sale revenue

   long period_qty_sale_item;                   NUM    Period item on
sale quantity

   long period_markdown_sale;                   NUM_D  Period item on
sale markdown

   struct  maint_up_                            STRUCT Update time stamp

   {

      unsigned char hi_date;                             High part of
update date

      unsigned long pc_created :1;                       PC number
generated

      unsigned long time       :23;                      Update time

      unsigned long low_date   :8;                       Low part of
update date

   } time_stamp;



   unsigned char q_dex_flags;                   BITS   QDEX reserved
flag

};
```

7.3 Parameter Structure

```
struct f_parm_

{

   unsigned indx_num;    /* 0=plu,5=item,6=dep  index    */

   unsigned option;      /* bit0: 1= Do not create POS maintenance
records */

                         /* bit1: Must be = 0            */

                         /* bit2: Must be = 1 for V7.3   */

                         /* bit3: 1=multi user           */

                         /* bit4: 1=read relative+index  */

                         /* bit5: reserved, set to 0     */

                         /* bit6: reserved, set to 0     */

                         /* bit7: reserved, set to 0     */

   unsigned hi_word_write_part_offset;

   unsigned write_part_offset; /* offset of string in partial write */

   unsigned write_part_length; /* length of string in partial write */

   char filler[20];

}
```

7.1 PLU Record Structure

```
struct  rlt_plu_rec_v7           /* OFFSET | TYPE                    */
{                                /* ------------------------------- */
   // RELATIVE PART
   unsigned char plu_number[7];  /* 0 - 6    13 digits BCD          */
   unsigned char dep[2];         /* 7 - 8    0 - 999                */
   unsigned char name[40];       /* 9 -48    ASCII name             */
   unsigned char name_pos[20];   /* 49-68                           */
   unsigned char tax;            /* 69       byte :                 */
                                 /*          bit 7- TAX rate 8      */
                                 /*          bit 6- TAX rate 7      */
                                 /*          bit 5- TAX rate 6      */
                                 /*          bit 4- TAX rate 5      */
                                 /*          bit 3- TAX rate 4      */
                                 /*          bit 2- TAX rate 3      */
                                 /*          bit 1- TAX rate 2      */
                                 /*          bit 0- TAX rate 1      */
   unsigned char item_flags1;    /* 70       byte                   */
                                 /*          bit 7- payment by WIC  */
                                 /*          bit 6- assign EAN IREF */
                                 /*          bit 5- vendor coupon   */
                                 /*          bit 4- store coupon    */
                                 /*          bit 3- negative entry  */
                                 /*          bit 2- RESERVED ******* */
                                 /*          bit 1- non-merchandize */
                                 /*          bit 0- pay
credit/foodstamp */
```

```
unsigned char item_flags2;     /* 71       byte                    */
                               /*          bit 7- Non-discount     */
                               /*          bit 6- Inhibit Qty/Rpt  */
                               /*          bit 5- weighed item     */
                               /*          bit 4- Fore Qty         */
                               /*          bit 3- Manual price     */
                               /*          bit 2- Default price    */
                               /*          bit 1- Not for sale     */
                               /*          bit 0- promotional      */
unsigned char item_flags3;     /* 72       byte                    */
                               /*          bit 7-  item sale       */
                               /*          bit 6-                  */
                               /*          bit 5-                  */
                               /*          bit 4-                  */
                               /*          bit 3-                  */
                               /*          bit 2-                  */
                               /*          bit 1-                  */
                               /*          bit 0-                  */
long price;                    /* 73-76                            */
unsigned int  date_price;      /* 77-78    bits:yyyyyyymmmmddddd   */
                               /*          ,year 0 is 1980         */
long center_price;             /* 79-82                            */
unsigned int  date_ctr_price;  /* 83-84    bits:yyyyyyymmmmddddd   */
long cost_per_case_price;      /* 85-88                            */
unsigned int  date_cost_price; /* 89-90    bits:yyyyyyymmmmddddd   */
unsigned int  unit_per_case;   /* 91-92                            */
unsigned char msu_c;           /* 93       multiply sell unit 0-99H*/
unsigned char mm_c;            /* 94       mix & match             */
```

```
   unsigned char return_no;       /* 95       return type or tare weight
*/
   unsigned char mm_flags;        /* 96       bit 7: no mix match
*/
                                  /*          bit 6: return to user with
OK_MORE      */
                                  /*          bit 5: continuation record of
mix match */
   unsigned char subdept[6];      /* 97-102   12 digits BCD           */
   unsigned char family_num[2];   /* 103-104   4 digits BCD           */
   unsigned char user_defined[13];/* 105-117   13 bytes               */
   unsigned long weight;          /* 118-121   weight in grams        */
   unsigned char place[3];        /* 122-124   row/colomn/shelf+label#*/
                                  /*           1b   1b     «b    «b   */
   unsigned char s_money_ptr;     /* 125                              */

   unsigned long second_price;    /* 126- 129 reversed binary long    */
   unsigned char second_price_flags; /* 130                           */

   /* FILLER FOR FUTURE USE        */
   char filler[2];                /* 131-132  Filler                  */

   /* sales fields */
   unsigned int  day_date_sales;  /* 133-134  bits:yyyyyyymmmmddddd   */
   long day_qty_sales;            /* 135-138                          */
   long day_revenue;              /* 139-142                          */
   long day_discount;             /* 143-146  discount     revenue    */
   long day_qty_promotion;        /* 147-150                          */
   long day_promotion;            /* 151-154  promotion    revenue    */
   long p_day_qty_sales;          /* 155-158  previous day            */
```

```
long p_day_revenue;            /* 159-162                          */
long p_day_discount;           /* 163-166  discount day revenue    */
long week_qty_sales;           /* 167-170                          */
long week_revenue;             /* 171-174                          */
long week_discount;            /* 175-178                          */
long p_week_qty_sales;         /* 179-182                          */
long p_week_revenue;           /* 183-186                          */
long p_week_discount;          /* 187-190                          */
long month_qty_sales;          /* 191-194                          */
long month_revenue;            /* 195-198                          */
long month_discount;           /* 199-202                          */
long p_month_qty_sales;        /* 203-206                          */
long p_month_revenue;          /* 207-210                          */
long p_month_discount;         /* 211-214                          */
long year_qty_sales;           /* 215-218                          */
long year_revenue;             /* 219-222                          */
long year_discount;            /* 223-226                          */
long z_qty_sales;              /* 227-230                          */
long z_revenue;                /* 231-234                          */
long z_discount;               /* 235-238                          */
long z_qty_promotion;          /* 239-242                          */
long z_promotion;              /* 243-246                          */
long pccs_save_qty_sales;      /* 247-250                          */
long pccs_save_revenue;        /* 251-254                          */

// new fields for 6.4
unsigned char not_used_bt[16]; /* 255      reserved for future BT  */
long day_reduce_qty_prom;      /* 271-274  sale item qauntity      */
```

```
   long day_sale_item;            /* 275-278  sale item revenue       */
   long day_qty_sale_item;        /* 279-282  sale item qauntity      */
   long period_sale_item;         /* 283-286                          */
   long period_qty_sale_item;     /* 287-290                          */
   long period_markdown_sale;     /* 291-294                          */
   long save_price;               /* 295-298                          */

   unsigned char not_used[31];    /* 299-329                          */

   struct  maint_up_
   {
      unsigned char hi_date;
      unsigned long pc_created :1;
      unsigned long time       :23;
      unsigned long low_date   :8;
   }last_up;   /*  330-334 last up date time + pc created */

   unsigned char q_dex_flags;     /* 335      byte for QuickDex       */
}plu7;
```

7.1 Parameter Structure

```
struct f_parm_

{

   unsigned indx_num;    /* 0=plu,5=item,6=dep  index    */

   unsigned option;      /* bit0: 1= Do not create POS maintenance
records */

                         /* bit1: 1= V7.1                */

                         /* bit2: reserved, set to 0     */

                         /* bit3: 1=multi user           */

                         /* bit4: 1=read relative+index  */

                         /* bit5: reserved, set to 0     */

                         /* bit6: reserved, set to 0     */

                         /* bit7: reserved, set to 0     */

   unsigned hi_word_write_part_offset;

   unsigned write_part_offset; /* offset of string in partial write */

   unsigned write_part_length; /* length of string in partial write */

   char filler[20];

}
```

7.1 Inquiry PLU Calls

- **Enhanced read matching item ('READ') - EPLUREAD**

  Read ('get') the matching item. The plu-record must include the PLU as a key before the call. After the call, if the return code is OK, the plu-record is filled with the details of the item which is equal to the requested PLU.

  ```
  rtc = epluread(&f_parm,&plu);
  ```

- **Enhanced read matching or next item ('START') - EPLUSTART**

  Read ('get') the matching or next item. The plu-record must include the PLU as a key before the call. After the call, if the return code is OK, the plu-record is filled with the details of the item which is equal to, or the next following of the requested PLU.

  ```
  rtc = eplustart(&f_parm,&plu);
  ```

- **Enhanced read next item ('NEXT') - EPLUREADNEXT**

  Read ('get') the next item. The plu-record must include the current PLU as a key before the call. After the call, if the return code is OK, the plu-record is filled with the details of the item which is the next following of the requested PLU.

  ```
  rtc = eplureadnext(&f_parm,&plu);
  ```

- **Enhanced read previous item ('PREV') - EPLUREADPREV**

  Read ('get') the previous item. The plu-record must include the current PLU as a key before the call. After the call, if the return code is OK, the plu-record is filled with the details of the item which is the previous before the requested PLU

  ```
  rtc = eplureadprev(&f_parm,&plu);
  ```

- **Enhanced read last item - EPLUREADLAST**

  Read ('get') the last item. The plu-record must include the current PLU as a key before the call. After the call, if the return code is OK, the plu-record is filled with the details of the item which is the last PLU.

  ```
  rtc = eplureadlast(&f_parm,&plu);
  ```

7.1 Maintenance PLU Calls

Maintenance PLU calls are organized in a batch mode. Before any update, a routine for starting the batch must be called, then a batch of updates is built. Only the PLU file in the PC is updated in this phase, and afterwards a routine for executing the batch is called. The POSTs are then updated.

- **Enhanced update item ('UPDATE') - EPLUUPDATE**

  Update the matching item. The plu-record must be filled with all the fields before the call. After the call, if the return code is OK, the PC PLU file is updated and an update PLU record is added to the maintenance batch.

  ```
  rtc = epluupdate(&f_parm,&plu);
  ```

- **Enhanced delete item ('DELETE') - EPLUDELETE**

  Delete the matching item. The plu-record must only be filled by the PLU number field before the call. After the call, if the return code is OK, the PC PLU file is updated and a delete PLU record is added to the maintenance batch.

  ```
  rtc = epludelete(&f_parm,&plu);
  ```

- **Enhanced insert item ('INSERT') - EPLUINSERT**

  Insert a new item. The plu-record must be filled with all the fields before the call. After the call, if the return code is OK, the PC PLU file is updated.

  ```
  rtc = epluinsert(&f_parm,&plu);
  ```

6.4 PLU Record Structure

```
   struct  rlt_plu_rec_          /* OFFSET | TYPE                          */
{                                /* --------------------------------      */
   // RELATIVE PART
   unsigned char plu_number[7]; /* 0 - 6    13 digits BCD                 */
   unsigned char dep[2];         /* 7 - 8    0 - 999                       */
   unsigned char name[40];       /* 9 -48    ASCII name                    */
   unsigned char name_pos[16];   /* 49-64                                  */
   unsigned char tax;            /* 65       byte :                        */
                                 /*          bit 7- TAX rate 8             */
                                 /*          bit 6- TAX rate 7             */
                                 /*          bit 5- TAX rate 6             */
                                 /*          bit 4- TAX rate 5             */
                                 /*          bit 3- TAX rate 4             */
                                 /*          bit 2- TAX rate 3             */
                                 /*          bit 1- TAX rate 2             */
                                 /*          bit 0- TAX rate 1             */
   unsigned char item_flags1;    /* 66       byte                          */
                                 /*          bit 7- payment by WIC         */
                                 /*          bit 6- assign EAN IREF        */
                                 /*          bit 5- vendor coupon          */
                                 /*          bit 4- store coupon           */
                                 /*          bit 3- negative entry         */
                                 /*          bit 2- RESERVED *******       */
                                 /*          bit 1- non-merchandise        */
                                 /*          bit 0- pay credit/foodstamp   */
   unsigned char item_flags2;    /* 67       byte                          */
```

```
                                /*          bit 7- Non-discount         */
                                /*          bit 6- Inhibit Qty/Rpt      */
                                /*          bit 5- weighed item         */
                                /*          bit 4- Fore Qty             */
                                /*          bit 3- Manual price         */
                                /*          bit 2- Default price        */
                                /*          bit 1- Not for sale         */
                                /*          bit 0- promotional          */
unsigned char item_flags3;      /* 68       byte                        */
                                /*          bit 7-  item sale           */
                                /*          bit 6-                      */
                                /*          bit 5-                      */
                                /*          bit 4-                      */
                                /*          bit 3-                      */
                                /*          bit 2-                      */
                                /*          bit 1-                      */
                                /*          bit 0-                      */
long price;                     /* 69-72                                */
unsigned int  date_price;       /* 73-74    bits:yyyyyyymmmmddddd       */
                                /*          ,year 0 is 1980             */
long center_price;              /* 75-78                                */
unsigned int  date_ctr_price;/* 79-80       bits:yyyyyyymmmmddddd       */
long cost_per_case_price;       /* 81-84                                */
unsigned int  date_cost_price;/* 85-86      bits:yyyyyyymmmmddddd       */
unsigned int  unit_per_case; /* 87-88                                   */
unsigned char msu_c;            /* 89       multiply sell unit 0-99H    */
unsigned char mm_c;             /* 90       mix & match                 */
unsigned char return_no;        /* 91       return type or tare weight  */
```

```
   unsigned char mm_flags;      /* 92      bit 7: no mix match     */
                                /*         bit 6: return to user with
OK_MORE    */
                                /*         bit 5: continuation record of
mix match */
   unsigned char filler_u[4];   /* 93-96   reserved            */
   unsigned char subdept[6];    /* 97-102   12 digits BCD         */
   unsigned char family_num[2]; /* 103-104  4 digits BCD          */
   unsigned char user_defined[13];/* 105-117 13 bytes              */
   unsigned long weight;        /* 118-121  weight in grams        */
   unsigned char place[3];      /* 122-124  row/column/shelf+label#*/
                                /*          1b   1b     «b    «b   */
   unsigned char s_money_ptr;   /* 125                             */
   unsigned long second_price;  /* 126- 129 reversed binary long    */
   unsigned char second_price_flags;/* 130                          */
                               /* FILLER FOR FUTURE USE        */
   char filler[2];              /* 131-132  Filler                 */
                               /* sales fields                */
   unsigned int  day_date_sales;/* 133-134  bits:yyyyyyymmmmddddd   */
   long day_qty_sales;          /* 135-138                         */
   long day_revenue;            /* 139-142                         */
   long day_discount;           /* 143-146  discount     revenue    */
   long day_qty_promotion;      /* 147-150                     */
   long day_promotion;          /* 151-154  promotion    revenue */
   long p_day_qty_sales;        /* 155-158  previous day         */
   long p_day_revenue;          /* 159-162                        */
   long p_day_discount;         /* 163-166  discount day revenue   */
   long week_qty_sales;         /* 167-170                          */
   long week_revenue;           /* 171-174                          */
```

```
long week_discount;          /* 175-178                          */
long p_week_qty_sales;       /* 179-182                          */
long p_week_revenue;         /* 183-186                          */
long p_week_discount;        /* 187-190                          */
long month_qty_sales;        /* 191-194                           */
long month_revenue;          /* 195-198                      */
long month_discount;         /* 199-202                      */
long p_month_qty_sales;      /* 203-206                      */
long p_month_revenue;        /* 207-210                      */
long p_month_discount;       /* 211-214                      */
long year_qty_sales;         /* 215-218                      */
long year_revenue;           /* 219-222                      */
long year_discount;          /* 223-226                      */
long z_qty_sales;            /* 227-230                      */
long z_revenue;              /* 231-234                      */
long z_discount;             /* 235-238                      */
long z_qty_promotion;        /* 239-242                      */
long z_promotion;            /* 243-246                      */
long pccs_save_qty_sales;    /* 247-250                      */
long pccs_save_revenue;      /* 251-254                      */
// new fields for 6.4
unsigned char not_used_bt[16];/* 255      reserved for future BT */
long day_reduce_qty_prom;    /* 271-274  sale item quantity    */
long day_sale_item;          /* 275-278  sale item revenue    */
long day_qty_sale_item;      /* 279-282  sale item quantity  */
long period_sale_item;       /* 283-286                      */
long period_qty_sale_item;   /* 287-290                      */
long period_markdown_sale;   /* 291-294                      */
```

```
   long save_price;             /* 295-298                      */

   unsigned char not_used[31];  /* 299-329                       */


   struct maint_up_    last_up; /*  330-334 last up date time + pc
created    */


   unsigned char q_dex_flags;   /* 335      byte for QuickDex      */

};
```

6.4 Parameter Structure

```
struct f_parm_

{

   unsigned indx_num;    /* 0=PLU,5=NAME,6=dep           */

   unsigned option;      /* bit0: 1= Do not create POS maintenance
records */

                         /* bit1: 0= V6.4                */

                         /* bit2: reserved, set to 0     */

                         /* bit3: 1=multi user           */

                         /* bit4: 1=read relative+index  */

                         /* bit5: reserved, set to 0     */

                         /* bit6: reserved, set to 0     */

                         /* bit7: reserved, set to 0     */

   unsigned hi_word_write_part_offset;

   unsigned write_part_offset; /* offset of string in partial write
   */

   unsigned write_part_length; /* length of string in partial write
   */

char filler[20];              /* zero filler
   */

} f_parm;
```

6.4 Inquiry PLU Calls

- **Enhanced read matching item ('READ') - EPLUREAD**

  Read ('get') the matching item. The plu-record must include the PLU as a key before the call. After the call, if the return code is OK, the plu-record is filled with the details of the item which is equal to the requested PLU.

  ```
  rtc = epluread(&f_parm,&plu);
  ```

- **Enhanced read matching or next item ('START') - EPLUSTART**

  Read ('get') the matching or next item. The plu-record must include the PLU as a key before the call. After the call, if the return code is OK, the plu-record is filled with the details of the item which is equal to, or the next following of the requested PLU.

  ```
  rtc = eplustart(&f_parm,&plu);
  ```

- **Enhanced read next item ('NEXT') - EPLUREADNEXT**

  Read ('get') the next item. The plu-record must include the current PLU as a key before the call. After the call, if the return code is OK, the plu-record is filled with the details of the item which is the next following of the requested PLU.

  ```
  rtc = eplureadnext(&f_parm,&plu);
  ```

- **Enhanced read previous item ('PREV') - EPLUREADPREV**

  Read ('get') the previous item. The plu-record must include the current PLU as a key before the call. After the call, if the return code is OK, the plu-record is filled with the details of the item which is the previous before the requested PLU

  ```
  rtc = eplureadprev(&f_parm,&plu);
  ```

- **Enhanced read last item - EPLUREADLAST**

  Read ('get') the last item. The plu-record must include the current PLU as a key before the call. After the call, if the return code is

OK, the plu-record is filled with the details of the item which is the last PLU.

```
rtc = eplureadlast(&f_parm,&plu);
```

6.4 Maintenance PLU Calls

Maintenance PLU calls are organized in a batch mode. Before any update, a routine for starting the batch must be called, then a batch of updates is built. Only the PLU file in the PC is updated in this phase, and afterwards a routine for executing the batch is called. The POSTs are then updated.

- **Enhanced update item ('UPDATE') - EPLUUPDATE**

  Update the matching item. The plu-record must be filled with all the fields before the call. After the call, if the return code is OK, the PC PLU file is updated and an update PLU record is added to the maintenance batch.

  ```
  rtc = epluupdate(&f_parm,&plu);
  ```

- **Enhanced delete item ('DELETE') - EPLUDELETE**

  Delete the matching item. The plu-record must only be filled by the PLU number field before the call. After the call, if the return code is OK, the PC PLU file is updated and a delete PLU record is added to the maintenance batch.

  ```
  rtc = epludelete(&f_parm,&plu);
  ```

- **Enhanced insert item ('INSERT') - EPLUINSERT**

  Insert a new item. The plu-record must be filled with all the fields before the call. After the call, if the return code is OK, the PC PLU file is updated.

  ```
  rtc = epluinsert(&f_parm,&plu);
  ```

6.3 PLU Record Structure

```
   struct  rlt_plu_rec_         /* OFFSET | TYPE               */
{                               /* --------------------------------*/
   // RELATIVE PART
   unsigned char plu_number[7]; /* 0 - 6    13 digits BCD              */
   unsigned char dep[2];        /* 7 - 8    0 - 999                    */
   unsigned char name[40];      /* 9 -48    ASCII name                 */
   unsigned char name_pos[16];  /* 49-64                               */
   unsigned char tax;           /* 65       byte :                     */
                                /*          bit 7- TAX rate 8          */
                                /*          bit 6- TAX rate 7          */
                                /*          bit 5- TAX rate 6          */
                                /*          bit 4- TAX rate 5          */
                                /*          bit 3- TAX rate 4          */
                                /*          bit 2- TAX rate 3          */
                                /*          bit 1- TAX rate 2          */
                                /*          bit 0- TAX rate 1          */
   unsigned char item_flags1;   /* 66       byte                       */
                                /*          bit 7- payment by WIC      */
                                /*          bit 6- assign EAN IREF     */
                                /*          bit 5- vendor coupon       */
                                /*          bit 4- store coupon        */
                                /*          bit 3- negative entry      */
                                /*          bit 2- RESERVED *******    */
                                /*          bit 1- non-merchandise     */
                                /*          bit 0- pay credit/foodstamp */
   unsigned char item_flags2;   /* 67       byte                       */
```

```
                                /*          bit 7- Non-discount          */
                                /*          bit 6- Inhibit Qty/Rpt       */
                                /*          bit 5- weighed item          */
                                /*          bit 4- Fore Qty              */
                                /*          bit 3- Manual price          */
                                /*          bit 2- Default price         */
                                /*          bit 1- Not for sale          */
                                /*          bit 0- promotional           */
unsigned char item_flags3;      /* 68       byte                         */
                                /*          bit 7-  item sale            */
                                /*          bit 6-                       */
                                /*          bit 5-                      */
                                /*          bit 4-                      */
                                /*          bit 3-                      */
                                /*          bit 2-                      */
                                /*          bit 1-                      */
                                /*          bit 0-                      */
long price;                     /* 69-72                                 */
unsigned int  date_price;       /* 73-74    bits:yyyyyyymmmmddddd       */
                                /*          ,year 0 is 1980             */
long center_price;              /* 75-78                                 */
unsigned int  date_ctr_price;/* 79-80    bits:yyyyyyymmmmddddd       */
long cost_per_case_price;       /* 81-84                                 */
unsigned int  date_cost_price;/* 85-86   bits:yyyyyyymmmmddddd       */
unsigned int  unit_per_case; /* 87-88                                 */
unsigned char msu_c;            /* 89       multiply sell unit 0-99H     */
unsigned char mm_c;             /* 90       mix & match                  */
unsigned char return_no;        /* 91       return type or tare weight   */
```

```
   unsigned char mm_flags;      /* 92      bit 7: no mix match           */
                                /*         bit 6: return to user with
OK_MORE    */
                                /*         bit 5: continuation record of
mix match */
   unsigned char filler_u[4];   /* 93-96   reserved                      */
   unsigned char subdept[6];    /* 97-102   12 digits BCD                */
   unsigned char family_num[2]; /* 103-104  4 digits BCD                 */
   unsigned char user_defined[13];/* 105-117 13 bytes                    */
   unsigned long weight;        /* 118-121  weight in grams              */
   unsigned char place[3];      /* 122-124  row/column/shelf+label#      */
                                /*          1b   1b     «b    «b         */
   unsigned char s_money_ptr;   /* 125                                   */
   unsigned long second_price;  /* 126- 129 reversed binary long         */
   unsigned char second_price_flags;/* 130                               */
                               /* FILLER FOR FUTURE USE                  */
   char filler[2];              /* 131-132  Filler                       */
                               /* sales fields                           */
   unsigned int  day_date_sales;/* 133-134  bits:yyyyyyymmmmddddd        */
   long day_qty_sales;          /* 135-138                               */
   long day_revenue;            /* 139-142                               */
   long day_discount;           /* 143-146  discount     revenue         */
   long day_qty_promotion;      /* 147-150                               */
   long day_promotion;          /* 151-154  promotion    revenue         */
   long p_day_qty_sales;        /* 155-158  previous day                 */
   long p_day_revenue;          /* 159-162                               */
   long p_day_discount;         /* 163-166  discount day revenue         */
   long week_qty_sales;         /* 167-170                               */
   long week_revenue;           /* 171-174                               */
```

```
   long week_discount;          /* 175-178                              */
   long p_week_qty_sales;       /* 179-182                              */
   long p_week_revenue;         /* 183-186                              */
   long p_week_discount;        /* 187-190                              */
   long month_qty_sales;        /* 191-194                              */
   long month_revenue;          /* 195-198                              */
   long month_discount;         /* 199-202                              */
   long p_month_qty_sales;      /* 203-206                              */
   long p_month_revenue;        /* 207-210                              */
   long p_month_discount;       /* 211-214                              */
   long year_qty_sales;         /* 215-218                              */
   long year_revenue;           /* 219-222                              */
   long year_discount;          /* 223-226                              */
   long z_qty_sales;            /* 227-230                              */
   long z_revenue;              /* 231-234                              */
   long z_discount;             /* 235-238                             */
   long z_qty_promotion;        /* 239-242                             */
   long z_promotion;            /* 243-246                             */
   long pccs_save_qty_sales;    /* 247-250                             */
   long pccs_save_revenue;      /* 251-254                             */
unsigned char q_dex_flags;   /* 255      byte for QuickDex              */
};
```

Parameters for Execution of Maintenance Batch Structure
(Version 6.3 only)

```
struct exec_maint_prm_

{

   unsigned int  timeout_seconds;        // timeout in seconds

   unsigned char pos_result[32];         // pos 1-32 results:1=ok,0=not
updated

}exec_maint_p;
```

6.3 Inquiry PLU Calls

- **Read matching item ('READ') - PLUREAD**

  Read ('get') the matching item. The plu-record must include the PLU as a key before the call. After the call, if the return code is OK, the plu-record is filled with the details of the item which is equal to the requested PLU.

  ```
  rtc = pluread(&plu);
  ```

- **Read matching or next item ('START') - PLUSTART**

  Read ('get') the matching or next item. The plu-record must include the PLU as a key before the call. After the call, if the return code is OK, the plu-record is filled with the details of the item which is equal to or the next following of, the requested PLU.

  ```
  rtc = plustart(&plu);
  ```

- **Read next item ('NEXT') - PLUREADNEXT**

  Read ('get') the next item. The plu-record must include the current PLU as a key before the call. After the call, if the return code is OK, the plu-record is filled with the details of the item which is the next following of the requested PLU.

  ```
  rtc = plureadnext(&plu);
  ```

- **Read previous item ('PREV') - PLUREADPREV**

  Read ('get') the previous item. The plu-record must include the current PLU as a key before the call. After the call, if the return code is OK, the plu-record is filled with the details of the item which is the previous before the requested PLU.

  ```
  rtc = plureadprev(&plu);
  ```

6.3 Maintenance PLU Calls

Maintenance PLU calls are organized in a batch mode. Before any update a routine for starting the batch must be called, then a batch of updates is built. Only the PLU file in the PC is updated in this phase, and afterwards a routine for executing the batch is called. The POSTs are then updated.

- **Update item ('UPDATE') - PLUUPDATE**

  Update the matching item. The plu-record must be filled with all the fields before the call. After the call, if the return code is OK, the PC PLU file is updated and an update PLU record is added to the maintenance batch.

  ```
  rtc = pluupdate(&plu);
  ```

- **Delete item ('DELETE') - PLUDELETE**

  Delete the matching item. The plu-record must be filled the PLU number field only before the call. After the call, if the return code is OK, the PC PLU file is updated and a delete PLU record is added to the maintenance batch.

  ```
  rtc = pludelete(&plu);
  ```

- **Insert item ('INSERT') - PLUINSERT**

  Insert a new item. The plu-record must be filled with all the fields before the call. After the call, if the return code is OK, the PC PLU file is updated.

  ```
  rtc = pluinsert(&plu);
  ```

■ Start maintenance batch - STARTMAINT

```
rtc = startmaint(1);


          puts( ' EXECUTE MAINTENANCE ');
          rtc = executemaint(&exec_maint_p);
          if ( rtc == BAD_POS_UPDATE ) // not all pos updated
          for (i=0; i<32; i++)
          if (exec_maint_p.pos_result[i] == 0)
          printf (' POS %d not updated \n', i+1);


          puts( ' RESET MAINTENANCE ');
          rtc = resetmaint();
```

DRVFILE - Promotion Interface

A special interface for PROMOTION programming is provided through the DRVFILE extensions library. The library supports calls for promotions inquiry and maintenance.

Promotion Record Structure

```
struct prom_rec_

    {

    unsigned char key[11];          // BCD: 7 bytes plu + YY + MM + DD +
0

    unsigned char revoke_date[3];   // BCD: DD + MM YY

    long int price;                 // assume 2 dec. places

    unsigned char description[16];

    unsigned char id[4];

    unsigned char standard[2];      // BCD: 4 digits

    unsigned char discount[2];      // BCD: 4 digits

    unsigned char flag;             // Bit 0 (msb): reduced item

                                    // Bit 1:       offer item

                                    // Bit 2:       promotion item

                                    // Bit 3:       price is a
percentage

                                    // Bit 4:       linked offer

                                    // Bit 5:       limited quantity
offer

                                    // Bit 6:       mqt offer with
average price

                                    // Bit 7 (lsb): mqt offer with 2nd
plu price

    unsigned int offer_num;

    unsigned char action;           // always '3'

    unsigned char active;           // 1 = currently active
```

```
unsigned char revoked;
unsigned char seq_pointer;
unsigned char min_purchase_flag;
long          min_purchase_amount;
unsigned char min_purchase_ctrl[2];
char          filler[10];
char promo_data[61];
char q_dex;                    // reserved qdx flag
}   prom;
```

- **Read matching promotion ('READ') - PROMREAD**

  Read ('get') the matching promotion. The prom-record must include the PLU+date as a key before the call. After the call, if the return code is OK, the prom-record is filled with the details of the promotion which is equal to the requested Promotion.

  ```
  rtc = promread(&prom);
  ```

- **Read matching or next promotion ('START') - PROMSTART**

  Read ('get') the matching or next promotion. The prom-record must include the PLU+date as a key before the call. After the call, if the return code is OK, the prom-record is filled with the details of the promotion which is equal to, or the next following of the requested promotion.

  ```
  rtc = promstart(&prom);
  ```

- **Read next promotion ('NEXT') - PROMREADNEXT**

  Read ('get') the next item. The prom-record must include the current prom as a key before the call. After the call, if the return code is OK, the prom-record is filled with the details of the next following promotion.

  ```
  rtc = promreadnext(&prom);
  ```

- **Read previous promotion ('PREV') - PROMREADPREV**

  Read ('get') the previous item. The prom-record must include the current prom as a key before the call. After the call, if the return code is OK, the prom-record is filled with the details of the previous promotion.

  ```
  rtc = promreadprev(&prom);
  ```

Maintenance PROMOTION Calls

- **Update promotion ('UPDATE') - PROMUPDATE**

  Update the matching promotion. The promotion-record must be filled with all the fields before the call. After the call, if the return code is OK, the PC PROMOTION file is updated and an update PLU record is added to the maintenance batch.

  ```
  rtc = promupdate(&prom);
  ```

- **Delete promotion ('DELETE') - PROMDELETE**

  Delete the matching promotion. The prom-record must only be filled by the PLU+date fields before the call. After the call, if the return code is OK, the PC PROMOTION file is updated.

  ```
  rtc = promdelete(&prom);
  ```

- **Insert promotion ('INSERT') - PROMINSERT**

  Insert a new promotion. The prom-record must be filled with all the fields before the call. After the call, if the return code is OK, the PC PROMOTION file is updated.

  ```
  rtc = prominsert(&prom);
  ```

DRVFILE - QuickDex Interface

All DRVFILE routines starting with 'PQ' are identical to q-dex routines, but with redirection to remote computers and/or LAN backup, according to DRVFILE setup parameters. See more in QuickDex documentation.

Example:

```
CALL PQREAD (f.param$,f.record$,ret%)

   f.param is parameter string of 30 bytes formatted:

       Byte, File number : 0 to 31 Quickdex files

       Byte, Redirection : 0 redirect according to DRVFILE tables

                           255 force local operation only.

                           1-16 force operation on this PC number

       Word,   Opt bit 0:    Reserved for system  usage.

                   bit 1:            1=Do not update user record

                   bit 2:            1=Do not use buffer,always read
disk

                   bit 3:            1=Write zeros instead of user
data

                   bit 4:            1=Multi-user environment i.e:

                                     commands like read-next will
not be

                                     affected by other user read

                                     commands,user must supply last
record read.

                   bit 5:            1=read/write relative expansion
of

                                     enhanced index file

                   bit 6:            1=exchange hi and low parts of
offset

                   bits 7-15: Reserved.

       Word,          High word of write_part_offset
```

Word, Low word of write_part_offset: Offset of string in partial

write and other commands. Word, write_part_length: Length of

string in partial write

Word, QuickDex opcode: usually filled by filelnkX module

Word, Mode : usually will be filled by filelnkX module

bit 0 : 1= Comm answer after execution

bit 1 : 1= Write operation,0= Read operation

bits 4,3,2 : transmit data mode:

000 - xmt key only

001 - xmt all record

010 - xmt key+write_part_length data

011 - dont xmt data

bits 8,6,5 : receive data mode:

0 00 - rcv ret-code only

0 01 - rcv all record

0 10 - rcv active keys structure

0 11 - rcv first 4 bytes

bit: 1 - mask communication will be set according

to mask_comm parameter in FILE.PRM file.

Byte, Monitor_mode

bit 0: 1=Display message during monitoring

bit 1: 1=Display key during monitoring

bit 2: 1=Convert key from BCD to ASCII.

Byte, monitor_msg : Monitoring message number 0 to 255

14 bytes,Reserved.

f.record is user's record

ret% - return code:

all Q-DEX return codes plus

IHSS_COM_ERROR	0EH	
NET_FATAL_COM_ERROR	0FH	Communication time out
NET_TEMP_COM_ERROR	10H	
E_TIMEOUT	110H	Communication time out
E_IB_FAIL	112H	PC-IB comm fail error
LAN_WAIT	113H	wait for lan answer
INDEX_NOT_LOADED	114H	indexes not loaded yet

General Purpose File Interface (GPFI)

About GPFI

The GPFI allows the application to access any DOS file within the system.

The GPFI allows any B/O connected to POS*ware* via the LAN to access files on the PC during front-end serving. GPFI uses the LAN line as its link to transfer data back and forth. A set of enhanced messages has been added to the protocol to enable this job.

The GPFI is part of POS*ware*'s code that uses the B/O POLLING mechanism to create a communication link between the PCs. The PC is polled in exactly the same way as a POS, the only difference is in the address of the PC. PC's addresses on the LAN are F0, whereas on POS would be less than this value.

All drvfile routines starting with 'PG' are GPFI routines, with redirection to remote computers and/or LAN backup according to DRVFILE parameters.

Example:

```
   CALL PGSEQNEWOPEN(f.param$,f.record$,ret%)

      f.param is parameter string of 30 bytes formatted:

      Byte,  File number of relocation: determines on which PC the file
is located

             according to setup parameters.0 to 31 Q-dex files,

      Byte,  Redirection :  0 redirect according to DRVFILE tables 255
force local

             operation   only.  1-16 force operation on this PC number

      Byte,  Option: Reserved

      Byte,  Command: Usually will be filled by filelnkX module

             1-Sequential New Open.

             2-Sequential Append Open.

             3-Random Open.
```

4-Close.

5-Sequential Write.

6-Random Write.

7-Random Read.

8-Sequential Read

9-Random Open file if exist.

10-Get first matching file

11-Get next matching file

12-Set file date and time

13-Get system date and time

14-Set system date and time

15-Delete a file

16-Make a sub directory

17-Remove sub directory

18-Seek (move file pointer)

19-Flush dos buffers

20-Get free disk space

21-Reserved

22-Rename file or directory

23-IHSS control (poll list etc,detailed in Sub command)

24-Check password

Word, Data Length: Length of DATA sent/received in integer format ([low Byte]

[high Byte])

Long, Record Number: Record # Information for random Read/Write/open in 'C'

Long Format: [low Word] [high Word][low,high] [low, high]

For commands 12-14 first word is Date formatted yyyyyyymmmmddddd,

```
              year 0 is 1980. second word is time formatted
hhhhhmmmmmmxxxxx,

              xxxxx is seconds/2.

              For commands 11-12 first word is attribute of files to
search:

                   0      = Regular files only.

                   8      = Volume Label also.

                   10H    = Sub-directories also.

       Byte,  GPFI File Number, returned by OPEN commands.

       Byte,  Sub-command (used with IHSS control command 23)

                   0-START POLL

                   1-STOP POLL

                   2-SET POLL list

                   3-GET POS communication status

                   4-SEND data to IHSS POS

                   5-Reserved

                   6-SET POLL TYPE (dynamic or static polling)

                   7-Start/Stop monitoring


       Word,  Mode;

              bit 0 :     1= Comm answer after execution

              bit 1 :     1= Write operation,0= Read operation

              bits 4,3,2 :        Receive data mode:

                          000 - no rcv data

                          001 - rcv GPFI file number (1 byte)

                          010 - rcv data formatted:

                                  Word - length of following data

                                  N bytes - Data
```

```
                bit 7:      1 - mask communication will be set according
to mask_comm
                            parameter in FILE.PRM

                Byte,       Monitor_mode
                            bit 0: 1=Display msg during monitoring
                            bit 1: 1=Display key during monitoring
                            bit 2: 1=Convert key from bcd to ascii
                Byte,       Monitor_msg: Monitoring message number 0-255
                            14 bytes, Reserved.

                f.record is user record

                ret% - return code:
                            OK              0
                            GPFI_ERROR      99 Dec, 63H
```

GPFI Command Descriptions

Sequential new open: Command 01

Open a new sequential file (existing file is destroyed).

File name can include a path name and is transmitted in the DATA field.

Data length field contains the length of the filename.

Record size is 1 byte.

File Pointer is positioned at the beginning of file.

Return code:
- 00 - open ok
- 99 - file already open
- other - communication error

GPFI file number returned in f.param GPFI File Number field.

Call from BASIC:

```
call pgseqnewopen(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgseqnewopen(&f_param,f_record);
```

Sequential append open: Command 02

Open sequential file and place file pointer at the END OF FILE. If the file does not exist then create it. File name can include a path name and is transmitted in DATA field.

Record size is 1 byte.

File Pointer is positioned at the end of file.

Data length field contains the length of the filename.

Return code:
- 00 - open ok
- 99 - file already open
- other - communication error

GPFI file number returned in f.param GPFI File Number field.

Call from BASIC:

```
call pgseqapndopen(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgseqapndopen(&f_param,f_record);
```

Random open: Command 03

Open a random file. A specific record size must be defined in the open procedure. In a standard system the maximum record size is limited to 1000 bytes. File name can include a path name and is transmitted in the DATA field.

Record size is transmitted in the Record Number field.

File Pointer is positioned at the beginning of file.

Data length field contains the length of the filename.

Return code:	00	- open ok
	99	- file already open
	other	- communication error

GPFI file number returned in f.param GPFI File Number field.

Call from BASIC:

```
call pgrndopen(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgrndopen(&f_param,f_record);
```

Open random file only if exist: Command 09

Random file open only if exist. A specific record size must be defined in the open procedure. In a standard system the maximum record size is limited to 1000 bytes. File name can include a path name and is transmitted in the DATA filed.

Record size is transmitted in the Record Number field.

File Pointer is positioned at the beginning of file.

Data length field contains the length of the filename.

Return code: 00 - open ok
99 - file already open
other - communication error

GPFI file number returned in f.param GPFI File Number field.

Call from BASIC:

```
call pgrndopenex(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgrndopenex(&f_param,f_record);
```

Close: Command 04

Close the GPFI file specified in GPFI File Number field.

Return code: 00 - open ok
99 - file not open
other - communication error

Call from BASIC:

```
call pgclose(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgclose(&f_param,f_record);
```

Sequential write: Command 05

Write data at current File pointer position to the GPFI file specified in GPFI File Number field. Data length is limited to 1000 bytes. The data to write is transmitted in the DATA field. The File Pointer is positioned at the end of the written data.

Return code: 00 - write ok
99 - error in writing
other - communication error

Call from BASIC:

```
call pgseqwrt(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgseqwrt(&f_param,f_record);
```

Random write: Command 06

Write a data record at the record number location to the GPFI file specified in GPFI File Number field. The record size is the size defined during the random open. The exact write location in bytes is: Byte location = (Record No.)*(Record size). Amount of data = (Data Length). The actual data to write is transmitted in the DATA field. The File Pointer is positioned at the end of the written data.

Return code:	00	- write ok
	99	- error in writing
	other	- communication error

Call from BASIC:

```
call pgrndwrt(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgrndwrt(&f_param,f_record);
```

Random read: Command 07

Read a data record from the record number location from the GPFI file specified in GPFI File Number field. The record size is the size defined during the random open. The exact read location in bytes is: Byte location = (Record No.)*(Record size). Amount of data to read = (Data Length) File Pointer is positioned at the end of the data read. Data Length returns the actual number of bytes read.

Return code:	00	- read ok
	99	- error in reading
	other	- communication error
	DATA	- Data read.

Call from BASIC:

```
call pgrndread(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgrndread(&f_param,f_record);
```

Sequential read: Command 08

Read a data record from File Pointer location from the GPFI file specified in GPFI File Number field. Amount of data to read = (Data Length). The File Pointer is positioned at the end of the data read. Data Length returns the actual number of bytes read.

Return code:	00	- read ok
	99	- error in reading
	other	- communication error
	DATA	- Data read

Call from BASIC:

```
call pgseqread(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgseqread(&f_param,f_record);
```

Find first matching file: Command 10 (0Ahex)

This function searches for the first file name matching the filespec (ASCIIZ).The filespec may include a drive letter and path, and the file name may include wildcard characters. The filespec is transferred in the data buffer and the file name must be delimited with a null (0hex) byte. Attributes of files to search are passed in the Record Number field:

0	=	Regular files only.
8	=	Volume Label also.
10H	=	Sub-directories also.

This function is similar to the DOS int 21H function 4EH.

The return code may be:

00	-	Filespec found
02	-	File not found
03	-	Path not found
18	-	No more file to be found

If the file is found the DATA buffer will contain the following information (43 bytes):

Offset	Size	Description
00	21	Used by DOS for Find Next processing
15	01	Attribute of file found
16	02	Time stamp of file
18	02	Date stamp of file
1A	04	File size of bytes (double word)
1E	13	File name and extension, as an ASCIIZ string

The file name and extension are reported in conventional notation, with blanks removed and a period between the filename and extension. If the file has no extension the period is suppressed.

Call from BASIC:

```
call pgfindf(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgfindf(&f_param,f_record);
```

Find next matching file: Command 11 (0Bhex)

This function searches for the next filename after a previous command 10 or command 11.

The DATA must contain 43 bytes that were returned in the previous command.

This function is similar to the DOS int 21H function 4FH.

Return code	00	- Filespec found
	18	- No more files to be found

If the file is found, the DATA buffer will contain the following information (43 bytes):

Offset	**Size**	**Description**
00	21	Used by DOS for Find Next processing
15	01	Attribute of file found
16	02	Time stamp of file
18	02	Date stamp of file

1A	04	File size of bytes (double word)
1E	13	File name and extension, as an ASCIIZ string

The file name and extension are reported in conventional notation, with blanks removed and a period between the filename and extension. If the file has no extension the period is suppressed.

Call from BASIC:

```
call pgfindn(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgfindn(&f_param,f_record);
```

Set file date and time: Command 12 (0CH)

Sets file date and time stamp in the directory. The file is a previously opened GPFI file specified in GPFI File Number field.

Date and time are sent in Record Number field.

First word is Date formatted yyyyyyymmmmddddd, where:

yyyyyyy	is the binary number of year (0-119; year 0 is 1980).
mmmm	is the binary number of month (1-12).
dddd	is the binary number of day (1-31).

Second word is Time formatted hhhhhmmmmmmxxxxx, where:

hhhhh	is the binary number of hour (0-23),
mmmmmm	is the binary number of minutes (0-59),
xxxxx	is the binary number of two second increments (0-29).

Return code:	00	- ok
	99	- error (wrong date/time)
	other	- communication error

Call from BASIC:

```
call pgsetfdate(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgsetfdate(&f_param,f_record);
```

Get system date and time: Command 13 (0DH)

Gets the computer date and time.

Date and time are received in Record Number field.

First word is Date formatted yyyyyyymmmmdddddd, where:

yyyyyyy	is the binary number of year (0-119; year 0 is 1980).
Mmmm	is the binary number of month (1-12).
Dddd	is the binary number of day (1-31).

Second word is Time formatted hhhhhmmmmmmmxxxxx, where:

hhhhh	is the binary number of hour (0-23),
mmmmmm	is the binary number of minutes (0-59),
xxxxx	is the binary number of two second increments (0-29).

Return code: 00 - ok
99 - error

other - communication error

Call from BASIC:

```
call pggetfdat(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pggetdate(&f_param,f_record);
```

Set system date and time: Command 14 (0EH)

Sets the computer date and time.

Date and time are sent in Record Number field.

First word is Date formatted yyyyyyymmmmdddddd, where:

yyyyyyy	is the binary number of year (0-119; year 0 is 1980).
Mmmm	is the binary number of month (1-12).
Dddd	is the binary number of day (1-31).

Second word is Time formatted hhhhhmmmmmmxxxxx, where:

hhhhh	is the binary number of hour (0-23),
mmmmmm	is the binary number of minutes (0-59),
xxxxx	is the binary number of two second increments (0-29).

Return code:
- 00 - ok
- 99 - error (wrong date/time)
- other - communication error

Call from BASIC:

```
call pgsetdate(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgsetdate(&f_param,f_record);
```

Delete file: Command 15 (0FH

Deletes a file from the disk. File name can include a path, and is transmitted in the DATA field. Wildcard characters (?,*) are not supported. The Data length field contains the length of the filename.

Return code:
- 00 - ok
- 99 - error (file does not exist)
- other - communication error

Call from BASIC:

```
call pgdel(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgdel (&f_param,f_record);
```

Make a sub directory: Command 16 (10H)

The subdirectory name is transmitted in the DATA field. The Data length field contains the length of the subdirectory.

Return code:
- 00 - ok
- 99 - error (directory already exists)
- other - communication error

Call from BASIC:

```
call pgmkdir(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgmkdir (&f_param,f_record);
```

Remove a sub directory: Command 17 (11H)

The subdirectory name is transmitted in the DATA field. Data length field contains the length of the subdirectory.

Return code:	00	- ok
	99	- error (directory does not exist or not empty)
	other	- communication error

Call from BASIC:

```
call pgrmdir(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgrmdir (&f_param,f_record);
```

Seek: Command 18 (12H)

Moves the file pointer of a GPFI file specified in GPFI File Number field. The record number field specifies the new location of the file pointer.

Return code:	00	- ok
	99	- error (file not opened, seek failed)
	other	- communication error

Call from BASIC:

```
call pgseek (f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgseek (&f_param,f_record);
```

Flush DOS buffers: Command 19 (13H)

Writes all DOS buffers to disk.

Return code:	00	- ok
	99	- error
	other	- communication error

Call from BASIC:

```
call pgflushbuf(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgflushbuf (&f_param,f_record);
```

Get disk free space: Command 20 (14H)

The drive's value is placed in GPFI file number field (0=Default drive,1=A,2=B etc.). The free space in byte units is returned in record number field.

Return code:	00	- ok
	99	- error
	other	- communication error

Call from BASIC:

```
call pgdiskfree(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgdiskfree (&f_param,f_record);
```

Rename a file or directory: Command 22 (16H)

The old name and the new name are transmitted in the DATA field, in this format: Old file name, binary zero delimiter, new file name. Data length field contains: length of old filename + length of new file name + 1.

Return code:	00	- ok
	99	- error (directory already exists)
	other	- communication error

Call from BASIC:

```
call pgrename(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgrename (&f_param,f_record);
```

Notes

GPFI files can be opened randomly and read/write sequentially, or visa versa. Sequential open is the same as random open with 1 byte record size. Seek operation (moving File Pointer) can be performed either by seek command or by Random Read with zero data length. The File Pointer is positioned at location (Record Size * Record Number).

All GPFI files must be closed after processing ends. Failure to do so may cause file corruption, or 'lost clusters' if the DOS CHKDSK program is run.

Maximum of 20 GPFI files can be opened. Include this line in the CONFIG.SYS file: FILES=40 This allows the application to open other DOS files.

HSI Interface using GPFI Commands

One GPFI command (23 = 17Hex) is dedicated for various HSI control functions.

Set poll list: (subcommand-2)

The channel value is placed in GPFI file number field (0=HSI A 1=HSI B).

The number of POSs to poll is placed in the data_len field.

Return code: 00 - ok

Call from BASIC:

```
call pgsetpolist (f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgsetpolist (&f_param,f_record);
```

Start poll: (subcommand-0)

The channel value is placed in GPFI file number field (0=HSI A 1=HSI B).

Zero is placed in the data_len field.

Return code: 00 - ok

Call from BASIC:

```
call pgstartpol (f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgstartpol (&f_param,f_record);
```

Stoppoll: (subcommand-1)

The channel value is placed in GPFI file number field (0=HSI A 1=HSI B).

Zero is placed in the data_len field.

Return code: 00 - ok

Call from BASIC:

```
call pgstoppol (f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgstoppol (&f_param,f_record);
```

Start/Stop monitoring: (subcommand 7)

The user must allocate a buffer of 31KB formatted :

```
struct monitor_
          {
          unsigned int head;
          unsigned int tail;
          struct mon_msg_
          {
          unsigned char pos_num_a;
          unsigned char pc_num_a;
          unsigned char pc_num_b;
          unsigned char direction;   // 0= a->b1= b->a
          unsigned intlen;
          unsigned char data[1024];
          }msg[30];
          }monitor_buff;
```

To start monitoring, the first byte of the buffer must be zero. Otherwise the command is 'stop monitoring'.

Monitoring is used for applications that need to receive HSI and LAN messages.

Return code: 00 - ok

Call from BASIC:

```
call pgmon (f.param$,monitor.buff$,ret%)
```

Call from C:

```
unsigned int ret = pgmon (&f_param,&monitor_buff);
```

Set poll type (static or dynamic poll): (subcommand-6)

The channel value is placed in GPFI file number field (0=HSI A 1=HSI B).

1 is placed in the data_len field.

The number of times to poll responding POSs before a non-responding POS is polled. 0 means static polling according to the poll list.

Return code: 00 - ok

Call from BASIC:

```
call pgsetpoltyp (f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgsetpoltyp (&f_param,f_record);
```

Get pos status: (subcommand-3)

The channel value is placed in GPFI file number field (0=HSI A 1=HSI B).

Pairs of POS number, status is returned in user record. Status not equal to 0 means POS responds.

Return code: 00 - ok

Call from BASIC:

```
call pggetposstat(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgsetposstat (&f_param,f_record);
```

Send text to POS on HSI line: (subcommand-4)

The POS number is placed in GPFI file number field.

The channel A or B is chosen by the IB according to the poll list set and the POSs that respond on the lines. If the POS is not responding on any of the HSI lines, the text is not sent.

The length of the text is placed in the data_len field.

The text is placed in the record filed.

Return code: 00 - ok

Call from BASIC:

```
call pgsendtext(f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgsendtext (&f_param,f_record);
```

Lock Functions using GPFI Commands

One GPFI command (25 = 19Hex) is dedicated for various locking functions.

Lock communication: (subcommand-0)

after its execution, DRVFILE will not receive any messages from the IB. This is done by calling DRVDMT and requesting a DOS SUSPEND state. Only the UNLOCK call will terminate this state. Make sure that the locked period is short so that IBQ will not be filled with received messages.

Make sure that the redirection is to the local PC. The lock status will be active till the end of the calling application process (till DOS_EXIT function is called).

Return code: 00 - ok

Call from BASIC:

```
call pgcomlock (f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgcomlock (&f_param,f_record);
```

Unlock communication: (subcommand-1)

After its execution, DRVFILE is able to receive messages from the IB. This is done by calling DRVDMT and requesting a DOS UN-SUSPEND state.

Return code: 00 - ok

Call from BASIC:

```
call pgcomunlock (f.param$,f.record$,ret%)
```

Call from C:

```
Unsigned int ret = pgcomunlock (&f_param,f_record);
```

Lock communication after answer: (subcommand-2)

Similar to subcommand 0, but DRVFILE is actually locked only after receiving the next communication answer message.

Return code: 00 - ok

Call from BASIC:

```
call pgcomlockans (f.param$,f.record$,ret%)
```

Call from C:

```
unsigned int ret = pgcomlockans (&f_param,f_record);
```

Links to DRVFILE

DRVFILE - BASIC Interface

Interface to Microsoft BASIC compiler version 2-7 is provided by linking with FILELNKB.OBJ module.

DRVFILE - C Interface

Interface to Microsoft C compiler small model versions 3 to 6 is provided by linking with FILELNKC.OBJ module.

Interface to MSC medium model by linking with FILNKCM.OBJ module.

Interface to MSC large model by linking with FILNKCL.OBJ module.

DRVFILE - ASSEMBLER Interface

DRVFILE can be called by calling DRVPOS entry 6.

```
ALL Procedures On Entry:

          AX-     return code

                  0 =     OK

                  255 = bad DI number

                  all Q-DEX return codes plus

                  HSI_COM_ERROR          0EH

                  NET_TEMP_COM_ERROR     0FH

                  NET_FATAL_COM_ERROR    10H

          DI - function number

          BX - pointer to file parameters formatted:

               struct f_parm_

                  {

                  unsigned char indx_num;
```

```
                unsigned char redir_adrs;
                unsigned option;
                unsigned hi_word_write_part_offset;
                unsigned write_part_offset;
                unsigned write_part_length;
                unsigned qdx_opcode;
                unsigned mode;
                char filler[16];
                }f_parm;
        SI- pointer to user record
        --------------------------------
        DI = 0  Quickdex command.
        --------------------------------
        DI = 1  get parameters from drvfile to application
                        INPUT PARAMETERS:
                        DS:SI - pointer to DRVFILE parameter string
formatted the
                        same as FILE.PRM
                        OUTPUT PARAMETERS:
        AX -0 =         OK,
        AX -1 =         error
        --------------------------------
        DI =2   set parameters from application to drvfile
                        INPUT PARAMETERS:
                        DS:SI - pointer to DRVFILE parameter string
formatted the
                        same as FILE.PRM
                        OUTPUT PARAMETERS:
        AX - 0 =        OK,
```

```
AX - 1 =       error

-------------------------------------------------

DI = 3  reserved

-------------------------------------------------

DI = 4  GPFI command

-------------------------------------------------

DI = 5  reserved

-------------------------------------------------

DI = 6  reserved

-------------------------------------------------

DI = 7  Background task

-------------------------------------------------

DI = 8 - 11    reserved

-------------------------------------------------

DI =12  get LAN communication status

DS:SI-  pointer to an array of lan status

-------------------------------------------------

DI = 13 - 19   reserved for POS system function calls

-------------------------------------------------

DI = 20 - 80   reserved for POS CFS macro function calls

-------------------------------------------------

DI = 80 - 255  can be used for customized user macros

-------------------------------------------------
```

Return Codes

General return codes:

OK	0
ERROR	0x80
ERR_DRVPOS	0xFE
E_BAD_FUNCTION_NO	0xFF
E_DRV_BUSY	0x100

Return codes associated with QuickDex:

ERR_NOT_FOUND	1
ERR_INDX_START	2
ERR_INDX_READD	3
ERR_DELETED	4
ERR_EXISTS	5
ERR_DISK_READ	6
ERR_DISK_WRITE	7
ERR_INDX_NOT_LOADED	8
ERR_INDX_WRITE	9
ERR_INDX_DISK_MATCH	0xA
ERR_FILE_NOT_OPENED	0xB
ERR_LOAD_FAIL	0xC
ERR_BAD_FUNCTION_SECTION	0x20
ERR_FILE_FULL	0x21
ERR_RECORD_OVERFLOW	0x22
ERR_EXP_NOT_FOUND	0x23
ERR_EXP	0x24
ERR_EXP_FILE_FULL	0x25

```
ERR_EXP_DELETED                 0x26

ERR_EXP_EXISTS                  0x27

ERR_SUP_IDX                     0x28

ERR_MAP                         0xFD
```

Drvfile return codes:

```
E_TIMEOUT                       0x0F    //  Communication time
out

E_IB_FAIL                       0x112   //  PC-IB comm fail
error
```

Maintenance return codes:

```
MAINT_NOT_STARTED               300

MAINT_BUSY                      301

MAINT_STARTED_ALREADY           302

QDEX_ERROR                      303

MAINT_BUFF_FULL                 304

NO_HSI_MASTER                   305

BAD_POS_UPDATE                  306

OK_MORE_DRV_LOOP                307

OK_MORE_USER_LOOP               308

ERR_POS_MAINT_FILES             309
```

DRVFILE Parameters

DRVFILE reads parameters from FILE.PRM file, in this format:

```
   struct parm_

      {

      unsigned intdrvpos_entry;         // must be6

      unsigned intihss_ib_enable:1;

      unsigned intlan_ib_enable:1;

      unsigned intmaint_req_enable:1;

      unsigned intload_pos_enable:1;

      unsigned intprocess_trans_enable:1;

      unsigned intprocess_maint_enable:1;

      unsigned intreserved_ctrl:10;

      unsigned intmaint_req_time;       // send maintenance request
every N seconds

      unsigned char mode;               // bit 0 : 1=debug mode

      unsigned char pc_num;             // this PC number on LAN

      unsigned char ihss_pc_num;        // use 0 for MFS1 or MFS2

      _segment ib_adrs;                 // usually 0xd000

      unsigned intmaster_pc_num;        // usually 1

      unsigned char reserved[50];

      unsigned char password[8];        // used for RS-232
communication

                                        // redirection table - used for
LAN routing

      struct redir_table_

          {

          unsigned char read_lan_adrs;

          unsigned char write_lan_adrs;
```

```
            unsigned char write_local_disk:1;
            unsigned char mask_backup_lan:1;
            unsigned char unused:6;
            unsigned char backup_lan_adrs;
            }redir_table[10];
                                              // translation array from
Quickdex/Gpfi file
                                                 number
                                              // to redirection table entry
number
                                              // used for LAN routing char
redir_xlat[70];
            unsigned char reserved2[20];
       } parm;
```

ICL Retail Systems
2933 Bunker Hill Lane, #101
Santa Clara, CA 95054

P/N 89000057
PIN 45001/011