Danload 6000 COMMUNICATIONS SPECIFICATION

DANIEL MEASUREMENT AND CONTROL
HOUSTON, TEXAS

DanLoad 6000
COMMUNICATIONS
SPECIFICATION

Part Number: 3-9000-674
REVISION 2.1
Supports Software Version 2.1 and above

September 1998

DANIEL

Year 2000 Warranty

The Company represents and warrants that computer programs in any medium, software, firmware and combinations thereof ("Deliverables") manufactured by the Company and incorporated into or supplied by the Company for use with goods manufactured by the Company will, under normal use and care:

i) recognize and accept dates falling on or after 1 January 2000;

ii) recognize and accept the year 2000 and every succeeding fourth year as leap years;

iii) recognize and accept 29 February in the year 2000 and every succeeding fourth year;

iv) record, store, process, sequence, present and output calendar dates and data related to dates falling on or after 1 January 2000, in the same manner and with the same functionality as they do on or before 31 December 1999 and without errors or omissions; and

v) lose no functionality with respect to the introduction into them of dates or data related to dates falling on or after 1 January 2000;

provided that, in the case of any non-conforming Deliverables that are returned to the Company promptly following discovery of the non-conformity, the Company will, at its option and cost, repair or replace such Deliverable or refund to the Purchaser the purchase price therefor. This shall be the Purchaser's sole and exclusive remedy for breach of the foregoing warranty.

Notwithstanding the foregoing, the Company shall not, under any circumstances whatsoever, be liable for any defects or errors caused by: materials or workmanship made, furnished or specified by the Purchaser; non-compliance with the Company's installation or operation requirements; failure to install any revisions and/or upgrades to the Deliverables deemed mandatory by the Company; any modifications to Deliverables not previously authorized by the Company in writing; the use by the Purchaser of any non-authorized spare or replacement parts in connection with the goods used in conjunction with the Deliverables; or the use of the Deliverables with any hardware or software not supplied by the Company. The Purchaser shall at all times remain solely responsible for the adequacy and accuracy of all information supplied by it. Any third party content in Deliverables shall carry only the warranty extended by the original manufacturer.

THE FOREGOING CONSTITUTES THE COMPANY'S SOLE AND EXCLUSIVE WARRANTY IN RELATION TO THE PERFORMANCE OF THE DELIVERABLES AS IT RELATES TO THE CHANGE FROM YEAR 1999 TO YEAR 2000 OR THE OCCURRENCE OF LEAP YEARS THEREAFTER, AND THE PURCHASER'S EXCLUSIVE REMEDY FOR BREACH THEREOF. IN NO EVENT WILL THE COMPANY BE LIABLE FOR INDIRECT, CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING LOSS OF USE, BUSINESS INTERRUPTION OR LOSS OF PROFITS, IRRESPECTIVE OF WHETHER THE COMPANY HAD NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

The foregoing warranty shall remain valid until the later of December 31, 2000 or one year after the date that the Deliverable was shipped.

DANIEL INDUSTRIES, INC.
DanLoad 6000
COMMUNICATIONS
SPECIFICATION

NOTICE

DANIEL INDUSTRIES, INC. AND DANIEL MEASUREMENT AND CONTROL ("DANIEL") SHALL NOT BE LIABLE FOR TECHNICAL OR EDITORIAL ERRORS IN THIS MANUAL OR OMISSIONS FROM THIS MANUAL. **DANIEL MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THIS MANUAL AND, IN NO EVENT, SHALL DANIEL BE LIABLE FOR ANY SPECIAL OR CONSEQUENTIAL DAMAGES INCLUDING, BUT NOT LIMITED TO, LOSS OF PRODUCTION, LOSS OF PROFITS, ETC.**

PRODUCT NAMES USED HEREIN ARE FOR MANUFACTURER OR SUPPLIER IDENTIFICATION ONLY AND MAY BE TRADEMARKS/REGISTERED TRADEMARKS OF THESE COMPANIES.

WARRANTY

Daniel Measurement and Control ("Daniel") warrants all equipment manufactured by it to be free from defects in workmanship and material, provided that such equipment was properly selected for the service intended, properly installed, and not misused. Equipment which is returned, transportation prepaid to Daniel within twelve (12) months of the date of shipment (eighteen (18) months from date of shipment for destinations outside of the United States), which is found after inspection by Daniel to be defective in workmanship or material, will be repaired or replaced at Daniel's sole option, free of charge, and return-shipped at lowest cost transportation. All transportation charges and export fees will be billed to the customer. Warranties on devices purchased from third party manufacturers not bearing a Daniel label shall have the warranty provided by the third party manufacturer.

The warranties specified herein are in lieu of any and all other warranties, express or implied, including any warranty of merchantability or fitness for a particular purpose.

Daniel shall be liable only for loss or damage directly caused by its sole negligence. Daniel's liability for any loss or damage arising out of, connected with, or resulting from any breach hereof shall in no case exceed the price allocable to the equipment or unit thereof which gives rise to the claim. Daniel's liability shall terminate one year after the delivery of the equipment except for overseas deliveries and extended warranty products as noted above.

In no event, whether as a result of breach of warranty or alleged negligence, shall Daniel be liable for special or consequential damages, including, but not limited to, loss of profits or revenue; loss of equipment or any associated equipment; cost of capital; cost of substitute equipment, facilities or services; downtime costs; or claims of customers of the purchaser for such damages.

Table of Contents

This page intentionally left blank.

1 Preface

This document describes the master/slave communications protocol between an automation system (master) and Daniel Flow Products' DanLoad 6000 electronic preset unit (slave). The DanLoad 6000 electronic preset is described in more detail in the "DanLoad 6000 Reference Manual" (Daniel Industries part number 3-9000-670).

In this document, numbers are decimal (base ten) unless suffixed by an "h" in which case they are hexadecimal (base sixteen).

Modbus is a registered trademark of Modicon, Inc.

This page intentionally left blank.

2 Glossary

alarm A condition generated by the DanLoad 6000 when it detects a failure in operation or operating conditions.

additive A substance "injected" in relatively small quantities into a product which changes the qualities of that product.

automation Any equipment designed to improve safety, control access to product or facilitate measurement at a bulk loading terminal.

batch A delivery of one or more products (sequentially or in parallel) corresponding to a "preset volume" on the DanLoad 6000.

blend A combination of two or more products loaded as a single batch.

broadcast A communications command transmitted to more than one receiver simultaneously.

command A communications message requesting information or action.

component A product delivered and metered by the DanLoad 6000. (The DanLoad 6000 can deliver more than one product in a single batch.)

CRC-16 A very effective method of detecting errors in transmitted data.

DUART Dual Universal Asynchronous Receiver Transmitter. A device (or printed circuit board) that can transmit and receive data on two channels simultaneously.

keypad Arrangement of keys (push-buttons) by which data is entered into the DanLoad 6000.

load The delivery of a volume of one product through one meter as part of a batch.

Modbus A communications protocol, i.e. a method of communicating data from one device to another.

operator One who operates the DanLoad 6000 via its keypad, e.g. a truck driver, a tanker terminal employee or a service technician.

PPU Pulse per unit, e.g. one pulse per gallon or one pulse per liter.

recipe A specification of the proportions of one or more products to be loaded as a single batch.

transaction The consecutive delivery of one or more batches having the same recipe, additives and swing-arm side.

3 Protocol Overview

This section describes the communications protocol between an automation system and the DanLoad 6000. The following should be noted:

- The DanLoad 6000 has two communications "channels" (ports) with the same address that can accept commands from automation systems. The address is set on the DUART board. Address zero should not be used since only "broadcast" messages would be processed and the DanLoad 6000 would not respond to any query.

  If two automation systems are connected to the DanLoad 6000, one of them is typically responsible for "authorization", and the other one for "monitoring" (unless the two automation systems engage in some form of handshaking--possibly via the DanLoad 6000's status bits). The DanLoad 6000 receives queries on both communications channels (possibly simultaneously) and (if the query is not a retry) queues them to a single internal process which is responsible for building the response. Thus, queries are handled on a "first come, first served" basis, but an automation system never has to wait, i.e. worst case, for longer than it takes to respond to a single query on the other channel since it will always have the next place in the queue. When a response has been built, it is handed back to the appropriate channel process, i.e. the one for which the query was received, in order to be transmitted to the automation system.

  For DanLoad 6000 v5.30 and above, program code 662 must be used to configure the unit's communications address; the DUART's address switch is no longer used.

- Communications are full-duplex operating as master/slave.

- The maximum number of slaves on a common line (multidrop) is 32. In practice, due to cable lengths and data rates a smaller number of slaves may be necessary.

- The protocol is "Modbus compatible", i.e. it uses Modbus RTU message framing.

  The maximum query message length for Modbus RTU is 256 bytes (including check characters). The maximum response message length for Modbus RTU is 256 bytes (including check characters).

The first character of the data field is the "data field length". This is a count of the number of characters in the data field, and so normally has a value of 2 or greater. The maximum data field length is 252 characters, i.e. 256 - 4.

The Modbus function codes for normal queries and responses are 41h and 42h; the function codes are synchronized INDEPENDENTLY PER COMMUNICATIONS CHANNEL by the Start Communications command and are thereafter alternated by the automation system INDEPENDENTLY PER COMMUNICATIONS CHANNEL. The DanLoad 6000 knows which function code to expect next on each communications channel. If the DanLoad 6000 receives 41h when it expects 42h or receives 42h when it expects 41h (and the command is not a Start Communications command), it assumes a retry by the automation system and retransmits exactly the last response sent on the communications channel. The alternating function codes are provided primarily to allow the automation system to transmit retries if the DanLoad 6000's response is lost or corrupted. (The Modbus protocol reserves function codes 41h through 48h for user-defined "custom functions".) A communications channel's expected function code is independent of any other DanLoad 6000's on the same communications line. The DanLoad 6000's two communications channels should not be connected on the same communications line since they both have the same communications address. The automation system should maintain a "next function code" for each configured DanLoad 6000.

The DanLoad 6000 uses function codes C1h and C2h to indicate "exception responses" to function codes 41 hex and 42 hex respectively, i.e. invalid data field, in which case the response's data field contains a DanLoad 6000 Exception Response Code. C1h and C2h are not "standard" Modbus exception response codes. (See §9.1.)

The DanLoad 6000 does not respond to a "broadcast" message, i.e. address field equal to 0. The automation system must decide which commands are safe to be broadcast, e.g. Set Date and Time. A broadcast message's alternating sequence function code is not checked by the DanLoad 6000. 41h or 42h is acceptable and the DanLoad 6000 behaves as if the correct alternating sequence function code was received, so the automation system should update its copy of the alternating sequence function codes for each DanLoad 6000 that should have received the broadcast message.

See the "Modbus Protocol Reference Guide" (Gould/AEG) for additional details.

- The DanLoad 6000 operates as an addressable slave station, and "speaks only when spoken to"; this is a necessary consequence of RS-485 multidrop communications, which requires that a slave station reply only to a message directed to it.

  The DanLoad 6000 does not respond to a message that is "incorrectly framed", i.e. not its address, an unused function code or an incorrect error check.

  The DanLoad 6000 does not respond to a message that is "incomplete", i.e. fewer or more characters than expected.

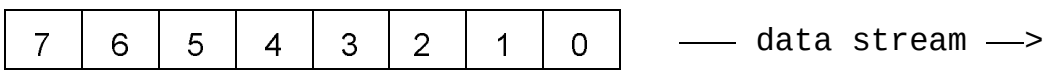
  "It is essential to program the Modbus master to assume a communications error has occurred if there is no response in a reasonable time. The length of this time period depends upon baud rate, type of message, and scan time of the slave. Once this time is determined, the master may be programmed to automatically transmit the message."

  See the "Modbus Protocol Reference Guide" (Gould/AEG) for additional details.

- The DanLoad 6000 ignores (throws away) any characters that are transmitted to it by the automation system while it is processing a previous command from the automation system.

- The DanLoad 6000 is not ready to receive characters until a few milliseconds after it has completed a transmission. The automation system should delay 50 milliseconds between receiving the last character of a message from a DanLoad 6000 and transmitting the first character of a message to the same DanLoad 6000.

- Numeric data (int, long, etc.) are transmitted in binary with the least significant byte first and the most significant byte last. Within bytes bits are labeled from 0 to 7 with 0 being the least significant bit.



  String data (arrays of characters) are null-terminated.

The DanLoad 6000 has two operating modes that limit the command codes that are accepted:

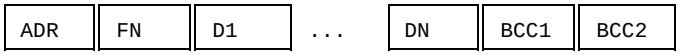
- Automatic. All command codes can be used. Status information can be requested and the DanLoad 6000's operation can be controlled.

- Manual. Only certain command codes can be used. status information can be requested but the DanLoad 6000's operation cannot be controlled.

4 Message Framing

>> *Modbus RTU* <<

Queries, responses and exception responses are framed as follows:



Where...

ADR	Address field, e.g. 2Ah.
FN	Function field. The function code is 41h or 42h for normal queries and responses, C1h or C2h for exception responses.
D1	Data field character 1.
DN	Data field character N.
BCC1	Error check field (CRC-16) character 1.
BCC2	Error check field (CRC-16) character 2.

The "data field length" and the "command code" are the first and second characters respectively of the data field (D1 through DN). An exception response's data field contains the data field length = 3 (D1), command code received by the DanLoad 6000 (D2) and exception response code (D3).

"The receiving device monitors the elapsed time between receipt of characters. If three and one-half character times elapse without a new character or completion of the frame, then the device flushes the frame and assumes that the next byte received will be an address." See the "Modbus Protocol Reference Guide" (Gould/AEG) for additional details.

In Modbus RTU, it is a bad idea to send the address, function code and data field and then compute and send the error check field since the elapsed time (indeterminate on many operating systems!) between the two writes to the port may cause an inadvertent "end of frame".

This page intentionally left blank.

5 Communications Methodology

5.1 Communications Start-Up

The automation system can communicate with the DanLoad 6000 in the manual or automatic operating modes. In either case a "communications start-up" must take place on each communications channel that is used. The DanLoad 6000 will not respond to any command from the automation system until communications on that channel have been started. Communications can be started only in the loading mode. In program mode the DanLoad 6000 does not respond to commands from the automation system; the automation system must restart communications with the Start Communications command after the DanLoad 6000 has exited program mode. (The automation system should repeatedly send the Start Communications command to a DanLoad 6000 that is offline in order to bring it back online when it is available.)

The automation system establishes communications with the DanLoad 6000 using the Start Communications command. This initializes the DanLoad 6000's alternating sequence function code to the function code in the Start Communications command (either 41h or 42h). The next (non-retry) query from the automation system should use the other function code. Operations at the DanLoad 6000 are unaffected by the Start Communications command.

The automation system can request the date and time that the DanLoad 6000 was last powered down using the Request Power Fail Date and Time command.

The automation system can request the transaction numbers of the transactions that are stored in the DanLoad 6000's non-volatile transaction storage memory using the Request Stored Transaction Numbers command. (The automation system may wish to retrieve stored transactions from the DanLoad 6000.)

5.2 Auto/Manual Change-Over

The DanLoad 6000's operating mode is "automatic" or "manual". The upper (yellow) LED to the right of the DanLoad 6000's display indicates "automatic" (on) or "manual" (off). The automation system can communicate with the DanLoad 6000 in either operating mode provided that one of the DanLoad 6000's serial ports is configured for this function.

If an "Auto/manual change-over" input is configured (program code 343) the DanLoad 6000's operating mode is determined solely by the state of that input.

If an "Auto/manual change-over" input is not configured the DanLoad 6000's operating mode is determined by communications or by setup (program code 025). The automation system can change the DanLoad 6000's operating mode between manual and automatic using the Change Operating Mode command. The Change Operating Mode command is the communications equivalent of the auto/manual change-over input. The operating mode is changed in exactly the same way as if the auto/manual change-over input was used.

If the auto/manual change-over input changes state, or the Change Operating Mode command is received indicating a change of operating mode, a batch or transaction in progress are ended before the operating mode changes. When the DanLoad 6000's operating mode changes to manual it displays the recipe selection screen, the additive selection screen or the loading screen depending on the configuration.

5.3 Communications Failure

In the automatic operating mode, the DanLoad 6000 maintains a timer per communications channel which is reset each time a correctly framed message is received on that channel. (This is not the same timer used to frame Modbus RTU messages.)

A "Comms failure channel A" occurs when the DanLoad 6000 does not receive a correctly framed message with its address on channel "A" for the configured number of seconds when communications on channel "A" are "up" in automatic mode.

Similarly for "Comms failure channel B".

When a "Comms failure channel A" or "Comms failure channel B" occurs the DanLoad 6000 raises a primary alarm. A batch in progress to stopped. (Standard primary alarm processing.) The DanLoad 6000 can be placed in the manual operating mode (the batch and transaction are ended), the alarm can be reset and loading can resume in manual. Alternatively, communications can be restarted using the Start Communications command, the alarm can be reset and loading can resume in automatic.

5.4 Transaction Authorization

The automation system can command the DanLoad 6000 to prompt the operator for data, e.g. customer number, order number, compartment number, using the Display Message command. The DanLoad 6000 sets a flag to indicate when keypad data is available. The automation system obtains the keypad data using the Request Keypad Data command.

While waiting for a particular status change on the DanLoad 6000 the automation system should be aware that another, unexpected event could occur that changes the outcome of the data exchange, e.g. a primary alarm is raised or the DanLoad 6000 times out waiting for input.

The DanLoad 6000 allows a preset volume of a product, blend or recipe to be loaded as a "batch". The operator or the automation system sets the (preset) volume. The batch begins when the "START" key is pressed. The batch ends when delivery of product has stopped and cannot be resumed (without beginning a new batch). When a preset volume of a multi-component sequential or in-line blend is loaded, this is just one batch.

On the DanLoad 6000, one or more batches are loaded on a single "transaction". Each transaction is assigned a 4-digit sequence number when it is started. Each batch is assigned a 4-digit sequence number when it is started. All batches on a transaction are for the same recipe. All batches on a transaction are with the same additives. All batches on a transaction share the same operator-entered "data items". A "transaction volume" is the sum of the associated "batch volumes".

A batch can start and end without any volume having been loaded. The automation system may choose to "discard" such batches, though this would result in apparently "missing" batch sequence numbers.

A recipe must be selected before a transaction can begin. If the DanLoad 6000 has only one recipe then it is selected by default. The automation system can command the DanLoad 6000 to prompt the operator for a recipe selection using the Prompt Recipe command. The DanLoad 6000 sets a flag to indicate that a recipe has been selected by the operator. The automation system requests the selected recipe from the DanLoad 6000 using the Request Selected Recipe command. If the automation system can determine the recipe without input from the operator, the Prompt Recipe and Request Selected Recipe commands are not required. If multiple recipes are available at the DanLoad 6000 the automation system may require a table to cross-reference the DanLoad 6000's recipe numbers with the automation system's "product codes". The flag that indicates that a recipe has been selected is cleared by the Authorize Transaction command, the Prompt Recipe command or the Clear Status command.

Additives can be selected before a transaction is started. Up to six additives can be selected. A selection of "no additives" is a valid additive selection. The automation system can command the DanLoad 6000 to prompt for an additive selection using the Prompt Additives command. A "suggested" additive selection can be displayed for the operator. The DanLoad 6000 sets a flag to indicate that an additive selection has been made by the operator. The automation system requests the additive selection from the

DanLoad 6000 using the Request Selected Additives command. If the automation system can determine the additive selection without input from the operator, the Prompt Additives and Request Selected Additives commands are not required. The flag that indicates that an additive selection has been made is cleared by the Authorize Transaction command, the Prompt Additives command or the Clear Status command.

The automation system "authorizes" a transaction using the Authorize Transaction command. The authorization command sets the recipe, additives, side, etc. The DanLoad 6000 sets a flag to indicate that a transaction has been authorized. The Authorize transaction command also causes the loading screen to be displayed (if it is not already displayed). The transaction is not actually started ("in progress") until the first batch is started.

5.5 Batch Authorization

The automation system can command the DanLoad 6000 to prompt for a preset volume using the Prompt Preset Volume command. Optionally, the Prompt Preset Volume command includes a "suggested preset volume" which can be entered/accepted or cleared and modified (up or down) by the operator. The DanLoad 6000 sets a flag to indicate that preset volume has been entered. The automation system requests the preset volume from the DanLoad 6000 using the Request Preset Volume command. If the preset volume is valid, the automation system can set the preset volume using the Authorize Batch command. If the automation system can determine the batch volume without input from the operator, the Prompt Preset Volume and Request Preset Volume commands are not required. The flag that indicates that a preset volume has been entered is cleared by a Prompt Preset Volume command or a Clear Status command.

The DanLoad 6000 "presets the batch" and displays "Press START when ready or STOP to cancel" when the Authorize Batch command is received. The batch is started when the operator presses the "START" key or the Start Batch command is received. The DanLoad 6000 sets a flag to indicate that the batch has been started and assigns a batch sequence number (which rolls from 9999 to 0). If the batch is aborted, e.g. the operator presses the "STOP" key (instead of the "START" key) or the DanLoad 6000 times out waiting for the "START" key to be pressed, the DanLoad 6000 sets a flag to indicate that the batch has been aborted.

When a batch has been started the DanLoad 6000 sets a flag to indicate that a batch is in progress.

5.6 End of Batch

The batch delivery is stopped when the operator presses the "STOP" key or the Stop Batch command is received from the automation system or a primary or secondary alarm occurs. A batch that is stopped in this way may be restartable. The DanLoad 6000 sets a flag to indicate that the batch has stopped and is restartable.

A batch ends when the preset volume has been delivered or an End Batch command is received from the automation system. A batch may also end if it has been stopped and certain other conditions apply. (See §9.2.) A batch that has ended cannot be restarted. The DanLoad 6000 sets a flag to indicate that the batch has ended.

The automation system can obtain batch data (batch volumes, component temperatures, batch sequence number, transaction sequence number, swing-arm side, etc.) for a batch that has ended using the Batch Data by Component command, . The flag that indicates that a batch has ended and cannot be restarted is cleared by the Clear Status command.

5.7 End of Transaction

In the automatic operating mode a transaction is ended by an End Transaction command from the automation system when there is not a batch in progress, by powering down the DanLoad 6000 or by change-over to manual. The operator can press the "STOP" key at the DanLoad 6000 to request the automation system to end the transaction. When it receives an End Transaction command the DanLoad 6000 sets a flag to indicate that the transaction has ended. The automation system can request transaction data from the DanLoad 6000 using the Transaction Data by Component command.

5.8 Swing-Arm Processing (Double-Side Load Racks)

On a double-sided loading rack it is possible that drivers on both sides of the load rack are authorized to use the same DanLoad 6000. The automation system can determined the current swing-arm side from the DanLoad 6000 using the Request Status command or the Swing-arm side command (the DanLoad 6000 has dedicated swing-arm inputs on its CPU board) or from other inputs to the automation system independent of the DanLoad 6000.

The automation system should send commands to control authorization based on the current swing-arm side. The swing-arm side can change at almost any time. If the side changes, further commands should be transmitted. If a transaction is in progress on side

1 and the side changes to side 2 a Stop Batch/End Batch/End Transaction command should be sent (to end the transaction on side 1) and an Authorized Transaction command (for example) could be sent to authorize a transaction on side 2.

5.9 Manual Operating Mode

Generally speaking, in the manual operating mode an automation system can request data from the DanLoad 6000, e.g. request data code values, but cannot control its operation, e.g. prompt for preset volume.

The transaction data by component (1Fh) command is available in the manual and automatic operating modes. However, its use in the manual operating mode is not recommended; the transaction data by component command is valid only for the most recently ended transaction, and one transaction may end and another become authorized faster than an automation system can respond (particularly if the configuration is such that the DanLoad 6000 stays in the loading screen when a transaction is ended).

When implementing an automation system interface with the DanLoad 6000 in the manual operating mode, i.e. monitoring only, the automation system should make use of the DanLoad 6000's transaction storage facility. The automation system should poll the status of transaction storage (command code 1Eh) and request batches (command code 3Ah) and transactions (command code 39h) as they are stored by the DanLoad 6000. The automation system needs to be aware of the formats of the batch and/or transaction records, which are user-configurable. The automation system can derive the batch and/or transaction record formats by requesting (using command code 22h) the configured storage code offsets given by program codes 840 through 902. (Refer to the DanLoad 6000 Reference Manual for details.)

This page intentionally left blank.

6 Implementing the Interface

The following information and examples may be helpful in implementing the interface. A "DanLoad 6000 Automation System Communications Support Diskette" is available on request from Automation Sales (Tel. 713-467 6000).

6.1 Constants (All Versions)

The following constants are used by the DanLoad 6000:

```
////
/* Maximum # users. */
#define MAXUSER 8

/* Maximum # meters. */
#define MAXMTR 4

/* Maximum # additives. */
#define MAXADDS 6

/* Maximum # recipes. */
#define MAXRECIPES 30

/* Maximum # components. */
#define MAXCOMPS 4

/* Maximum # data items. */
#define MAXDATAITEMS 5

/* Maximum # safety circuits. */
#define MAXSAFETY 8

/* Maximum # T/Pe pairs for Pe linearization. */
#define MAXPELIN 3

/* Maximum # transaction storage codes. */
#define MAXTSC 36

/* Maximum # batch storage codes. */
#define MAXBSC 22 /* v3.00 */
#define MAXBSC 23 /* v4.00 and above. */

/* Maximum batch sequence number before rolling over to 0. */
#define MAXBATCHSEQ 9999

/* Maximum transaction sequence number before rolling over to 0. */
#define MAXTRANSEQ 9999

/* Maximum # program code. */
#define MAXPC 831 /* v2.00 */
#define MAXPC 981 /* v3.00 */
#define MAXPC 982 /* v4.00 and above. */

/* # bytes required to store one bit for each program code. */
#define NUMPCBYTES 104 /* v2.00 */
#define NUMPCBYTES 123 /* v3.00 and above. */

/* Maximum # data codes. */
#define MAXDC 129 /* v2.00 */
#define MAXDC 134 /* v3.00 */
#define MAXDC 146 /* v4.00 */
#define MAXDC 222 /* v5.00 */
#define MAXDC 231 /* v5.11 */
////
```

6.2 DanLoad 6000 v4.00 Database

The "database" is the name given to the stored program code values. The database for DanLoad 6000 v4.00 is shown below. (A "byte" is an unsigned char.)

```
////

/*
**
** Database
*/

struct {
    long passcode;
    char userid[17];
    byte pwacess;
} pw[MAXUSER];

int  nummtrs;
struct {
    char mtrid[6];
    int  mpulsin;
    int  mtrvlv;
    int  radjfac; /* Ratio adjustment factor. */
    int  fadjfac; /* Flow adjustment factor. */
    int  ramppct; /* Ramp percentage. */
} mtrparms[MAXMTR];

byte opmode;
byte language;
byte dtefmt;
byte dtesep;
byte decsep;
byte units;
byte numdataprompts; /* Number of data prompts. */
char dataprompt[MAXDATAITEMS][33];
int  pmt_timout;
byte chk_disp;
byte unit_type;
byte fcv_type;
byte dspdr;
int  transeqnum;
int  batchseqnum;
byte roundmthd; /* Rounding method. */
int densscale; /* Density/gravity scale. */
int maxbatch; /* Maximum batches/transaction. */

int  numvalves;
int initflwtime; /* Initial flow time. */

/* Component parameters */
int numcomps;
struct {
    char compid[17];
    int compmtr;
    byte compavail;
} comp[MAXCOMPS];

/* Delivery parameters */
```

```
long maxpreset;
long minpreset;
byte predeltype;
byte deldisptype;
byte stopkey;
long fallbackvol;
int maxblenddev;

struct Compdel {
    long loflwstart;
    long loflwrestart;
    long loflwstop;
    int  linepackdly;
    int  pumpstopdly;
    int  blkvalvedly;
} compdel[MAXCOMPS];
int maxrampclicks;
int maxmaintclicks;
int addpmpdly; /* Additive pump stop delay. */
int primcmp;

/* Digital Valve Parameters */
struct {
    int dvlferrpct;
    int dvhferrpct;
    int dvdly;
    byte openmthd;
} dvparms[MAXVLV];

/* Pulse per unit parameters */
int numdaus;
struct Dau {
    struct Dauctl {
        byte gross;
        byte net;
    } dauctl[MAXMTR]; /* 2 * MAXMTR bytes for each program code. */
    int daufactor;
    int daupulwidth;
} dau[MAXDAU];

/* Additive Parameters */
int numadds;
byte addprompt;
byte addtotmthd;
int addclnlinvol; /* Was "injlowflw". */
byte addunits;
struct Add {
    struct Addctl {
        byte gross;
        byte net;
    } addctl[MAXMTR]; /* 2 * MAXMTR bytes for each program code. */
    int ratiovol;
    int offsetvol;
    long totvol; /* Volume per pulse or K-factor. */
    /*** byte addavail; ***/
    long reqdvol; /* Required volume of additive per 1000 of product for
                  ** additive volume verification.
                  */
} add[MAXADDS];

/* Factors  */
int numfacs;
```

```
byte mfacmthd;
struct Fac {
    int nomkfac;
    long mstrmfac;
    /* byte tempcomp; */
    struct Mtrfac {
        long flwrate;
        long mfac;
    } mtrfac[MAXFACS];
} fac[MAXCOMPS];
int mstrmfacpct;
int adjmfacpct;
byte pwpertr; /* Password (passcode) per transaction. */

/* Alarm Parameters */
int secalrst;
byte lowflwact;
long lowflwlim;
int lowflwtime;
byte hiflwact;
long hiflwlim;
int hiflwtime;
int ovrunlim;
byte undrflwact;
int undrflwlim;
byte noflwact;
int noflwtime;
int unauthflwlim;
int pserr;
long psresetcnt;
byte dlfailact;
byte commfailact;
int commtimout;
byte tempfailact;
int tempmin;
int tempmax;
byte densfailact;
long densmin;
long densmax;
long presmin;
long presmax;
int adderrlim;
int feedbacktime;
int bvclsdly;

struct Safety {
    byte action;
    char msg[33];
} safety[MAXSAFETY];

byte sc5type;
byte sc6type;
byte sc7type;
byte sc8type;

int praddminpct; /* Additive volume verification minimum error percentage. */
int praddmaxpct; /* Additive volume verification maximum error percentage. */

byte storeact;
byte poweract;
byte rampact; /* Ramp down alarm action. */
int ramptime; /* Ramp down time. */
```

```
/* I/O Parameters */
byte islot[9];

struct Ioctl mtrin[MAXMTRIN];
struct Ioctl rtdin[MAXRTDIN];
struct Ioctl anain[MAXANAIN];
struct Ioctl digin[MAXDIGIN];
struct Ioctl digout[MAXDIGOUT];

struct Almout {
    int out;
    unsigned long mask;
} almout[MAXALMOUT];

int pmpctlout[MAXCOMPS];
struct Vlvio {
    int sol1;
    int sol2;
    int stemsw1;
    int stemsw2;
    int clsinp;
} vlvio[MAXVLV];
byte sidemthd;
int dauout[MAXDAU];
int trip1out;
int trip2out;
int trip3out;
struct Mtrtmp {
    int mtempinp;
    long moffset;
    int mstrtstop;
    int mpresinp;
    long pres4;
    long pres20;
    int mdensinp;
    long dens4;
    long dens20;
} mtrtmp[MAXMTR];

int startend;
int autochng;
int primalrst;

int safetyinp[MAXSAFETY];
byte temprange;
int opdly; /* Delay after outputs, e.g. for DART 2. */

byte pridsptyp;
int priminlt;
int primaxlt;
byte secdsptyp;
int secminlt;
int secmaxlt;

int recipeinp[MAXRECIN]; /* Recipe selection inputs. */
int recipeout[MAXRECOUT]; /* Recipe selection outputs. */

struct Addinp {
       int ratio;
       int feedback;
       int select;
    int pmpctlout;      /* Additive pump control output. */
```

```
    int bvout;          /* Additive block valve output. */
} addinp[MAXADDS];

int addclnlinout; /* Additive clean line output. */

struct Cmpio {
       int bvout;
       int bvin;
} cmpio[MAXCOMPS];
int combio[15];

byte tempunits;
byte densunits;
byte presunits;
int reftemp;
long tempsmpl;

struct Cmptemp {
       byte option;
       long alpha;
       int backup;
} cmptemp[MAXCOMPS];
struct Cmppres {
       byte option;
       long pffac;
       long vp;
} cmppres[MAXCOMPS];
struct Cmpdens {
       byte option;
       long backup;
} cmpdens[MAXCOMPS];

int numrecipes;
struct Recipe {
       char name[17];
       int cmppct[MAXCOMPS];
    struct Cmpseq {
        int cmpno;
    } cmpseq[MAXCOMPS];
} recipe[MAXRECIPES];

struct Commun {
       byte mode;
       byte baud;
       byte dbits;
       byte sbits;
       byte parity;
} commun[2];

int ddpc[16];

byte batch;
byte trans;
byte almlog;
byte pflog;
byte progmode;
byte wmsw;
byte pcchnglog;
byte provelog;
byte crash;
byte totlog;
byte trtkauto;
```

```
byte trtk;
byte recipethru;
byte dlogseq;

/* Blending. */
int rampdownpct; /* The percentage of the blend's current flow rate, etc. */
int deltarate; /* Change in flow rate for batch fall back. */
long loflwstart;
int clnlinvol; /* Clean line volume. */
int lopropfac; /* The meter factor flow rate for low proportion components. */
int blchkvol; /* Volume after which blend ratio adjustment begins. */
int blchkvol2; /* Volume after which blend ratio alarm is checked. */
int vadjust; /* Volume over which blend adjustment is made. */
byte comppctmthd;
byte blenderrmthd;
int blnddev;
int blndtol;
int blnddevvol;
int blndtolvol;
int blendsmpl;
long stoprate[MAXCOMPS];
long rflwrate[MAXRFLWRATE][2]; /* Low and high recipe flow rates. */

byte pttype; /* Pressure transmitter types. */
byte pemthd; /* Pe method. */
byte pinchmthd; /* Pinch back method. */
long pinchpres; /* Pinch back pressure. */
long atmospres; /* Atmospheric pressure. */
struct Pe_lin {
    int temp;
    long pres;
} pe_lin[MAXPELIN]; /* T/Pe pairs for Pe linearization. */

/* Transaction storage. */

byte trrecl; /* Transaction record length. */
int trnumpgs; /* Transaction file start page (/ 100) and # pages (% 100). */
byte barecl; /* Batch record length. */
int banumpgs; /* Batch file start page (/ 100) and # pages (% 100). */
byte trtkfmt; /* Transaction ticket predefined format. */

/* Transaction storage code configuration. */
struct Sccfg tsccfg[MAXTSC];

/* Batch storage code configuration. */
struct Sccfg bsccfg[MAXBSC];

/*
**
** Database
***************
*/

////
```

6.3 DanLoad 6000 v5.00 Database

The "database" is the name given to the stored program code values. The database for DanLoad 6000 v5.00 is shown below. (A "byte" is an unsigned char.)

```
////

/*
**
** Database
*/

struct {
    long passcode; /* PC 1, etc.  User X 9-digit passcode. */
    char userid[17]; /* PC 2, etc.  User X NUL terminated ID. */
    byte pwacess; /* PC 3, etc.  User X supervisor privilege. */
} pw[MAXUSER];

int nummtrs; /* PC 50.  Number of meters. */
struct {
    char mtrid[6]; /* PC 51, etc.  Meter X ID. */
    int mtrvlv; /* PC 52, etc.  Meter X flow control valve. */
    int radjfac; /* PC 729, etc.  Meter X ratio adjustment factor. */
    int fadjfac; /* PC 728, etc.  Meter X flow adjustment factor. */
    int ramppct; /* PC 730, etc.  Meter X ramp percentage. */
    int fpo; /* PC 756, etc.  Meter X MPMC factored pulse output. */
    int numblades; /* PC 757, etc.  Meter X CALMON number of blades. */
    long maxmaxchardev; /* PC 758, etc.  Meter X CALMON max. max. characteristic deviation. */
    long maxtotchardevs; /* PC 759, etc.  Meter X CALMON max. total characteristic deviations.
*/
    unsigned char calibsmplstat; /* PC 760, etc.  Meter X calibration sample status. */
} mtrparms[MAXMTR];

byte opmode; /* PC 25.  Operating mode. */
byte language; /* PC 28.  Language. */
byte dtefmt; /* PC 38.  Date format. */
byte dtesep; /* PC 39.  Date separator. */
byte decsep; /* PC 40.  Decimal separator. */
byte units; /* PC 29.  Product units. */
byte numdataitems; /* PC 30.  Number of data items. */
char dataprompt[5][33]; /* PC 31, etc.  Data item X NUL terminated prompt. */
int pmt_timout; /* PC 36.  Prompt time-out. */
byte chk_disp; /* PC 37.  Check display. */
byte unit_type; /* PC 26.  Unit type. */
byte fcv_type; /* PC 27.  Valve type. */
unsigned char dspdr; /* PC 41.  Display data rate. */
int transeqnum; /* PC 42.  Transaction #. */
int batchseqnum; /* PC 43.  Batch #.  */
unsigned char roundmthd; /* PC 45.  Rounding method. */
int densscale; /* PC 46.  Density/gravity scale. */
int maxbatch; /* PC 47.  Maximum batches/transaction. */

int numvalves; /* PC 48.  Number of (flow control) valves. */
int initflwtime; /* PC 49.  Initial flow time. */

/* Component parameters */
int numcomps; /* PC 65.  Number of components. */
struct {
    char compid[17]; /* PC 66, etc.  Component X NUL terminated ID. */
```

```
    int compmtr; /* PC 67, etc.  Component X meter number. */
    byte compavail; /* PC 68, etc.  Component X available. */
} comp[MAXCOMPS];

/* Delivery parameters */
long maxpreset; /* PC 78.  Maximum preset volume. */
long minpreset; /* PC 79.  Minimum preset volume. */
byte predeltype; /* PC 80.  Preset/delivery type. */
byte deldisptype; /* PC 81.  Delivery display type. */
byte stopkey; /* PC 82.  Stop key action. */
long fallbackvol; /* PC 83.  Fall back volume. */

struct Compdel {
    long loflwstart; /* PC 84, etc.  Component X low flow start volume. */
    long loflwrestart; /* PC 85, etc.  Component X low flow restart volume. */
    long loflwstop; /* PC 86, etc.  Component X low flow stop volume. */
    int linepackdly; /* PC 87, etc.  Component X line pack delay. */
    int pumpstopdly; /* PC 88, etc.  Component X pump stop delay. */
    int blkvalvedly; /* PC 89, etc.  Component X block valve delay. */
} compdel[MAXCOMPS];
int maxrampclicks; /* PC 108.  Ramp clicks. */
int maxmaintclicks; /* PC 109.  Maintenance clicks. */
int addpmpdly; /* PC 110.  Additive pump stop delay. */
int primcmp; /* PC 111.  Primary component. */

/* Digital Valve Parameters */
struct {
    int dvlferrpct; /* PC 112, etc.  Valve X low flow % error. */
    int dvhferrpct; /* PC 113, etc.  Valve X high flow % error. */
    int dvdly; /* PC 114, etc.  Valve X maint click adjustment. */
    byte openmthd; /* PC 115, etc.  Valve X open method. */
} dvparms[MAXVLV];

/* Pulse per unit (DAU) parameters */
int numdaus; /* PC 128.  Number of PPU's. */
struct Dau {
    struct Dauctl {
        byte gross;
        byte net;
    } dauctl[MAXMTR]; /* PC 129, etc.  PPU X control meters. */
    int daufactor; /* PC 130, etc.  PPU X factor. */
    int daupulwidth; /* PC 131, etc.  PPU X pulse width. */
} dau[MAXDAU];

/* Additive Parameters */
int numadds; /* PC 135.  Number of additives. */
byte addselmthd; /* PC 136.  Additive selection method. */
int injpct; /* PC 137.  Additive inject % */
int addclnlinvol; /* PC 138.  Additive clean line volume. */
byte addunits; /* PC 44. Additive units. */
struct Add {
    struct Addctl {
        byte gross;
        byte net;
    } addctl[MAXMTR]; /* PC 139, etc.  Additive X control meters. */
    int ratiovol; /* PC 140, etc.  Additive X ratio volume in whole units. */
    unsigned char injmthd; /* PC 141, etc.  Additive X injection method. */
    long kfac; /* PC 142, etc.  Additive X volume per pulse or K-factor in
               ** hundredths.
               */
    long reqdvol; /* PC 143, etc.  Additive X volume (in external product
                  ** units x 10,000) per 1000 units of product.
```

```
                */
} add[MAXADDS];

/* Factors  */
int numfacs; /* PC 169.  Number of factors/component. */
byte mfacmthd; /* PC 170.  Meter factor method. */
struct Fac {
    long nomkfac; /* PC 171, etc.  Component X nominal K-factor. */
    long mstrmfac; /* PC 172, etc.  Component X master meter factor. */
    struct Mtrfac {
        long flwrate; /* PC 174, etc.  Component X flow rate Y. */
        long mfac; /* PC 175.  Component X meter factor Y. */
    } mtrfac[MAXFACS];
} fac[MAXCOMPS];
int mstrmfacpct; /* PC 215.  Master meter factor percentage. */
int adjmfacpct; /* PC 216.  Adjacent meter factor percentage. */
unsigned char pwpertr; /* PC 217.  Password (passcode) per transaction. */

/* Alarm Parameters */
int secalrst; /* PC 220.  Secondary alarm reset (secs). */
byte lowflwact; /* PC 221.  Low flow alarm action. */
long lowflwlim; /* PC 222.  Minimum flow rate. */
int lowflwtime; /* PC 223.  Low flow time. */
byte hiflwact; /* PC 224.  High flow alarm action. */
long hiflwlim; /* PC 225.  Maximum flow rate. */
int hiflwtime; /* PC 226.  High flow time. */
int ovrunlim; /* PC 227.  Overrun limit volume. */
byte undrflwact; /* PC 228.  Underflow alarm action. */
int undrflwlim; /* PC 229.  Underflow limit volume. */
byte noflwact; /* PC 230.  No flow alarm action. */
int noflwtime; /* PC 231.  No flow time. */
int unauthflw; /* PC 232.  Unauth. flow limit volume. */
int pserr; /* PC 233.  Pulse security error limit. */
long psresetcnt; /* PC 234.  Pulse security reset count. */
byte dlfailact; /* PC 235.  Data logging alarm action. */
byte commfailact; /* PC 236.  Comms fail alarm action. */
int commtimout; /* PC 237.  Comms fail time-out. */
byte tempfailact; /* PC 238.  Temperature fail alarm action. */
int tempmin; /* PC 239.  Min. temperature. */
int tempmax; /* PC 240.  Max. temperature. */
byte densfailact; /* PC 241.  Density fail alarm action. */
long densmin; /* PC 242.  Minimum density/gravity. */
long densmax; /* PC 243.  Maximum density/gravity. */
long presmin; /* PC 244.  Minimum pressure. */
long presmax; /* PC 245.  Maximum pressure. */
int adderrlim; /* PC 246.  Additive error limit. */
int addfbckcnt; /* PC 247.  # pulses per additive cycle or # seconds. */
int bvclsdly; /* PC 248.  Block valve time. */

struct Safety {
    byte action; /* PC 249, etc.  Safety circuit X alarm action. */
    char msg[33]; /* PC 250, etc.  Safety ciruit X alarm message. */
} safety[MAXSAFETY];

byte sc5type; /* PC 265.  Safety circuit 5 type. */
byte sc6type; /* PC 266.  Safety circuit 6 type. */
byte sc7type; /* PC 267.  Safety circuit 7 type. */
byte sc8type; /* PC 268.  Safety circuit 8 type. */

long unauthaddflw; /* PC 271.  Unauthorized additive flow volume. */
int praddminpct; /* PC 272.  Additive volume verification minimum error
                ** percentage.
```

```
                */
int praddmaxpct; /* PC 273.  Additive volume verification maximum error
                ** percentage.
                */

byte storeact; /* PC 274.  Storage alarm action. */
byte poweract; /* PC 275.  Power fail alarm action. */
byte rampact; /* PC 276.  Ramp down alarm action. */
int ramptime; /* PC 277.  Ramp down time. */
int add1000errlim; /* PC 278.  Additive per 1000 error limit. */
int endbatim; /* PC 279.  End batch time. */

/* I/O Parameters */
unsigned char islot[9]; /* Slots J1 thru J7, DUART and ARCNET. */

struct Ioctl mtrin[MAXMTRIN];
struct Ioctl rtdin[MAXRTDIN];
struct Ioctl anain[MAXANAIN];
struct Ioctl digin[MAXDIGIN];
struct Ioctl digout[MAXDIGOUT];

struct Almout {
    int out; /* PC 287, etc.  Alarm output X. */
    unsigned long mask; /* PC 269, etc.  Alarm output X mask. */
} almout[MAXALMOUT];

int pmpctlout[MAXCOMPS]; /* PC 288, etc.  Component X pump output. */
struct Vlvio {
    int sol1; /* PC 292, etc.  Valve X solenoid 1. */
    int sol2; /* PC 293, etc.  Valve X solenoid 2. */
    int stemsw1; /* PC 294, etc.  Valve X stem switch 1. */
    int stemsw2; /* PC 295, etc.  Valve X stem switch 2. */
    int clsinp; /* PC 296, etc.  Valve X close input. */
} vlvio[MAXVLV];
byte sidemthd; /* PC 312.  Side detect method. */
int dauout[MAXDAU]; /* PC 313, etc.  PPU output X. */
int trip1out; /* PC 315.  Trip 1 output. */
int trip2out; /* PC 316.  Trip 2 output. */
int trip3out; /* PC 317.  Trip 3 output. */
struct Mtrtmp {
    int mtempinp; /* PC 318, etc.  Meter X temperature input. */
    long moffset; /* PC 319, etc.  Meter X offset (Ohms). */
    int mpresinp; /* PC 320, etc.  Meter X pressure input. */
    long pres4; /* PC 464, etc.  Meter X pressure at 4 mA. */
    long pres20; /* PC 465, etc.  Meter X pressure at 20 mA. */
    int mdensinp; /* PC 321, etc.  Meter X density input. */
    long dens4; /* PC 466, etc.  Meter X density at 4 mA. */
    long dens20; /* PC 467, etc.  Meter X density at 20 mA. */
    int flowinp; /* PC 322, etc.  Meter X flow input. */
} mtrtmp[MAXMTR];

int startend; /* PC 342.  Off-rack start/end input. */
int autochng; /* PC 343.  Auto/manual change-over input. */
int primalrst; /* PC 344.  Primary alarm reset input. */

int safetyinp[MAXSAFETY]; /* PC 345, etc.  Safety circuit X input. */
byte temprange; /* PC 353.  RTD inputs type. */
int opdly; /* PC 354.  Delay after outputs, e.g. for DART 2. */

byte pridsptyp; /* PC 355.  Primary display type. */
int priminlt; /* PC 356.  Primary display min. light. */
int primaxlt; /* PC 357.  Primary display max. light. */
```

```
byte secdsptyp; /* PC 358.  Secondary display type. */
int secminlt; /* PC 359.  Secondary display min. light. */
int secmaxlt; /* PC 360.  Secondary display max. light. */

int recipeinp[MAXRECIN]; /* PC 361, etc.  Recipe X selection input. */
int recipeout[MAXRECOUT]; /* PC 367, etc.  Recipe X selection output. */

unsigned char fpobasis; /* PC 752.  Factored pulse outputs basis. */
int calmonerr; /* PC 753.  Calibration monitoring error limit. */
int calmonrcnt; /* PC 754.  Calibration monitoring reset count. */

struct Addinp {
       int ratio; /* PC 380, etc.  Additive X ratio output. */
       int feedback; /* PC 381, etc.  Additive X feedback input. */
       int select; /* PC 382, etc.  Additive X selection input. */
    int pmpctlout; /* PC 800, etc.  Additive X pump control output. */
    int bvout; /* PC 801, etc.  Additive X block valve output. */
} addinp[MAXADDS];

int addclnlinout; /* PC 398.  Additive clean line output. */

struct Cmpio {
       int bvout; /* PC 400, etc.  Component X block valve output. */
       int bvin; /* PC 401, etc.  Component X block valve input. */
} cmpio[MAXCOMPS];
int combio[15]; /* PC 408, etc.  Component combination X. */
int endbaout; /* PC 423.  End batch output. */
int endbain; /* PC 424.  End batch input. */

byte tempunits; /* PC 427.  Temperature units. */
byte densunits; /* PC 428.  Density units. */
byte presunits; /* PC 429.  Pressure units. */
int reftemp; /* PC 430.  Reference temperature. */
long tempsmpl; /* PC 431.  Sample volume. */

struct Cmptemp {
    byte option; /* PC 432, etc.  Component X temperature correction option. */
    long alpha; /* PC 433, etc.  Component X alpha. */
    int backup; /* PC 434, etc.  Component X backup temperature. */
} cmptemp[MAXCOMPS];
struct Cmppres {
    byte option; /* PC 444, etc.  Component X pressure correction option. */
    long pffac; /* PC 445, etc.  Component X F-factor. */
    long backup; /* PC 446, etc.  Component X backup pressure. */
} cmppres[MAXCOMPS];
struct Cmpdens {
    byte option; /* PC 456, etc.  Component X density option. */
    long backup; /* PC 457, etc.  Component X backup density/gravity. */
} cmpdens[MAXCOMPS];

int numrecipes; /* PC 480.  Number of recipes. */
struct Recipe {
    char name[17]; /* PC 481, etc.  Recipe X NUL terminated name. */
    int cmppct[MAXCOMPS]; /* PC 482, etc.  Recipe X component percentages. */
    struct Cmpseq {
        int cmpno;
    } cmpseq[MAXCOMPS]; /* PC 486, etc.  Recipe X sequence. */
} recipe[MAXRECIPES];

struct Commun {
    byte mode; /* PC 663, etc.  Channel X mode. */
    byte baud; /* PC 664, etc.  Channel X data rate. */
```

```
    byte dbits; /* PC 665, etc.  Channel X word size. */
    byte sbits; /* PC 666, etc.  Channel X stop bits. */
    byte parity; /* PC 667, etc.  Channel X parity. */
} commun[2];
int ddpc[16]; /* PC 680, etc.  Dynamic data display element X data code. */
byte batch; /* PC 696.  Batch summary. */
byte trans; /* PC 697.  Transaction summary. */
byte almlog; /* PC 698.  Alarm log. */
byte pflog; /* PC 699.  Power fail log. */
byte progmode; /* PC 700.  Program mode entry/exit. */
byte wmsw; /* PC 701.  W&M switch opened/closed. */
byte pcchnglog; /* PC 702.  Program value change. */
byte cfglog; /* PC 703.  Configuration summary data log. */
byte crash; /* PC 704.  Crash memory summary. */
byte totlog; /* PC 705.  Totalizers. */
byte trtkauto; /* PC 706.  Transaction ticket. */
byte trtk; /* PC 708.  Transaction ticket reprint. */
byte thrulog; /* PC 708.  Thruput. */
byte dlogseq; /* PC 709.  Sequence numbers. */
int cuthour; /* PC 710.  Cutoff hour. */

/* Blending. */
int rampdownpct; /* PC 711.  The percentage of the blend's current flow rate, etc. */
int deltarate; /* PC 712.  Change in flow rate for batch fall back. */
long loflwstart; /* PC 713.  Low flow start volume for in-line blend. */
int clnlinvol; /* PC 714.  Clean line volume. */
int lopropfac; /* PC 715.  The meter factor flow rate for low proportion
                ** components of an in-line blend.
                */
int blchkvol; /* PC 716.  Correct after volume.  Volume after which blend
               ** ratio adjustment begins.
               */
int blchkvol2; /* PC 717.  Alarm after volume.  Volume after which blend
                ** ratio alarm is checked.
                */
int vadjust; /* PC 718.  Volume over which blend adjustment is made. */
unsigned char comppctmthd; /* PC 719.  Component % display. */
unsigned char blenderrmthd; /* PC 722.  Blend error method. */
int blnddev; /* PC 723.  Max. dev. % */
int blndtol; /* PC 724.  Blend tol. % */
int blnddevvol; /* PC 725.  Max. dev. volume. */
int blndtolvol; /* PC 726.  Blend tol. volume. */
int blendsmpl; /* PC 727.  Blend sample volume. */
long stoprate[MAXCOMPS]; /* PC 173.  Component X stop rate for in-line blend. */
long rflwrate[MAXRFLWRATE][2]; /* PC 776, 777, etc.  Recipe "group" low and
                               ** high flow rates.
                               */

unsigned char pttype; /* PC 818.  Pressure transmitter types. */
unsigned char pemthd; /* PC 819.  Pe method. */
unsigned char pinchmthd; /* PC 820.  Pinch back method. */
long pinchpres; /* PC 821.  Pinch back pressure. */
long atmospres; /* PC 822.  Atmospheric pressure. */
struct Pe_lin {
    int temp; /* PC 824, etc.  Tx. */
    long pres; /* PC 825, etc.  Pe at Tx. */
} pe_lin[MAXPELIN]; /* T/Pe pairs for Pe linearization. */

/* Transaction storage. */

unsigned char trrecl; /* PC 834.  Transaction record length. */
int trnumpgs; /* PC 835.  Transaction file start page (/ 100) and # pages (% 100). */
```

```
unsigned char barecl; /* PC 836.  Batch record length. */
int banumpgs; /* PC 837.  Batch file start page (/ 100) and # pages (% 100). */
unsigned char trtkfmt; /* PC 838.  Transaction ticket predefined format. */

/* Transaction storage code configuration. */
struct Sccfg tsccfg[MAXTSC]; /* PC 840, etc.  TSC X offset and row/column. */

/* Batch storage code configuration. */
struct Sccfg bsccfg[MAXBSC];/* PC 880, etc.  TSC X offset and row/column. */

/*
**
** Database
****************
*/

////
```

6.4 DanLoad 6000 v5.11 Database

The "database" is the name given to the stored program code values. The database for DanLoad 6000 v5.11 is shown below. (A "byte" is an unsigned char.)

```
////

/*
**
** Data Base
*/

struct {
    long passcode; /* PC 1, etc.  User X 9-digit passcode. */
    char userid[17]; /* PC 2, etc.  User X NUL terminated ID. */
    byte pwacess; /* PC 3, etc.  User X supervisor privilege. */
} pw[MAXUSER];

int nummtrs; /* PC 50.  Number of meters. */
struct {
    char mtrid[6]; /* PC 51, etc.  Meter X ID. */
    int mtrvlv; /* PC 52, etc.  Meter X flow control valve. */
    int radjfac; /* PC 729, etc.  Meter X ratio adjustment factor. */
    int fadjfac; /* PC 728, etc.  Meter X flow adjustment factor. */
    int ramppct; /* PC 730, etc.  Meter X ramp percentage. */
    int fpo; /* PC 756, etc.  Meter X MPMC factored pulse output. */
    int numblades; /* PC 757, etc.  Meter X CALMON number of blades. */
    long maxmaxchardev; /* PC 758, etc.  Meter X CALMON max. max. characteristic deviation. */
    long maxtotchardevs; /* PC 759, etc.  Meter X CALMON max. total characteristic deviations.
*/
    unsigned char calibsmplstat; /* PC 760, etc.  Meter X calibration sample status. */
    unsigned char calmonanalmthd; /* PC 731, etc.  Meter X CALMON analysis method. */
} mtrparms[MAXMTR];

byte opmode; /* PC 25.  Operating mode. */
byte language; /* PC 28.  Language. */
byte dtefmt; /* PC 38.  Date format. */
byte dtesep; /* PC 39.  Date separator. */
byte decsep; /* PC 40.  Decimal separator. */
byte units; /* PC 29.  Product units. */
byte numdataitems; /* PC 30.  Number of data items. */
char dataprompt[5][33]; /* PC 31, etc.  Data item X NUL terminated prompt. */
int pmt_timout; /* PC 36.  Prompt time-out. */
byte chk_disp; /* PC 37.  Check display. */
byte unit_type; /* PC 26.  Unit type. */
byte fcv_type; /* PC 27.  Valve type. */
unsigned char dspdr; /* PC 41.  Display data rate. */
int transeqnum; /* PC 42.  Transaction #. */
int batchseqnum; /* PC 43.  Batch #.  */
unsigned char roundmthd; /* PC 45.  Rounding method. */
int densscale; /* PC 46.  Density/gravity scale. */
int maxbatch; /* PC 47.  Maximum batches/transaction. */

int numvalves; /* PC 48.  Number of (flow control) valves. */
int initflwtime; /* PC 49.  Initial flow time. */

/* Component parameters */
int numcomps; /* PC 65.  Number of components. */
struct {
```

```
    char compid[17]; /* PC 66, etc.  Component X NUL terminated ID. */
    int compmtr; /* PC 67, etc.  Component X meter number. */
    long massadj; /* PC 68, etc.  Component X mass adjustment. */
} comp[MAXCOMPS];

/* Delivery parameters */
long maxpreset; /* PC 78.  Maximum preset volume. */
long minpreset; /* PC 79.  Minimum preset volume. */
byte predeltype; /* PC 80.  Preset/delivery type. */
byte deldisptype; /* PC 81.  Delivery display type. */
byte stopkey; /* PC 82.  Stop key action. */
long fallbackvol; /* PC 83.  Fall back volume. */

struct Compdel {
    long loflwstart; /* PC 84, etc.  Component X low flow start volume. */
    long loflwrestart; /* PC 85, etc.  Component X low flow restart volume. */
    long loflwstop; /* PC 86, etc.  Component X low flow stop volume. */
    int linepackdly; /* PC 87, etc.  Component X line pack delay. */
    int pumpstopdly; /* PC 88, etc.  Component X pump stop delay. */
    int blkvalvedly; /* PC 89, etc.  Component X block valve delay. */
} compdel[MAXCOMPS];
int maxrampclicks; /* PC 108.  Ramp clicks. */
int maxmaintclicks; /* PC 109.  Maintenance clicks. */
int addpmpdly; /* PC 110.  Additive pump stop delay. */
int primcmp; /* PC 111.  Primary component. */

/* Digital Valve Parameters */
struct {
    int dvlferrpct; /* PC 112, etc.  Valve X low flow % error. */
    int dvhferrpct; /* PC 113, etc.  Valve X high flow % error. */
    int dvdly; /* PC 114, etc.  Valve X maint click adjustment. */
    byte openmthd; /* PC 115, etc.  Valve X open method. */
} dvparms[MAXVLV];

/* Pulse per unit (DAU) parameters */
int numdaus; /* PC 128.  Number of PPU's. */
struct Dau {
    struct Dauctl {
        byte gross;
        byte net;
    } dauctl[MAXMTR]; /* PC 129, etc.  PPU X control meters. */
    int daufactor; /* PC 130, etc.  PPU X factor. */
    int daupulwidth; /* PC 131, etc.  PPU X pulse width. */
} dau[MAXDAU];

/* Additive Parameters */
int numadds; /* PC 135.  Number of additives. */
byte addselmthd; /* PC 136.  Additive selection method. */
int injpct; /* PC 137.  Additive inject % */
int addclnlinvol; /* PC 138.  Additive clean line volume. */
byte addunits; /* PC 44. Additive units. */
struct Add {
    struct Addctl {
        byte gross;
        byte net;
    } addctl[MAXMTR]; /* PC 139, etc.  Additive X control meters. */
    int ratiovol; /* PC 140, etc.  Additive X ratio volume in whole units. */
    unsigned char injmthd; /* PC 141, etc.  Additive X injection method. */
    long kfac; /* PC 142, etc.  Additive X volume per pulse or K-factor in
               ** hundredths.
               */
    long reqdvol; /* PC 143, etc.  Additive X volume (in external product
```

```
                 ** units x 10,000) per 1000 units of product.
                 */
} add[MAXADDS];

/* Factors  */
int numfacs; /* PC 169.  Number of factors/component. */
byte mfacmthd; /* PC 170.  Meter factor method. */
struct Fac {
    long nomkfac; /* PC 171, etc.  Component X nominal K-factor. */
    long mstrmfac; /* PC 172, etc.  Component X master meter factor. */
    struct Mtrfac {
        long flwrate; /* PC 174, etc.  Component X flow rate Y. */
        long mfac; /* PC 175.  Component X meter factor Y. */
    } mtrfac[MAXFACS];
} fac[MAXCOMPS];
int mstrmfacpct; /* PC 215.  Master meter factor percentage. */
int adjmfacpct; /* PC 216.  Adjacent meter factor percentage. */
unsigned char pwpertr; /* PC 217.  Password (passcode) per transaction. */
char produnitsmnemo[7]; /* PC 218.  Product units mnemonic. */
unsigned char userestart; /* PC 219.  Use restart quantity. */

/* Alarm Parameters */
int secalrst; /* PC 220.  Secondary alarm reset (secs). */
byte lowflwact; /* PC 221.  Low flow alarm action. */
long lowflwlim; /* PC 222.  Minimum flow rate. */
int lowflwtime; /* PC 223.  Low flow time. */
byte hiflwact; /* PC 224.  High flow alarm action. */
long hiflwlim; /* PC 225.  Maximum flow rate. */
int hiflwtime; /* PC 226.  High flow time. */
int ovrunlim; /* PC 227.  Overrun limit volume. */
byte undrflwact; /* PC 228.  Underflow alarm action. */
int undrflwlim; /* PC 229.  Underflow limit volume. */
byte noflwact; /* PC 230.  No flow alarm action. */
int noflwtime; /* PC 231.  No flow time. */
int unauthflw; /* PC 232.  Unauth. flow limit volume. */
int pserr; /* PC 233.  Pulse security error limit. */
long psresetcnt; /* PC 234.  Pulse security reset count. */
byte dlfailact; /* PC 235.  Data logging alarm action. */
int commtimout[2]; /* PC 236's and 237.  Comms fail time-out per channel. */
byte tempfailact; /* PC 238.  Temperature fail alarm action. */
int tempmin; /* PC 239.  Min. temperature. */
int tempmax; /* PC 240.  Max. temperature. */
byte densfailact; /* PC 241.  Density fail alarm action. */
long densmin; /* PC 242.  Minimum density/gravity. */
long densmax; /* PC 243.  Maximum density/gravity. */
long presmin; /* PC 244.  Minimum pressure. */
long presmax; /* PC 245.  Maximum pressure. */
int adderrlim; /* PC 246.  Additive error limit. */
int addfbckcnt; /* PC 247.  # pulses per additive cycle or # seconds. */
int bvclsdly; /* PC 248.  Block valve time. */

struct Safety {
    byte action; /* PC 249, etc.  Safety circuit X alarm action. */
    char msg[33]; /* PC 250, etc.  Safety ciruit X alarm message. */
} safety[MAXSAFETY];

byte sctype[MAXSAFETY]; /* PC 265, etc.  Safety circuit X type. */

long unauthaddflw; /* PC 271.  Unauthorized additive flow volume. */
int praddminpct; /* PC 272.  Additive volume verification minimum error
                 ** percentage.
                 */
```

```
int praddmaxpct; /* PC 273.  Additive volume verification maximum error
                 ** percentage.
                 */

byte storeact; /* PC 274.  Storage alarm action. */
byte poweract; /* PC 275.  Power fail alarm action. */
byte rampact; /* PC 276.  Ramp down alarm action. */
int ramptime; /* PC 277.  Ramp down time. */
int add1000errlim; /* PC 278.  Additive per 1000 error limit. */
int endbatim; /* PC 279.  End batch time. */

/* I/O Parameters */
unsigned char islot[9]; /* Slots J1 thru J7, DUART and ARCNET. */

struct Ioctl mtrin[MAXMTRIN];
struct Ioctl rtdin[MAXRTDIN];
struct Ioctl anain[MAXANAIN];
struct Ioctl digin[MAXDIGIN];
struct Ioctl digout[MAXDIGOUT];

struct Almout {
    int out; /* PC 287, etc.  Alarm output X. */
    unsigned long mask; /* PC 269, etc.  Alarm output X mask. */
} almout[MAXALMOUT];

int pmpctlout[MAXCOMPS]; /* PC 288, etc.  Component X pump output. */
struct Vlvio {
    int sol1; /* PC 292, etc.  Valve X solenoid 1. */
    int sol2; /* PC 293, etc.  Valve X solenoid 2. */
    int stemsw1; /* PC 294, etc.  Valve X stem switch 1. */
    int stemsw2; /* PC 295, etc.  Valve X stem switch 2. */
    int clsinp; /* PC 296, etc.  Valve X close input. */
} vlvio[MAXVLV];
byte sidemthd; /* PC 312.  Side detect method. */
int dauout[MAXDAU]; /* PC 313, etc.  PPU output X. */
int trip1out; /* PC 315.  Trip 1 output. */
int trip2out; /* PC 316.  Trip 2 output. */
int trip3out; /* PC 317.  Trip 3 output. */
struct Mtrtmp {
    int mtempinp; /* PC 318, etc.  Meter X temperature input. */
    long moffset; /* PC 319, etc.  Meter X offset (Ohms). */
    int mpresinp; /* PC 320, etc.  Meter X pressure input. */
    long pres4; /* PC 464, etc.  Meter X pressure at 4 mA. */
    long pres20; /* PC 465, etc.  Meter X pressure at 20 mA. */
    int mdensinp; /* PC 321, etc.  Meter X density input. */
    long dens4; /* PC 466, etc.  Meter X density at 4 mA. */
    long dens20; /* PC 467, etc.  Meter X density at 20 mA. */
    int flowinp; /* PC 322, etc.  Meter X flow input. */
} mtrtmp[MAXMTR];

int startend; /* PC 342.  Off-rack start/end input. */
int autochng; /* PC 343.  Auto/manual change-over input. */
int primalrst; /* PC 344.  Primary alarm reset input. */

int safetyinp[MAXSAFETY]; /* PC 345, etc.  Safety circuit X input. */
byte temprange; /* PC 353.  RTD inputs type. */
int opdly; /* PC 354.  Delay after outputs, e.g. for DART 2. */

byte pridsptyp; /* PC 355.  Primary display type. */
int priminlt; /* PC 356.  Primary display min. light. */
int primaxlt; /* PC 357.  Primary display max. light. */
byte secdsptyp; /* PC 358.  Secondary display type. */
```

```
int secminlt; /* PC 359.  Secondary display min. light. */
int secmaxlt; /* PC 360.  Secondary display max. light. */

int recipeinp[MAXRECIN]; /* PC 361, etc.  Recipe X selection input. */
int recipeout[MAXRECOUT]; /* PC 367, etc.  Recipe X selection output. */
int swing2; /* PC 376.  Swing-arm switch 2. */
int swing1; /* PC 377.  Swing-arm switch 1. */
unsigned char psmode; /* PC 378.  Pulse security mode. */

unsigned char fpobasis; /* PC 752.  Factored pulse outputs basis. */
int calmonerr; /* PC 753.  Calibration monitoring error limit. */
int calmonrcnt; /* PC 754.  Calibration monitoring reset count. */
byte calibfailact; /* PC 755.  Calibration failure alarm action for CALMON. */

struct Addinp {
       int ratio; /* PC 380, etc.  Additive X ratio output. */
       int feedback; /* PC 381, etc.  Additive X feedback input. */
       int select; /* PC 382, etc.  Additive X selection input. */
    int pmpctlout; /* PC 800, etc.  Additive X pump control output. */
    int bvout; /* PC 801, etc.  Additive X block valve output. */
} addinp[MAXADDS];

int addclnlinout; /* PC 398.  Additive clean line output. */
int addmtrflushout; /* PC 399.  Additive meter flush output. */

struct Cmpio {
       int bvout; /* PC 400, etc.  Component X block valve output. */
       int bvin; /* PC 401, etc.  Component X block valve input. */
} cmpio[MAXCOMPS];
int combio[15]; /* PC 408, etc.  Component combination X. */
int endbaout; /* PC 423.  End batch output. */
int endbain; /* PC 424.  End batch input. */
unsigned long addmtrflushpulses; /* PC 425.  Additive meter flush pulses. */

long buoyancy; /* PC 426.  Density adjustment for buoyancy in air. */
byte tempunits; /* PC 427.  Temperature units. */
byte densunits; /* PC 428.  Density units. */
byte presunits; /* PC 429.  Pressure units. */
int reftemp; /* PC 430.  Reference temperature. */
long tempsmpl; /* PC 431.  Sample volume. */

struct Cmptemp {
    byte option; /* PC 432, etc.  Component X temperature correction option. */
    long alpha; /* PC 433, etc.  Component X alpha. */
    int backup; /* PC 434, etc.  Component X backup temperature. */
} cmptemp[MAXCOMPS];
struct Cmppres {
    byte option; /* PC 444, etc.  Component X pressure correction option. */
    long pffac; /* PC 445, etc.  Component X F-factor. */
    long backup; /* PC 446, etc.  Component X backup pressure. */
} cmppres[MAXCOMPS];
struct Cmpdens {
    byte option; /* PC 456, etc.  Component X density option. */
    long backup; /* PC 457, etc.  Component X backup density/gravity. */
} cmpdens[MAXCOMPS];

int numrecipes; /* PC 480.  Number of recipes. */
struct Recipe {
    char name[17]; /* PC 481, etc.  Recipe X NUL terminated name. */
    int cmppct[MAXCOMPS]; /* PC 482, etc.  Recipe X component percentages. */
    struct Cmpseq {
        int cmpno;
```

```
    } cmpseq[MAXCOMPS]; /* PC 486, etc.  Recipe X sequence. */
} recipe[MAXRECIPES];

struct Commun {
    byte mode; /* PC 663, etc.  Channel X mode. */
    byte baud; /* PC 664, etc.  Channel X data rate. */
    byte dbits; /* PC 665, etc.  Channel X word size. */
    byte sbits; /* PC 666, etc.  Channel X stop bits. */
    byte parity; /* PC 667, etc.  Channel X parity. */
} commun[2];

unsigned char ddenabled; /* PC 679.  Dynamic data display enabled. */
int ddpc[16]; /* PC 680, etc.  Dynamic data display element X data code. */

byte batch; /* PC 696.  Batch summary. */
byte trans; /* PC 697.  Transaction summary. */
byte almlog; /* PC 698.  Alarm log. */
byte pflog; /* PC 699.  Power fail log. */
byte progmode; /* PC 700.  Program mode entry/exit. */
byte wmsw; /* PC 701.  W&M switch opened/closed. */
byte pcchnglog; /* PC 702.  Program value change. */
byte cfglog; /* PC 703.  Configuration summary data log. */
byte crash; /* PC 704.  Crash memory summary. */
byte totlog; /* PC 705.  Totalizers. */
byte trtkauto; /* PC 706.  Transaction ticket. */
byte trtk; /* PC 708.  Transaction ticket reprint. */
byte thrulog; /* PC 708.  Thruput. */
byte dlogseq; /* PC 709.  Sequence numbers. */
int cuthour; /* PC 710.  Cutoff hour. */

/* Blending. */
int rampdownpct; /* PC 711.  The percentage of the blend's current flow rate, etc. */
int deltarate; /* PC 712.  Change in flow rate for batch fall back. */
long loflwstart; /* PC 713.  Low flow start volume for in-line blend. */
int clnlinvol; /* PC 714.  Clean line volume. */
int lopropfac; /* PC 715.  The meter factor flow rate for low proportion
                ** components of an in-line blend.
                */
int blchkvol; /* PC 716.  Correct after volume.  Volume after which blend
               ** ratio adjustment begins.
               */
int blchkvol2; /* PC 717.  Alarm after volume.  Volume after which blend
                ** ratio alarm is checked.
                */
int vadjust; /* PC 718.  Volume over which blend adjustment is made. */
unsigned char comppctmthd; /* PC 719.  Component % display. */
unsigned char blenderrmthd; /* PC 722.  Blend error method. */
int blnddev; /* PC 723.  Max. dev. % */
int blndtol; /* PC 724.  Blend tol. % */
int blnddevvol; /* PC 725.  Max. dev. volume. */
int blndtolvol; /* PC 726.  Blend tol. volume. */
int blendsmpl; /* PC 727.  Blend sample volume. */
long stoprate[MAXCOMPS]; /* PC 173.  Component X stop rate for in-line blend. */
long rflwrate[MAXRFLWRATE][2]; /* PC 776, 777, etc.  Recipe "group" low and
                               ** high flow rates.
                               */

unsigned char pttype; /* PC 818.  Pressure transmitter types. */
unsigned char pemthd; /* PC 819.  Pe method. */
unsigned char pinchmthd; /* PC 820.  Pinch back method. */
long pinchpres; /* PC 821.  Pinch back pressure. */
long atmospres; /* PC 822.  Atmospheric pressure. */
```

```
struct Pe_lin {
    int temp; /* PC 824, etc.  Tx. */
    long pres; /* PC 825, etc.  Pe at Tx. */
} pe_lin[MAXPELIN]; /* T/Pe pairs for Pe linearization. */

/* Transaction storage. */

unsigned char trrecl; /* PC 834.  Transaction record length. */
int trnumpgs; /* PC 835.  Transaction file start page (/ 100) and # pages (% 100). */
unsigned char barecl; /* PC 836.  Batch record length. */
int banumpgs; /* PC 837.  Batch file start page (/ 100) and # pages (% 100). */
unsigned char trtkfmt; /* PC 838.  Transaction ticket predefined format. */

/* Transaction storage code configuration. */
struct Sccfg tsccfg[MAXTSC]; /* PC 840, etc.  TSC X offset and row/column. */

/* Batch storage code configuration. */
struct Sccfg bsccfg[MAXBSC];/* PC 880, etc.  TSC X offset and row/column. */

/*
**
** Data Base
****************
*/

////
```

6.5 DanLoad 6000 v5.30 Database

The "database" is the name given to the stored program code values. The database for DanLoad 6000 v5.30 is shown below. (A "byte" is an unsigned char.)

```
////

/*
**
** Data Base
*/

struct {
    long passcode; /* PC 1, etc.  User X 9-digit passcode. */
    char userid[17]; /* PC 2, etc.  User X NUL terminated ID. */
    byte pwacess; /* PC 3, etc.  User X supervisor privilege. */
} pw[MAXUSER];

int nummtrs; /* PC 50.  Number of meters. */
struct {
    char mtrid[6]; /* PC 51, etc.  Meter X ID. */
    int mtrvlv; /* PC 52, etc.  Meter X flow control valve. */
    int radjfac; /* PC 729, etc.  Meter X ratio adjustment factor. */
    int fadjfac; /* PC 728, etc.  Meter X flow adjustment factor. */
    int ramppct; /* PC 730, etc.  Meter X ramp percentage. */
    int fpo; /* PC 756, etc.  Meter X MPMC factored pulse output. */
    int numblades; /* PC 757, etc.  Meter X CALMON number of blades. */
    long maxmaxchardev; /* PC 758, etc.  Meter X CALMON max. max. characteristic deviation. */
    long maxtotchardevs; /* PC 759, etc.  Meter X CALMON max. total characteristic deviations.
*/
    unsigned char calibsmplstat; /* PC 760, etc.  Meter X calibration sample status. */
    unsigned char calmonanalmthd; /* PC 731, etc.  Meter X CALMON analysis method. */
} mtrparms[MAXMTR];

byte opmode; /* PC 25.  Operating mode. */
byte language; /* PC 28.  Language. */
byte dtefmt; /* PC 38.  Date format. */
byte dtesep; /* PC 39.  Date separator. */
byte decsep; /* PC 40.  Decimal separator. */
byte units; /* PC 29.  Product units. */
byte numdataitems; /* PC 30.  Number of data items. */
char dataprompt[5][33]; /* PC 31, etc.  Data item X NUL terminated prompt. */
int pmt_timout; /* PC 36.  Prompt time-out. */
byte chk_disp; /* PC 37.  Check display. */
byte unit_type; /* PC 26.  Unit type. */
byte fcv_type; /* PC 27.  Valve type. */
unsigned char dspdr; /* PC 41.  Display data rate. */
int transeqnum; /* PC 42.  Transaction #. */
int batchseqnum; /* PC 43.  Batch #.  */
unsigned char roundmthd; /* PC 45.  Rounding method. */
int densscale; /* PC 46.  Density/gravity scale. */
int maxbatch; /* PC 47.  Maximum batches/transaction. */

int numvalves; /* PC 48.  Number of (flow control) valves. */
int initflwtime; /* PC 49.  Initial flow time. */

/* Component parameters */
int numcomps; /* PC 65.  Number of components. */
struct {
```

```
    char compid[17]; /* PC 66, etc.  Component X NUL terminated ID. */
    int compmtr; /* PC 67, etc.  Component X meter number. */
    long massadj; /* PC 68, etc.  Component X mass adjustment. */
} comp[MAXCOMPS];

/* Delivery parameters */
long maxpreset; /* PC 78.  Maximum preset volume. */
long minpreset; /* PC 79.  Minimum preset volume. */
byte predeltype; /* PC 80.  Preset/delivery type. */
byte deldisptype; /* PC 81.  Delivery display type. */
byte stopkey; /* PC 82.  Stop key action. */
long fallbackvol; /* PC 83.  Fall back volume. */

struct Compdel {
    long loflwstart; /* PC 84, etc.  Component X low flow start volume. */
    long loflwrestart; /* PC 85, etc.  Component X low flow restart volume. */
    long loflwstop; /* PC 86, etc.  Component X low flow stop volume. */
    int linepackdly; /* PC 87, etc.  Component X line pack delay. */
    int pumpstopdly; /* PC 88, etc.  Component X pump stop delay. */
    int blkvalvedly; /* PC 89, etc.  Component X block valve delay. */
} compdel[MAXCOMPS];
int maxrampclicks; /* PC 108.  Ramp clicks. */
int maxmaintclicks; /* PC 109.  Maintenance clicks. */
int addpmpdly; /* PC 110.  Additive pump stop delay. */
int primcmp; /* PC 111.  Primary component. */

/* Digital Valve Parameters */
struct {
    int dvlferrpct; /* PC 112, etc.  Valve X low flow % error. */
    int dvhferrpct; /* PC 113, etc.  Valve X high flow % error. */
    int dvdly; /* PC 114, etc.  Valve X maint click adjustment. */
    byte openmthd; /* PC 115, etc.  Valve X open method. */
} dvparms[MAXVLV];

/* Pulse per unit (DAU) parameters */
int numdaus; /* PC 128.  Number of PPU's. */
struct Dau {
    struct Dauctl {
        byte gross;
        byte net;
    } dauctl[MAXMTR]; /* PC 129, etc.  PPU X control meters. */
    int daufactor; /* PC 130, etc.  PPU X factor. */
    int daupulwidth; /* PC 131, etc.  PPU X pulse width. */
} dau[MAXDAU];

/* Additive Parameters */
int numadds; /* PC 135.  Number of additives. */
byte addselmthd; /* PC 136.  Additive selection method. */
int injpct; /* PC 137.  Additive inject % */
int addclnlinvol; /* PC 138.  Additive clean line volume. */
byte addunits; /* PC 44. Additive units. */
struct Add {
    struct Addctl {
        byte gross;
        byte net;
    } addctl[MAXMTR]; /* PC 139, etc.  Additive X control meters. */
    int ratiovol; /* PC 140, etc.  Additive X ratio volume in whole units. */
    unsigned char injmthd; /* PC 141, etc.  Additive X injection method. */
    long kfac; /* PC 142, etc.  Additive X volume per pulse or K-factor in
               ** hundredths.
               */
    long reqdvol; /* PC 143, etc.  Additive X volume (in external product
```

```
                 ** units x 10,000) per 1000 units of product.
                 */
} add[MAXADDS];

/* Factors  */
int numfacs; /* PC 169.  Number of factors/component. */
byte mfacmthd; /* PC 170.  Meter factor method. */
struct Fac {
    long nomkfac; /* PC 171, etc.  Component X nominal K-factor. */
    long mstrmfac; /* PC 172, etc.  Component X master meter factor. */
    struct Mtrfac {
        long flwrate; /* PC 174, etc.  Component X flow rate Y. */
        long mfac; /* PC 175.  Component X meter factor Y. */
    } mtrfac[MAXFACS];
} fac[MAXCOMPS];
int mstrmfacpct; /* PC 215.  Master meter factor percentage. */
int adjmfacpct; /* PC 216.  Adjacent meter factor percentage. */
unsigned char pwpertr; /* PC 217.  Password (passcode) per transaction. */
char produnitsmnemo[7]; /* PC 218.  Product units mnemonic. */
unsigned char userestart; /* PC 219.  Use restart quantity. */

/* Alarm Parameters */
int secalrst; /* PC 220.  Secondary alarm reset (secs). */
byte lowflwact; /* PC 221.  Low flow alarm action. */
long lowflwlim; /* PC 222.  Minimum flow rate. */
int lowflwtime; /* PC 223.  Low flow time. */
byte hiflwact; /* PC 224.  High flow alarm action. */
long hiflwlim; /* PC 225.  Maximum flow rate. */
int hiflwtime; /* PC 226.  High flow time. */
int ovrunlim; /* PC 227.  Overrun limit volume. */
byte undrflwact; /* PC 228.  Underflow alarm action. */
int undrflwlim; /* PC 229.  Underflow limit volume. */
byte noflwact; /* PC 230.  No flow alarm action. */
int noflwtime; /* PC 231.  No flow time. */
int unauthflw; /* PC 232.  Unauth. flow limit volume. */
int pserr; /* PC 233.  Pulse security error limit. */
long psresetcnt; /* PC 234.  Pulse security reset count. */
byte dlfailact; /* PC 235.  Data logging alarm action. */
int commtimout[2]; /* PC 236's and 237.  Comms fail time-out per channel. */
byte tempfailact; /* PC 238.  Temperature fail alarm action. */
int tempmin; /* PC 239.  Min. temperature. */
int tempmax; /* PC 240.  Max. temperature. */
byte densfailact; /* PC 241.  Density fail alarm action. */
long densmin; /* PC 242.  Minimum density/gravity. */
long densmax; /* PC 243.  Maximum density/gravity. */
long presmin; /* PC 244.  Minimum pressure. */
long presmax; /* PC 245.  Maximum pressure. */
int adderrlim; /* PC 246.  Additive error limit. */
int addfbckcnt; /* PC 247.  # pulses per additive cycle or # seconds. */
int bvclsdly; /* PC 248.  Block valve time. */

struct Safety {
    byte action; /* PC 249, etc.  Safety circuit X alarm action. */
    char msg[33]; /* PC 250, etc.  Safety ciruit X alarm message. */
} safety[MAXSAFETY];

byte sctype[MAXSAFETY]; /* PC 265, etc.  Safety circuit X type. */

long unauthaddflw; /* PC 271.  Unauthorized additive flow volume. */
int praddminpct; /* PC 272.  Additive volume verification minimum error
                 ** percentage.
                 */
```

```
int praddmaxpct; /* PC 273.  Additive volume verification maximum error
                 ** percentage.
                 */

byte storeact; /* PC 274.  Storage alarm action. */
byte poweract; /* PC 275.  Power fail alarm action. */
byte rampact; /* PC 276.  Ramp down alarm action. */
int ramptime; /* PC 277.  Ramp down time. */
int add1000errlim; /* PC 278.  Additive per 1000 error limit. */
int endbatim; /* PC 279.  End batch time. */

/* I/O Parameters */
unsigned char islot[9]; /* Slots J1 thru J7, DUART and ARCNET. */

struct Ioctl mtrin[MAXMTRIN];
struct Ioctl rtdin[MAXRTDIN];
struct Ioctl anain[MAXANAIN];
struct Ioctl digin[MAXDIGIN];
struct Ioctl digout[MAXDIGOUT];

struct Almout {
    int out; /* PC 287, etc.  Alarm output X. */
    unsigned long mask; /* PC 269, etc.  Alarm output X mask. */
} almout[MAXALMOUT];

int pmpctlout[MAXCOMPS]; /* PC 288, etc.  Component X pump output. */
struct Vlvio {
    int sol1; /* PC 292, etc.  Valve X solenoid 1. */
    int sol2; /* PC 293, etc.  Valve X solenoid 2. */
    int stemsw1; /* PC 294, etc.  Valve X stem switch 1. */
    int stemsw2; /* PC 295, etc.  Valve X stem switch 2. */
    int clsinp; /* PC 296, etc.  Valve X close input. */
} vlvio[MAXVLV];
byte sidemthd; /* PC 312.  Side detect method. */
int dauout[MAXDAU]; /* PC 313, etc.  PPU output X. */
int trip1out; /* PC 315.  Trip 1 output. */
int trip2out; /* PC 316.  Trip 2 output. */
int trip3out; /* PC 317.  Trip 3 output. */
struct Mtrtmp {
    int mtempinp; /* PC 318, etc.  Meter X temperature input. */
    long moffset; /* PC 319, etc.  Meter X offset (Ohms). */
    int mpresinp; /* PC 320, etc.  Meter X pressure input. */
    long pres4; /* PC 464, etc.  Meter X pressure at 4 mA. */
    long pres20; /* PC 465, etc.  Meter X pressure at 20 mA. */
    int mdensinp; /* PC 321, etc.  Meter X density input. */
    long dens4; /* PC 466, etc.  Meter X density at 4 mA. */
    long dens20; /* PC 467, etc.  Meter X density at 20 mA. */
    int flowinp; /* PC 322, etc.  Meter X flow input. */
} mtrtmp[MAXMTR];

int startend; /* PC 342.  Off-rack start/end input. */
int autochng; /* PC 343.  Auto/manual change-over input. */
int primalrst; /* PC 344.  Primary alarm reset input. */

int safetyinp[MAXSAFETY]; /* PC 345, etc.  Safety circuit X input. */
byte temprange; /* PC 353.  Range for RTD inputs on analog inputs boards. */
int opdly; /* PC 354.  Delay after outputs, e.g. for DART 2. */

byte pridsptyp; /* PC 355.  Primary display type. */
int priminlt; /* PC 356.  Primary display min. light. */
int primaxlt; /* PC 357.  Primary display max. light. */
byte secdsptyp; /* PC 358.  Secondary display type. */
```

```
int secminlt; /* PC 359.  Secondary display min. light. */
int secmaxlt; /* PC 360.  Secondary display max. light. */

int recipeinp[MAXRECIN]; /* PC 361, etc.  Recipe X selection input. */
int recipeout[MAXRECOUT]; /* PC 367, etc.  Recipe X selection output. */
int swing2; /* PC 376.  Swing-arm switch 2. */
int swing1; /* PC 377.  Swing-arm switch 1. */
unsigned char psmode; /* PC 378.  Pulse security mode. */

unsigned char fpobasis; /* PC 752.  Factored pulse outputs basis. */
int calmonerr; /* PC 753.  Calibration monitoring error limit. */
int calmonrcnt; /* PC 754.  Calibration monitoring reset count. */
byte calibfailact; /* PC 755.  Calibration failure alarm action for CALMON. */

struct Addinp {
       int ratio; /* PC 380, etc.  Additive X ratio output. */
       int feedback; /* PC 381, etc.  Additive X feedback input. */
       int select; /* PC 382, etc.  Additive X selection input. */
    int pmpctlout; /* PC 800, etc.  Additive X pump control output. */
    int bvout; /* PC 801, etc.  Additive X block valve output. */
} addinp[MAXADDS];

int addclnlinout; /* PC 398.  Additive clean line output. */
int addmtrflushout; /* PC 399.  Additive meter flush output. */

struct Cmpio {
       int bvout; /* PC 400, etc.  Component X block valve output. */
       int bvin; /* PC 401, etc.  Component X block valve input. */
} cmpio[MAXCOMPS];
int combio[15]; /* PC 408, etc.  Component combination X. */
int endbaout; /* PC 423.  End batch output. */
int endbain; /* PC 424.  End batch input. */
unsigned long addmtrflushpulses; /* PC 425.  Additive meter flush pulses. */

long buoyancy; /* PC 426.  Density adjustment for buoyancy in air. */
byte tempunits; /* PC 427.  Temperature units. */
byte densunits; /* PC 428.  Density units. */
byte presunits; /* PC 429.  Pressure units. */
int reftemp; /* PC 430.  Reference temperature. */
long tempsmpl; /* PC 431.  Sample volume. */

struct Cmptemp {
    byte option; /* PC 432, etc.  Component X temperature correction option. */
    long alpha; /* PC 433, etc.  Component X alpha. */
    int backup; /* PC 434, etc.  Component X backup temperature. */
} cmptemp[MAXCOMPS];
struct Cmppres {
    byte option; /* PC 444, etc.  Component X pressure correction option. */
    long pffac; /* PC 445, etc.  Component X F-factor. */
    long backup; /* PC 446, etc.  Component X backup pressure. */
} cmppres[MAXCOMPS];
struct Cmpdens {
    byte option; /* PC 456, etc.  Component X density option. */
    long backup; /* PC 457, etc.  Component X backup density/gravity. */
} cmpdens[MAXCOMPS];

int numrecipes; /* PC 480.  Number of recipes. */
struct Recipe {
    char name[17]; /* PC 481, etc.  Recipe X NUL terminated name. */
    int cmppct[MAXCOMPS]; /* PC 482, etc.  Recipe X component percentages. */
    struct Cmpseq {
        int cmpno;
```

```
    } cmpseq[MAXCOMPS]; /* PC 486, etc.  Recipe X sequence. */
} recipe[MAXRECIPES];

/* v5.30 */
int commsaddr; /* PC 662.  Communications address. */

struct Commun {
    byte mode; /* PC 663, etc.  Channel X mode. */
    byte baud; /* PC 664, etc.  Channel X data rate. */
    byte dbits; /* PC 665, etc.  Channel X word size. */
    byte sbits; /* PC 666, etc.  Channel X stop bits. */
    byte parity; /* PC 667, etc.  Channel X parity. */
} commun[2];

unsigned char ddenabled; /* PC 679.  Dynamic data display enabled. */
int ddpc[16]; /* PC 680, etc.  Dynamic data display element X data code. */

byte batch; /* PC 696.  Batch summary. */
byte trans; /* PC 697.  Transaction summary. */
byte almlog; /* PC 698.  Alarm log. */
byte pflog; /* PC 699.  Power fail log. */
byte progmode; /* PC 700.  Program mode entry/exit. */
byte wmsw; /* PC 701.  W&M switch opened/closed. */
byte pcchnglog; /* PC 702.  Program value change. */
byte cfglog; /* PC 703.  Configuration summary data log. */
byte crash; /* PC 704.  Crash memory summary. */
byte totlog; /* PC 705.  Totalizers. */
byte trtkauto; /* PC 706.  Transaction ticket. */
byte trtk; /* PC 708.  Transaction ticket reprint. */
byte thrulog; /* PC 708.  Thruput. */
byte dlogseq; /* PC 709.  Sequence numbers. */
int cuthour; /* PC 710.  Cutoff hour. */

/* Blending. */
int rampdownpct; /* PC 711.  The percentage of the blend's current flow rate, etc. */
int deltarate; /* PC 712.  Change in flow rate for batch fall back. */
long loflwstart; /* PC 713.  Low flow start volume for in-line blend. */
int clnlinvol; /* PC 714.  Clean line volume. */
int lopropfac; /* PC 715.  The meter factor flow rate for low proportion
                ** components of an in-line blend.
                */
int blchkvol; /* PC 716.  Correct after volume.  Volume after which blend
               ** ratio adjustment begins.
               */
int blchkvol2; /* PC 717.  Alarm after volume.  Volume after which blend
                ** ratio alarm is checked.
                */
int vadjust; /* PC 718.  Volume over which blend adjustment is made. */
unsigned char comppctmthd; /* PC 719.  Component % display. */
unsigned char blenderrmthd; /* PC 722.  Blend error method. */
int blnddev; /* PC 723.  Max. dev. % */
int blndtol; /* PC 724.  Blend tol. % */
int blnddevvol; /* PC 725.  Max. dev. volume. */
int blndtolvol; /* PC 726.  Blend tol. volume. */
int blendsmpl; /* PC 727.  Blend sample volume. */
long stoprate[MAXCOMPS]; /* PC 173.  Component X stop rate for in-line blend. */
long rflwrate[MAXRFLWRATE][2]; /* PC 776, 777, etc.  Recipe "group" low and
                               ** high flow rates.
                               */

unsigned char pttype; /* PC 818.  Pressure transmitter types. */
unsigned char pemthd; /* PC 819.  Pe method. */
```

```
unsigned char pinchmthd; /* PC 820.  Pinch back method. */
long pinchpres; /* PC 821.  Pinch back pressure. */
long atmospres; /* PC 822.  Atmospheric pressure. */
struct Pe_lin {
    int temp; /* PC 824, etc.  Tx. */
    long pres; /* PC 825, etc.  Pe at Tx. */
} pe_lin[MAXPELIN]; /* T/Pe pairs for Pe linearization. */

/* Transaction storage. */

unsigned char trrecl; /* PC 834.  Transaction record length. */
int trnumpgs; /* PC 835.  Transaction file start page (/ 100) and # pages (% 100). */
unsigned char barecl; /* PC 836.  Batch record length. */
int banumpgs; /* PC 837.  Batch file start page (/ 100) and # pages (% 100). */
unsigned char trtkfmt; /* PC 838.  Transaction ticket predefined format. */

/* Transaction storage code configuration. */
struct Sccfg tsccfg[MAXTSC]; /* PC 840, etc.  TSC X offset and row/column. */

/* Batch storage code configuration. */
struct Sccfg bsccfg[MAXBSC];/* PC 880, etc.  TSC X offset and row/column. */

/* v5.30 */
/* Calibration for 4-20 mA and RTD inputs on CPU board. */
unsigned char rtdalpha; /* PC 983.  Alpha for all RTD's. */
int clcalib[4]; /* PC 984, etc.  CPU board 4-20 ma input LOLO, LOHI, HILO and
                ** HIHI counts.
                */
int rtdcalib[4]; /* PC 988, etc. CPU board RTD input LOLO, LOHI, HILO and
                 ** HIHI counts.
                 */

/*
**
** Data Base
***************
*/

////
```

6.6 DanLoad 6000 v5.40 Database

The "database" is the name given to the stored program code values. The database for DanLoad 6000 v5.40 is shown below. (A "byte" is an unsigned char.)

```
////

/*
**
** Data Base
*/

struct {
    long passcode; /* PC 1, etc.  User X 9-digit passcode. */
    char userid[17]; /* PC 2, etc.  User X NUL terminated ID. */
    byte pwacess; /* PC 3, etc.  User X supervisor privilege. */
} pw[MAXUSER];

int nummtrs; /* PC 50.  Number of meters. */
struct {
    char mtrid[6]; /* PC 51, etc.  Meter X ID. */
    int mtrvlv; /* PC 52, etc.  Meter X flow control valve. */
    int radjfac; /* PC 729, etc.  Meter X ratio adjustment factor. */
    int fadjfac; /* PC 728, etc.  Meter X flow adjustment factor. */
    int ramppct; /* PC 730, etc.  Meter X ramp percentage. */
    int fpo; /* PC 756, etc.  Meter X MPMC factored pulse output. */
    int numblades; /* PC 757, etc.  Meter X CALMON number of blades. */
    long maxmaxchardev; /* PC 758, etc.  Meter X CALMON max. max. characteristic deviation. */
    long maxtotchardevs; /* PC 759, etc.  Meter X CALMON max. total characteristic deviations.
*/
    unsigned char calibsmplstat; /* PC 760, etc.  Meter X calibration sample status. */
    unsigned char calmonanalmthd; /* PC 731, etc.  Meter X CALMON analysis method. */
} mtrparms[MAXMTR];

byte opmode; /* PC 25.  Operating mode. */
byte language; /* PC 28.  Language. */
byte dtefmt; /* PC 38.  Date format. */
byte dtesep; /* PC 39.  Date separator. */
byte decsep; /* PC 40.  Decimal separator. */
byte units; /* PC 29.  Product units. */
byte numdataitems; /* PC 30.  Number of data items. */
char dataprompt[5][33]; /* PC 31, etc.  Data item X NUL terminated prompt. */
int pmt_timout; /* PC 36.  Prompt time-out. */
byte chk_disp; /* PC 37.  Check display. */
byte unit_type; /* PC 26.  Unit type. */
byte fcv_type; /* PC 27.  Valve type. */
unsigned char dspdr; /* PC 41.  Display data rate. */
int transeqnum; /* PC 42.  Transaction #. */
int batchseqnum; /* PC 43.  Batch #.  */
unsigned char roundmthd; /* PC 45.  Rounding method. */
int densscale; /* PC 46.  Density/gravity scale. */
int maxbatch; /* PC 47.  Maximum batches/transaction. */

int numvalves; /* PC 48.  Number of (flow control) valves. */
int initflwtime; /* PC 49.  Initial flow time. */

/* Component parameters */
int numcomps; /* PC 65.  Number of components. */
struct {
```

```
    char compid[17]; /* PC 66, etc.  Component X NUL terminated ID. */
    int compmtr; /* PC 67, etc.  Component X meter number. */
    long massadj; /* PC 68, etc.  Component X mass adjustment. */
} comp[MAXCOMPS];

/* Delivery parameters */
long maxpreset; /* PC 78.  Maximum preset volume. */
long minpreset; /* PC 79.  Minimum preset volume. */
byte predeltype; /* PC 80.  Preset/delivery type. */
byte deldisptype; /* PC 81.  Delivery display type. */
byte stopkey; /* PC 82.  Stop key action. */
long fallbackvol; /* PC 83.  Fall back volume. */

struct Compdel {
    long loflwstart; /* PC 84, etc.  Component X low flow start volume. */
    long loflwrestart; /* PC 85, etc.  Component X low flow restart volume. */
    long loflwstop; /* PC 86, etc.  Component X low flow stop volume. */
    int linepackdly; /* PC 87, etc.  Component X line pack delay. */
    int pumpstopdly; /* PC 88, etc.  Component X pump stop delay. */
    int blkvalvedly; /* PC 89, etc.  Component X block valve delay. */
} compdel[MAXCOMPS];
int maxrampclicks; /* PC 108.  Ramp clicks. */
int maxmaintclicks; /* PC 109.  Maintenance clicks. */
int addpmpdly; /* PC 110.  Additive pump stop delay. */
int primcmp; /* PC 111.  Primary component. */

/* Digital Valve Parameters */
struct {
    int dvlferrpct; /* PC 112, etc.  Valve X low flow % error. */
    int dvhferrpct; /* PC 113, etc.  Valve X high flow % error. */
    int dvdly; /* PC 114, etc.  Valve X maint click adjustment. */
    byte openmthd; /* PC 115, etc.  Valve X open method. */
} dvparms[MAXVLV];

/* Pulse per unit (DAU) parameters */
int numdaus; /* PC 128.  Number of PPU's. */
struct Dau {
    struct Dauctl {
        byte gross;
        byte net;
    } dauctl[MAXMTR]; /* PC 129, etc.  PPU X control meters. */
    int daufactor; /* PC 130, etc.  PPU X factor. */
    int daupulwidth; /* PC 131, etc.  PPU X pulse width. */
} dau[MAXDAU];

/* Additive Parameters */
int numadds; /* PC 135.  Number of additives. */
byte addselmthd; /* PC 136.  Additive selection method. */
int injpct; /* PC 137.  Additive inject % */
int addclnlinvol; /* PC 138.  Additive clean line volume. */
byte addunits; /* PC 44. Additive units. */
struct Add {
    struct Addctl {
        byte gross;
        byte net;
    } addctl[MAXMTR]; /* PC 139, etc.  Additive X control meters. */
    int ratiovol; /* PC 140, etc.  Additive X ratio volume in whole units. */
    unsigned char injmthd; /* PC 141, etc.  Additive X injection method. */
    long kfac; /* PC 142, etc.  Additive X volume per pulse or K-factor in
               ** hundredths.
               */
    long reqdvol; /* PC 143, etc.  Additive X volume (in external product
```

```
                 ** units x 10,000) per 1000 units of product.
                 */
} add[MAXADDS];

/* Factors  */
int numfacs; /* PC 169.  Number of factors/component. */
byte mfacmthd; /* PC 170.  Meter factor method. */
struct Fac {
    long nomkfac; /* PC 171, etc.  Component X nominal K-factor. */
    long mstrmfac; /* PC 172, etc.  Component X master meter factor. */
    struct Mtrfac {
        long flwrate; /* PC 174, etc.  Component X flow rate Y. */
        long mfac; /* PC 175.  Component X meter factor Y. */
    } mtrfac[MAXFACS];
} fac[MAXCOMPS];
int mstrmfacpct; /* PC 215.  Master meter factor percentage. */
int adjmfacpct; /* PC 216.  Adjacent meter factor percentage. */
char grsunitsmnemo[7]; /* PC 217.  Gross quantity units mnemonic. */
char stdunitsmnemo[7]; /* PC 218.  Standard quantity units mnemonic. */
unsigned char userestart; /* PC 219.  Use restart quantity. */

/* Alarm Parameters */
int secalrst; /* PC 220.  Secondary alarm reset (secs). */
byte lowflwact; /* PC 221.  Low flow alarm action. */
long lowflwlim; /* PC 222.  Minimum flow rate. */
int lowflwtime; /* PC 223.  Low flow time. */
byte hiflwact; /* PC 224.  High flow alarm action. */
long hiflwlim; /* PC 225.  Maximum flow rate. */
int hiflwtime; /* PC 226.  High flow time. */
int ovrunlim; /* PC 227.  Overrun limit volume. */
byte undrflwact; /* PC 228.  Underflow alarm action. */
int undrflwlim; /* PC 229.  Underflow limit volume. */
byte noflwact; /* PC 230.  No flow alarm action. */
int noflwtime; /* PC 231.  No flow time. */
int unauthflw; /* PC 232.  Unauth. flow limit volume. */
int pserr; /* PC 233.  Pulse security error limit. */
long psresetcnt; /* PC 234.  Pulse security reset count. */
byte dlfailact; /* PC 235.  Data logging alarm action. */
int commtimout[2]; /* PC 236's and 237.  Comms fail time-out per channel. */
byte tempfailact; /* PC 238.  Temperature fail alarm action. */
int tempmin; /* PC 239.  Min. temperature. */
int tempmax; /* PC 240.  Max. temperature. */
byte densfailact; /* PC 241.  Density fail alarm action. */
long densmin; /* PC 242.  Minimum density/gravity. */
long densmax; /* PC 243.  Maximum density/gravity. */
long presmin; /* PC 244.  Minimum pressure. */
long presmax; /* PC 245.  Maximum pressure. */
int adderrlim; /* PC 246.  Additive error limit. */
int addfbckcnt; /* PC 247.  # pulses per additive cycle or # seconds. */
int bvclsdly; /* PC 248.  Block valve time. */

struct Safety {
    byte action; /* PC 249, etc.  Safety circuit X alarm action. */
    char msg[33]; /* PC 250, etc.  Safety ciruit X alarm message. */
} safety[MAXSAFETY];

byte sctype[MAXSAFETY]; /* PC 265, etc.  Safety circuit X type. */

long unauthaddflw; /* PC 271.  Unauthorized additive flow volume. */
int praddminpct; /* PC 272.  Additive volume verification minimum error
                 ** percentage.
                 */
```

```
int praddmaxpct; /* PC 273.  Additive volume verification maximum error
                 ** percentage.
                 */

byte storeact; /* PC 274.  Storage alarm action. */
byte poweract; /* PC 275.  Power fail alarm action. */
byte rampact; /* PC 276.  Ramp down alarm action. */
int ramptime; /* PC 277.  Ramp down time. */
int add1000errlim; /* PC 278.  Additive per 1000 error limit. */
int endbatim; /* PC 279.  End batch time. */

/* I/O Parameters */
unsigned char islot[9]; /* Slots J1 thru J7, DUART and ARCNET. */

struct Ioctl mtrin[MAXMTRIN];
struct Ioctl rtdin[MAXRTDIN];
struct Ioctl anain[MAXANAIN];
struct Ioctl digin[MAXDIGIN];
struct Ioctl digout[MAXDIGOUT];

struct Almout {
    int out; /* PC 287, etc.  Alarm output X. */
    unsigned long mask; /* PC 269, etc.  Alarm output X mask. */
} almout[MAXALMOUT];

int pmpctlout[MAXCOMPS]; /* PC 288, etc.  Component X pump output. */
struct Vlvio {
    int sol1; /* PC 292, etc.  Valve X solenoid 1. */
    int sol2; /* PC 293, etc.  Valve X solenoid 2. */
    int stemsw1; /* PC 294, etc.  Valve X stem switch 1. */
    int stemsw2; /* PC 295, etc.  Valve X stem switch 2. */
    int clsinp; /* PC 296, etc.  Valve X close input. */
} vlvio[MAXVLV];
byte sidemthd; /* PC 312.  Side detect method. */
int dauout[MAXDAU]; /* PC 313, etc.  PPU output X. */
int trip1out; /* PC 315.  Trip 1 output. */
int trip2out; /* PC 316.  Trip 2 output. */
int trip3out; /* PC 317.  Trip 3 output. */
struct Mtrtmp {
    int mtempinp; /* PC 318, etc.  Meter X temperature input. */
    long moffset; /* PC 319, etc.  Meter X offset (Ohms). */
    int mpresinp; /* PC 320, etc.  Meter X pressure input. */
    long pres4; /* PC 464, etc.  Meter X pressure at 4 mA. */
    long pres20; /* PC 465, etc.  Meter X pressure at 20 mA. */
    int mdensinp; /* PC 321, etc.  Meter X density input. */
    long dens4; /* PC 466, etc.  Meter X density at 4 mA. */
    long dens20; /* PC 467, etc.  Meter X density at 20 mA. */
    int flowinp; /* PC 322, etc.  Meter X flow input. */
} mtrtmp[MAXMTR];

int startend; /* PC 342.  Off-rack start/end input. */
int autochng; /* PC 343.  Auto/manual change-over input. */
int primalrst; /* PC 344.  Primary alarm reset input. */

int safetyinp[MAXSAFETY]; /* PC 345, etc.  Safety circuit X input. */
byte temprange; /* PC 353.  Range for RTD inputs on analog inputs boards. */
int opdly; /* PC 354.  Delay after outputs, e.g. for DART 2. */

byte pridsptyp; /* PC 355.  Primary display type. */
int priminlt; /* PC 356.  Primary display min. light. */
int primaxlt; /* PC 357.  Primary display max. light. */
byte secdsptyp; /* PC 358.  Secondary display type. */
```

```
int secminlt; /* PC 359.  Secondary display min. light. */
int secmaxlt; /* PC 360.  Secondary display max. light. */

int recipeinp[MAXRECIN]; /* PC 361, etc.  Recipe X selection input. */
int recipeout[MAXRECOUT]; /* PC 367, etc.  Recipe X selection output. */
int swing2; /* PC 376.  Swing-arm switch 2. */
int swing1; /* PC 377.  Swing-arm switch 1. */
unsigned char psmode; /* PC 378.  Pulse security mode. */

unsigned char fpobasis; /* PC 752.  Factored pulse outputs basis. */
int calmonerr; /* PC 753.  Calibration monitoring error limit. */
int calmonrcnt; /* PC 754.  Calibration monitoring reset count. */
byte calibfailact; /* PC 755.  Calibration failure alarm action for CALMON. */

struct Addinp {
       int ratio; /* PC 380, etc.  Additive X ratio output. */
       int feedback; /* PC 381, etc.  Additive X feedback input. */
       int select; /* PC 382, etc.  Additive X selection input. */
    int pmpctlout; /* PC 800, etc.  Additive X pump control output. */
    int bvout; /* PC 801, etc.  Additive X block valve output. */
} addinp[MAXADDS];

int addclnlinout; /* PC 398.  Additive clean line output. */
int addmtrflushout; /* PC 399.  Additive meter flush output. */

struct Cmpio {
       int bvout; /* PC 400, etc.  Component X block valve output. */
       int bvin; /* PC 401, etc.  Component X block valve input. */
} cmpio[MAXCOMPS];
int combio[15]; /* PC 408, etc.  Component combination X. */
int endbaout; /* PC 423.  End batch output. */
int endbain; /* PC 424.  End batch input. */
unsigned long addmtrflushpulses; /* PC 425.  Additive meter flush pulses. */

long buoyancy; /* PC 426.  Density adjustment for buoyancy in air. */
byte tempunits; /* PC 427.  Temperature units. */
byte densunits; /* PC 428.  Density units. */
byte presunits; /* PC 429.  Pressure units. */
int reftemp; /* PC 430.  Reference temperature. */
long tempsmpl; /* PC 431.  Sample volume. */

struct Cmptemp {
    byte option; /* PC 432, etc.  Component X temperature correction option. */
    long alpha; /* PC 433, etc.  Component X alpha. */
    int backup; /* PC 434, etc.  Component X backup temperature. */
} cmptemp[MAXCOMPS];
struct Cmppres {
    byte option; /* PC 444, etc.  Component X pressure correction option. */
    long pffac; /* PC 445, etc.  Component X F-factor. */
    long backup; /* PC 446, etc.  Component X backup pressure. */
} cmppres[MAXCOMPS];
struct Cmpdens {
    byte option; /* PC 456, etc.  Component X density option. */
    long backup; /* PC 457, etc.  Component X backup density/gravity. */
} cmpdens[MAXCOMPS];

int numrecipes; /* PC 480.  Number of recipes. */
struct Recipe {
    char name[17]; /* PC 481, etc.  Recipe X NUL terminated name. */
    int cmppct[MAXCOMPS]; /* PC 482, etc.  Recipe X component percentages. */
    struct Cmpseq {
        int cmpno;
```

```
    } cmpseq[MAXCOMPS]; /* PC 486, etc.  Recipe X sequence. */
} recipe[MAXRECIPES];

/* v5.30 */
int commsaddr; /* PC 662.  Communications address. */

struct Commun {
    byte mode; /* PC 663, etc.  Channel X mode. */
    byte baud; /* PC 664, etc.  Channel X data rate. */
    byte dbits; /* PC 665, etc.  Channel X word size. */
    byte sbits; /* PC 666, etc.  Channel X stop bits. */
    byte parity; /* PC 667, etc.  Channel X parity. */
} commun[2];

unsigned char commalarmmanb; /* PC 673.  Channel B alarm in manual. */

unsigned char ddenabled; /* PC 679.  Dynamic data display enabled. */
int ddpc[16]; /* PC 680, etc.  Dynamic data display element X data code. */

byte batch; /* PC 696.  Batch summary. */
byte trans; /* PC 697.  Transaction summary. */
byte almlog; /* PC 698.  Alarm log. */
byte pflog; /* PC 699.  Power fail log. */
byte progmode; /* PC 700.  Program mode entry/exit. */
byte wmsw; /* PC 701.  W&M switch opened/closed. */
byte pcchnglog; /* PC 702.  Program value change. */
byte cfglog; /* PC 703.  Configuration summary data log. */
byte crash; /* PC 704.  Crash memory summary. */
byte totlog; /* PC 705.  Totalizers. */
byte trtkauto; /* PC 706.  Transaction ticket. */
byte trtk; /* PC 708.  Transaction ticket reprint. */
byte thrulog; /* PC 708.  Thruput. */
byte dlogseq; /* PC 709.  Sequence numbers. */
int cuthour; /* PC 710.  Cutoff hour. */

/* Blending. */
int rampdownpct; /* PC 711.  The percentage of the blend's current flow rate, etc. */
int deltarate; /* PC 712.  Change in flow rate for batch fall back. */
long loflwstart; /* PC 713.  Low flow start volume for in-line blend. */
int clnlinvol; /* PC 714.  Clean line volume. */
int lopropfac; /* PC 715.  The meter factor flow rate for low proportion
                ** components of an in-line blend.
                */
int blchkvol; /* PC 716.  Correct after volume.  Volume after which blend
               ** ratio adjustment begins.
               */
int blchkvol2; /* PC 717.  Alarm after volume.  Volume after which blend
                ** ratio alarm is checked.
                */
int vadjust; /* PC 718.  Volume over which blend adjustment is made. */
unsigned char comppctmthd; /* PC 719.  Component % display. */
unsigned char blenderrmthd; /* PC 722.  Blend error method. */
int blnddev; /* PC 723.  Max. dev. % */
int blndtol; /* PC 724.  Blend tol. % */
int blnddevvol; /* PC 725.  Max. dev. volume. */
int blndtolvol; /* PC 726.  Blend tol. volume. */
int blendsmpl; /* PC 727.  Blend sample volume. */
long stoprate[MAXCOMPS]; /* PC 173.  Component X stop rate for in-line blend. */
long rflwrate[MAXRFLWRATE][2]; /* PC 776, 777, etc.  Recipe "group" low and
                               ** high flow rates.
                               */
```

```
unsigned char pttype; /* PC 818.  Pressure transmitter types. */
unsigned char pemthd; /* PC 819.  Pe method. */
unsigned char pinchmthd; /* PC 820.  Pinch back method. */
long pinchpres; /* PC 821.  Pinch back pressure. */
long atmospres; /* PC 822.  Atmospheric pressure. */
struct Pe_lin {
    int temp; /* PC 824, etc.  Tx. */
    long pres; /* PC 825, etc.  Pe at Tx. */
} pe_lin[MAXPELIN]; /* T/Pe pairs for Pe linearization. */

/* Transaction storage. */

unsigned char trrecl; /* PC 834.  Transaction record length. */
int trnumpgs; /* PC 835.  Transaction file start page (/ 100) and # pages (% 100). */
unsigned char barecl; /* PC 836.  Batch record length. */
int banumpgs; /* PC 837.  Batch file start page (/ 100) and # pages (% 100). */
unsigned char trtkfmt; /* PC 838.  Transaction ticket predefined format. */

/* Transaction storage code configuration. */
struct Sccfg tsccfg[MAXTSC]; /* PC 840, etc.  TSC X offset and row/column. */

/* Batch storage code configuration. */
struct Sccfg bsccfg[MAXBSC];/* PC 880, etc.  TSC X offset and row/column. */

/* v5.30 */
/* Calibration for 4-20 mA and RTD inputs on CPU board. */
unsigned char rtdalpha; /* PC 983.  Alpha for all RTD's. */
int clcalib[4]; /* PC 984, etc.  CPU board 4-20 ma input LOLO, LOHI, HILO and
                ** HIHI counts.
                */
int rtdcalib[4]; /* PC 988, etc. CPU board RTD input LOLO, LOHI, HILO and
                 ** HIHI counts.
                 */

/*
**
** Data Base
***************
*/

////
```

6.7 DanLoad 6000 v5.50 Database

The "database" is the name given to the stored program code values. The database for DanLoad 6000 v5.50 is shown below. (A "byte" is an unsigned char.)

```
////

/*
**
** Data Base
*/

struct {
    long passcode; /* PC 1, etc.  User X 9-digit passcode. */
    char userid[17]; /* PC 2, etc.  User X NUL terminated ID. */
    byte pwacess; /* PC 3, etc.  User X supervisor privilege. */
} pw[MAXUSER];

int nummtrs; /* PC 50.  Number of meters. */
struct {
    char mtrid[6]; /* PC 51, etc.  Meter X ID. */
    int mtrvlv; /* PC 52, etc.  Meter X flow control valve. */
    int radjfac; /* PC 729, etc.  Meter X ratio adjustment factor. */
    int fadjfac; /* PC 728, etc.  Meter X flow adjustment factor. */
    int ramppct; /* PC 730, etc.  Meter X ramp percentage. */
    int fpo; /* PC 756, etc.  Meter X MPMC factored pulse output. */
    int numblades; /* PC 757, etc.  Meter X CALMON number of blades. */
    long maxmaxchardev; /* PC 758, etc.  Meter X CALMON max. max. characteristic deviation. */
    long maxtotchardevs; /* PC 759, etc.  Meter X CALMON max. total characteristic deviations.
*/
    unsigned char calibsmplstat; /* PC 760, etc.  Meter X calibration sample status. */
    unsigned char calmonanalmthd; /* PC 731, etc.  Meter X CALMON analysis method. */
} mtrparms[MAXMTR];

byte opmode; /* PC 25.  Operating mode. */
byte language; /* PC 28.  Language. */
byte dtefmt; /* PC 38.  Date format. */
byte dtesep; /* PC 39.  Date separator. */
byte decsep; /* PC 40.  Decimal separator. */
byte units; /* PC 29.  Product units. */
byte numdataitems; /* PC 30.  Number of data items. */
char dataprompt[5][33]; /* PC 31, etc.  Data item X NUL terminated prompt. */
int pmt_timout; /* PC 36.  Prompt time-out. */
byte chk_disp; /* PC 37.  Check display. */
byte unit_type; /* PC 26.  Unit type. */
byte fcv_type; /* PC 27.  Valve type. */
unsigned char dspdr; /* PC 41.  Display data rate. */
int transeqnum; /* PC 42.  Transaction #. */
int batchseqnum; /* PC 43.  Batch #.  */
unsigned char roundmthd; /* PC 45.  Rounding method. */
int densscale; /* PC 46.  Density/gravity scale. */
int maxbatch; /* PC 47.  Maximum batches/transaction. */

int numvalves; /* PC 48.  Number of (flow control) valves. */
int initflwtime; /* PC 49.  Initial flow time. */

/* Component parameters */
int numcomps; /* PC 65.  Number of components. */
struct {
```

```
    char compid[17]; /* PC 66, etc.  Component X NUL terminated ID. */
    int compmtr; /* PC 67, etc.  Component X meter number. */
    long massadj; /* PC 68, etc.  Component X mass adjustment. */
} comp[MAXCOMPS];

/* Delivery parameters */
long maxpreset; /* PC 78.  Maximum preset volume. */
long minpreset; /* PC 79.  Minimum preset volume. */
byte predeltype; /* PC 80.  Preset/delivery type. */
byte deldisptype; /* PC 81.  Delivery display type. */
byte stopkey; /* PC 82.  Stop key action. */
long fallbackvol; /* PC 83.  Fall back volume. */

struct Compdel {
    long loflwstart; /* PC 84, etc.  Component X low flow start volume. */
    long loflwrestart; /* PC 85, etc.  Component X low flow restart volume. */
    long loflwstop; /* PC 86, etc.  Component X low flow stop volume. */
    int linepackdly; /* PC 87, etc.  Component X line pack delay. */
    int pumpstopdly; /* PC 88, etc.  Component X pump stop delay. */
    int blkvalvedly; /* PC 89, etc.  Component X block valve delay. */
} compdel[MAXCOMPS];
int maxrampclicks; /* PC 108.  Ramp clicks. */
int maxmaintclicks; /* PC 109.  Maintenance clicks. */
int addpmpdly; /* PC 110.  Additive pump stop delay. */
int primcmp; /* PC 111.  Primary component. */

/* Digital Valve Parameters */
struct {
    int dvlferrpct; /* PC 112, etc.  Valve X low flow % error. */
    int dvhferrpct; /* PC 113, etc.  Valve X high flow % error. */
    int dvdly; /* PC 114, etc.  Valve X maint click adjustment. */
    byte openmthd; /* PC 115, etc.  Valve X open method. */
} dvparms[MAXVLV];

/* Pulse per unit (DAU) parameters */
int numdaus; /* PC 128.  Number of PPU's. */
struct Dau {
    struct Dauctl {
        byte gross;
        byte net;
    } dauctl[MAXMTR]; /* PC 129, etc.  PPU X control meters. */
    int daufactor; /* PC 130, etc.  PPU X factor. */
    int daupulwidth; /* PC 131, etc.  PPU X pulse width. */
} dau[MAXDAU];

/* Additive Parameters */
int numadds; /* PC 135.  Number of additives. */
byte addselmthd; /* PC 136.  Additive selection method. */
int injpct; /* PC 137.  Additive inject % */
int addclnlinvol; /* PC 138.  Additive clean line volume. */
byte addunits; /* PC 44. Additive units. */
struct Add {
    struct Addctl {
        byte gross;
        byte net;
    } addctl[MAXMTR]; /* PC 139, etc.  Additive X control meters. */
    int ratiovol; /* PC 140, etc.  Additive X ratio volume in whole units. */
    unsigned char injmthd; /* PC 141, etc.  Additive X injection method. */
    long kfac; /* PC 142, etc.  Additive X volume per pulse or K-factor in
               ** hundredths.
               */
    long reqdvol; /* PC 143, etc.  Additive X volume (in external product
```

```
                 ** units x 10,000) per 1000 units of product.
                 */
} add[MAXADDS];

/* Factors  */
int numfacs; /* PC 169.  Number of factors/component. */
byte mfacmthd; /* PC 170.  Meter factor method. */
struct Fac {
    long nomkfac; /* PC 171, etc.  Component X nominal K-factor. */
    long mstrmfac; /* PC 172, etc.  Component X master meter factor. */
    struct Mtrfac {
        long flwrate; /* PC 174, etc.  Component X flow rate Y. */
        long mfac; /* PC 175.  Component X meter factor Y. */
    } mtrfac[MAXFACS];
} fac[MAXCOMPS];
int mstrmfacpct; /* PC 215.  Master meter factor percentage. */
int adjmfacpct; /* PC 216.  Adjacent meter factor percentage. */
char grsunitsmnemo[7]; /* PC 217.  Gross quantity units mnemonic. */
char stdunitsmnemo[7]; /* PC 218.  Standard quantity units mnemonic. */
unsigned char userestart; /* PC 219.  Use restart quantity. */

/* Alarm Parameters */
int secalrst; /* PC 220.  Secondary alarm reset (secs). */
byte lowflwact; /* PC 221.  Low flow alarm action. */
long lowflwlim; /* PC 222.  Minimum flow rate. */
int lowflwtime; /* PC 223.  Low flow time. */
byte hiflwact; /* PC 224.  High flow alarm action. */
long hiflwlim; /* PC 225.  Maximum flow rate. */
int hiflwtime; /* PC 226.  High flow time. */
int ovrunlim; /* PC 227.  Overrun limit volume. */
byte undrflwact; /* PC 228.  Underflow alarm action. */
int undrflwlim; /* PC 229.  Underflow limit volume. */
byte noflwact; /* PC 230.  No flow alarm action. */
int noflwtime; /* PC 231.  No flow time. */
int unauthflw; /* PC 232.  Unauth. flow limit volume. */
int pserr; /* PC 233.  Pulse security error limit. */
long psresetcnt; /* PC 234.  Pulse security reset count. */
byte dlfailact; /* PC 235.  Data logging alarm action. */
int commtimout[2]; /* PC 236's and 237.  Comms fail time-out per channel. */
byte tempfailact; /* PC 238.  Temperature fail alarm action. */
int tempmin; /* PC 239.  Min. temperature. */
int tempmax; /* PC 240.  Max. temperature. */
byte densfailact; /* PC 241.  Density fail alarm action. */
long densmin; /* PC 242.  Minimum density/gravity. */
long densmax; /* PC 243.  Maximum density/gravity. */
long presmin; /* PC 244.  Minimum pressure. */
long presmax; /* PC 245.  Maximum pressure. */
int adderrlim; /* PC 246.  Additive error limit. */
int addfbckcnt; /* PC 247.  # pulses per additive cycle or # seconds. */
int bvclsdly; /* PC 248.  Block valve time. */

struct Safety {
    byte action; /* PC 249, etc.  Safety circuit X alarm action. */
    char msg[33]; /* PC 250, etc.  Safety ciruit X alarm message. */
} safety[MAXSAFETY];

byte sctype[MAXSAFETY]; /* PC 265, etc.  Safety circuit X type. */

long unauthaddflw; /* PC 271.  Unauthorized additive flow volume. */
int praddminpct; /* PC 272.  Additive volume verification minimum error
                 ** percentage.
                 */
```

```
int praddmaxpct; /* PC 273.  Additive volume verification maximum error
                 ** percentage.
                 */

byte storeact; /* PC 274.  Storage alarm action. */
byte poweract; /* PC 275.  Power fail alarm action. */
byte rampact; /* PC 276.  Ramp down alarm action. */
int ramptime; /* PC 277.  Ramp down time. */
int add1000errlim; /* PC 278.  Additive per 1000 error limit. */
int endbatim; /* PC 279.  End batch time. */

/* I/O Parameters */
unsigned char islot[9]; /* Slots J1 thru J7, DUART and ARCNET. */

struct Ioctl mtrin[MAXMTRIN];
struct Ioctl rtdin[MAXRTDIN];
struct Ioctl anain[MAXANAIN];
struct Ioctl digin[MAXDIGIN];
struct Ioctl digout[MAXDIGOUT];

struct Almout {
    int out; /* PC 287, etc.  Alarm output X. */
    unsigned long mask; /* PC 269, etc.  Alarm output X mask. */
} almout[MAXALMOUT];

int pmpctlout[MAXCOMPS]; /* PC 288, etc.  Component X pump output. */
struct Vlvio {
    int sol1; /* PC 292, etc.  Valve X solenoid 1. */
    int sol2; /* PC 293, etc.  Valve X solenoid 2. */
    int stemsw1; /* PC 294, etc.  Valve X stem switch 1. */
    int stemsw2; /* PC 295, etc.  Valve X stem switch 2. */
    int clsinp; /* PC 296, etc.  Valve X close input. */
} vlvio[MAXVLV];
byte sidemthd; /* PC 312.  Side detect method. */
int dauout[MAXDAU]; /* PC 313, etc.  PPU output X. */
int trip1out; /* PC 315.  Trip 1 output. */
int trip2out; /* PC 316.  Trip 2 output. */
int trip3out; /* PC 317.  Trip 3 output. */
struct Mtrtmp {
    int mtempinp; /* PC 318, etc.  Meter X temperature input. */
    long moffset; /* PC 319, etc.  Meter X offset (Ohms). */
    int mpresinp; /* PC 320, etc.  Meter X pressure input. */
    long pres4; /* PC 464, etc.  Meter X pressure at 4 mA. */
    long pres20; /* PC 465, etc.  Meter X pressure at 20 mA. */
    int mdensinp; /* PC 321, etc.  Meter X density input. */
    long dens4; /* PC 466, etc.  Meter X density at 4 mA. */
    long dens20; /* PC 467, etc.  Meter X density at 20 mA. */
    int flowinp; /* PC 322, etc.  Meter X flow input. */
} mtrtmp[MAXMTR];

int numcpmtszin; /* PC 341.  Number of compartment size inputs. */ /* v5.50 */
byte startmthd; /* PC 342.  Start batch method. */ /* v5.50 */
int autochng; /* PC 343.  Auto/manual change-over input. */
int primalrst; /* PC 344.  Primary alarm reset input. */

int safetyinp[MAXSAFETY]; /* PC 345, etc.  Safety circuit X input. */
byte temprange; /* PC 353.  Range for RTD inputs on analog inputs boards. */
int opdly; /* PC 354.  Delay after outputs, e.g. for DART 2. */

byte pridsptyp; /* PC 355.  Primary display type. */
int priminlt; /* PC 356.  Primary display min. light. */
int primaxlt; /* PC 357.  Primary display max. light. */
```

```
byte secdsptyp; /* PC 358.  Secondary display type. */
int secminlt; /* PC 359.  Secondary display min. light. */
int secmaxlt; /* PC 360.  Secondary display max. light. */

int recipeinp[MAXRECIN]; /* PC 361, etc.  Recipe X selection input. */
int recipeout[MAXRECOUT]; /* PC 367, etc.  Recipe X selection output. */
int levelinp; /* PC 374.  Intermediate level input. */ /* v5.50 */
int hatchinp; /* PC 375.  Hatch sensor input. */ /* v5.50 */
int swing2; /* PC 376.  Swing-arm switch 2. */
int swing1; /* PC 377.  Swing-arm switch 1. */
int loflwinp; /* PC 378.  Stay in low flow input. */

unsigned char fpobasis; /* PC 752.  Factored pulse outputs basis. */
int calmonerr; /* PC 753.  Calibration monitoring error limit. */
int calmonrcnt; /* PC 754.  Calibration monitoring reset count. */
byte calibfailact; /* PC 755.  Calibration failure alarm action for CALMON. */

struct Addinp {
       int ratio; /* PC 380, etc.  Additive X ratio output. */
       int feedback; /* PC 381, etc.  Additive X feedback input. */
       int select; /* PC 382, etc.  Additive X selection input. */
    int pmpctlout; /* PC 800, etc.  Additive X pump control output. */
    int bvout; /* PC 801, etc.  Additive X block valve output. */
} addinp[MAXADDS];

int addclnlinout; /* PC 398.  Additive clean line output. */
int addmtrflushout; /* PC 399.  Additive meter flush output. */

struct Cmpio {
       int bvout; /* PC 400, etc.  Component X block valve output. */
       int bvin; /* PC 401, etc.  Component X block valve input. */
} cmpio[MAXCOMPS];
int combio[15]; /* PC 408, etc.  Component combination X. */
int endbaout; /* PC 423.  End batch output. */
int endbain; /* PC 424.  End batch input. */
unsigned long addmtrflushpulses; /* PC 425.  Additive meter flush pulses. */

long buoyancy; /* PC 426.  Density adjustment for buoyancy in air. */
byte tempunits; /* PC 427.  Temperature units. */
byte densunits; /* PC 428.  Density units. */
byte presunits; /* PC 429.  Pressure units. */
int reftemp; /* PC 430.  Reference temperature. */
long tempsmpl; /* PC 431.  Sample volume. */

struct Cmptemp {
    byte option; /* PC 432, etc.  Component X temperature correction option. */
    long alpha; /* PC 433, etc.  Component X alpha. */
    int backup; /* PC 434, etc.  Component X backup temperature. */
} cmptemp[MAXCOMPS];
struct Cmppres {
    byte option; /* PC 444, etc.  Component X pressure correction option. */
    long pffac; /* PC 445, etc.  Component X F-factor. */
    long backup; /* PC 446, etc.  Component X backup pressure. */
} cmppres[MAXCOMPS];
struct Cmpdens {
    byte option; /* PC 456, etc.  Component X density option. */
    long backup; /* PC 457, etc.  Component X backup density/gravity. */
} cmpdens[MAXCOMPS];

int numrecipes; /* PC 480.  Number of recipes. */
struct Recipe {
    char name[17]; /* PC 481, etc.  Recipe X NUL terminated name. */
```

```
    int cmppct[MAXCOMPS]; /* PC 482, etc.  Recipe X component percentages. */
    struct Cmpseq {
        int cmpno;
    } cmpseq[MAXCOMPS]; /* PC 486, etc.  Recipe X sequence. */
} recipe[MAXRECIPES];

/* v5.30 */
int commsaddr; /* PC 662.  Communications address. */

struct Commun {
    byte mode; /* PC 663, etc.  Channel X mode. */
    byte baud; /* PC 664, etc.  Channel X data rate. */
    byte dbits; /* PC 665, etc.  Channel X word size. */
    byte sbits; /* PC 666, etc.  Channel X stop bits. */
    byte parity; /* PC 667, etc.  Channel X parity. */
} commun[2];

unsigned char commalarmmanb; /* PC 673.  Channel B alarm in manual. */

unsigned char ddmode; /* PC 679.  Dynamic data display mode. */
int ddpc[16]; /* PC 680, etc.  Dynamic data display element X data code. */

byte batch; /* PC 696.  Batch summary. */
byte trans; /* PC 697.  Transaction summary. */
byte almlog; /* PC 698.  Alarm log. */
byte pflog; /* PC 699.  Power fail log. */
byte progmode; /* PC 700.  Program mode entry/exit. */
byte wmsw; /* PC 701.  W&M switch opened/closed. */
byte pcchnglog; /* PC 702.  Program value change. */
byte cfglog; /* PC 703.  Configuration summary data log. */
byte crash; /* PC 704.  Crash memory summary. */
byte totlog; /* PC 705.  Totalizers. */
byte trtkauto; /* PC 706.  Transaction ticket. */
byte trtk; /* PC 708.  Transaction ticket reprint. */
byte thrulog; /* PC 708.  Thruput. */
byte dlogseq; /* PC 709.  Sequence numbers. */
int cuthour; /* PC 710.  Cutoff hour. */

/* Blending. */
int rampdownpct; /* PC 711.  The percentage of the blend's current flow rate, etc. */
int deltarate; /* PC 712.  Change in flow rate for batch fall back. */
long loflwstart; /* PC 713.  Low flow start volume for in-line blend. */
int clnlinvol; /* PC 714.  Clean line volume. */
int lopropfac; /* PC 715.  The meter factor flow rate for low proportion
                ** components of an in-line blend.
                */
int blchkvol; /* PC 716.  Correct after volume.  Volume after which blend
               ** ratio adjustment begins.
               */
int blchkvol2; /* PC 717.  Alarm after volume.  Volume after which blend
                ** ratio alarm is checked.
                */
int vadjust; /* PC 718.  Volume over which blend adjustment is made. */
unsigned char comppctmthd; /* PC 719.  Component % display. */
unsigned char blenderrmthd; /* PC 722.  Blend error method. */
int blnddev; /* PC 723.  Max. dev. % */
int blndtol; /* PC 724.  Blend tol. % */
int blnddevvol; /* PC 725.  Max. dev. volume. */
int blndtolvol; /* PC 726.  Blend tol. volume. */
int blendsmpl; /* PC 727.  Blend sample volume. */
long stoprate[MAXCOMPS]; /* PC 173.  Component X stop rate for in-line blend. */
long rflwrate[MAXRFLWRATE][2]; /* PC 776, 777, etc.  Recipe "group" low and
```

```
                                ** high flow rates.
                                */

unsigned char pttype; /* PC 818.  Pressure transmitter types. */
unsigned char pemthd; /* PC 819.  Pe method. */
unsigned char pinchmthd; /* PC 820.  Pinch back method. */
long pinchpres; /* PC 821.  Pinch back pressure. */
long atmospres; /* PC 822.  Atmospheric pressure. */
long pinchqty; /* PC 823.  Pinch back qty. */
struct Pe_lin {
    int temp; /* PC 824, etc.  Tx. */
    long pres; /* PC 825, etc.  Pe at Tx. */
} pe_lin[MAXPELIN]; /* T/Pe pairs for Pe linearization. */

/* Transaction storage. */

unsigned char trrecl; /* PC 834.  Transaction record length. */
int trnumpgs; /* PC 835.  Transaction file start page (/ 100) and # pages (% 100). */
unsigned char barecl; /* PC 836.  Batch record length. */
int banumpgs; /* PC 837.  Batch file start page (/ 100) and # pages (% 100). */
unsigned char trtkfmt; /* PC 838.  Transaction ticket predefined format. */
unsigned char timouttr; /* PC 839.  Time out transaction. */

/* Transaction storage code configuration. */
struct Sccfg tsccfg[MAXTSC]; /* PC 840, etc.  TSC X offset and row/column. */

/* Batch storage code configuration. */
struct Sccfg bsccfg[MAXBSC];/* PC 880, etc.  TSC X offset and row/column. */

/* v5.30 */
/* Calibration for 4-20 mA and RTD inputs on CPU board. */
unsigned char rtdalpha; /* PC 983.  Alpha for all RTD's. */
int clcalib[4]; /* PC 984, etc.  CPU board 4-20 ma input LOLO, LOHI, HILO and
                ** HIHI counts.
                */
int rtdcalib[4]; /* PC 988, etc. CPU board RTD input LOLO, LOHI, HILO and
                 ** HIHI counts.
                 */

/*
**
** Data Base
***************
*/

////
```

6.8 DanLoad 6000 v5.60 Database

```
////

/*
**
** Data Base
*/

struct {
    long passcode; /* PC 1, etc.  User X 9-digit passcode. */
    char userid[17]; /* PC 2, etc.  User X NUL terminated ID. */
    byte pwacess; /* PC 3, etc.  User X supervisor privilege. */
} pw[MAXUSER];

int nummtrs; /* PC 50.  Number of meters. */
struct {
    char mtrid[6]; /* PC 51, etc.  Meter X ID. */
    int mtrvlv; /* PC 52, etc.  Meter X flow control valve. */
    int radjfac; /* PC 729, etc.  Meter X ratio adjustment factor. */
    int fadjfac; /* PC 728, etc.  Meter X flow adjustment factor. */
    int ramppct; /* PC 730, etc.  Meter X ramp percentage. */
    int fpo; /* PC 756, etc.  Meter X MPMC factored pulse output. */
    int numblades; /* PC 757, etc.  Meter X CALMON number of blades. */
    long maxmaxchardev; /* PC 758, etc.  Meter X CALMON max. max. characteristic deviation. */
    long maxtotchardevs; /* PC 759, etc.  Meter X CALMON max. total characteristic deviations.
*/
    unsigned char calibsmplstat; /* PC 760, etc.  Meter X calibration sample status. */
    unsigned char calmonanalmthd; /* PC 731, etc.  Meter X CALMON analysis method. */
} mtrparms[MAXMTR];

byte opmode; /* PC 25.  Operating mode. */
byte language; /* PC 28.  Language. */
byte dtefmt; /* PC 38.  Date format. */
byte dtesep; /* PC 39.  Date separator. */
byte decsep; /* PC 40.  Decimal separator. */
byte produnits; /* PC 29.  Product units. */
byte numdataitems; /* PC 30.  Number of data items. */
char dataprompt[5][33]; /* PC 31, etc.  Data item X NUL terminated prompt. */
int pmt_timout; /* PC 36.  Prompt time-out. */
byte chk_disp; /* PC 37.  Check display. */
byte unit_type; /* PC 26.  Unit type. */
byte fcv_type; /* PC 27.  Valve type. */
unsigned char dspdr; /* PC 41.  Display data rate. */
int transeqnum; /* PC 42.  Transaction #. */
int batchseqnum; /* PC 43.  Batch #.  */
unsigned char roundmthd; /* PC 45.  Rounding method. */
int densscale; /* PC 46.  Density/gravity scale. */
int maxbatch; /* PC 47.  Maximum batches/transaction. */

int numvalves; /* PC 48.  Number of (flow control) valves. */
int initflwtime; /* PC 49.  Initial flow time. */

/* Component parameters */
int numcomps; /* PC 65.  Number of components. */
struct {
    char compid[17]; /* PC 66, etc.  Component X NUL terminated ID. */
    int compmtr; /* PC 67, etc.  Component X meter number. */
    long massadj; /* PC 68, etc.  Component X mass adjustment. */
} comp[MAXCOMPS];
```

```
/* Delivery parameters */
long maxpreset; /* PC 78.  Maximum preset volume. */
long minpreset; /* PC 79.  Minimum preset volume. */
byte predeltype; /* PC 80.  Preset/delivery type. */
byte deldisptype; /* PC 81.  Delivery display type. */
byte stopkey; /* PC 82.  Stop key action. */
long fallbackvol; /* PC 83.  Fall back volume. */

struct Compdel {
    long loflwstart; /* PC 84, etc.  Component X low flow start volume. */
    long loflwrestart; /* PC 85, etc.  Component X low flow restart volume. */
    long loflwstop; /* PC 86, etc.  Component X low flow stop volume. */
    int linepackdly; /* PC 87, etc.  Component X line pack delay. */
    int pumpstopdly; /* PC 88, etc.  Component X pump stop delay. */
    int blkvalvedly; /* PC 89, etc.  Component X block valve delay. */
} compdel[MAXCOMPS];
int maxrampclicks; /* PC 108.  Ramp clicks. */
int maxmaintclicks; /* PC 109.  Maintenance clicks. */
int addpmpdly; /* PC 110.  Additive pump stop delay. */
int primcmp; /* PC 111.  Primary component. */

/* Digital Valve Parameters */
struct {
    int dvlferrpct; /* PC 112, etc.  Valve X low flow % error. */
    int dvhferrpct; /* PC 113, etc.  Valve X high flow % error. */
    int dvdly; /* PC 114, etc.  Valve X maint click adjustment. */
    byte openmthd; /* PC 115, etc.  Valve X open method. */
} dvparms[MAXVLV];

/* Pulse per unit (DAU) parameters */
int numdaus; /* PC 128.  Number of PPU's. */
struct Dau {
    struct Dauctl {
        byte gross;
        byte net;
    } dauctl[MAXMTR]; /* PC 129, etc.  PPU X control meters. */
    int daufactor; /* PC 130, etc.  PPU X factor. */
    int daupulwidth; /* PC 131, etc.  PPU X pulse width. */
} dau[MAXDAU];

/* Additive Parameters */
int numadds; /* PC 135.  Number of additives. */
byte addselmthd; /* PC 136.  Additive selection method. */
int injpct; /* PC 137.  Additive inject % */
int addclnlinvol; /* PC 138.  Additive clean line volume. */
byte addunits; /* PC 44. Additive units. */
struct Add {
    struct Addctl {
        byte gross;
        byte net;
    } addctl[MAXMTR]; /* PC 139, etc.  Additive X control meters. */
    int ratiovol; /* PC 140, etc.  Additive X ratio volume in whole units. */
    unsigned char injmthd; /* PC 141, etc.  Additive X injection method. */
    long kfac; /* PC 142, etc.  Additive X volume per pulse or K-factor in
                ** hundredths.
                */
    long reqdvol; /* PC 143, etc.  Additive X volume (in external product
                   ** units x 10,000) per 1000 units of product.
                   */
} add[MAXADDS];

/* Factors  */
```

```
int numfacs; /* PC 169.  Number of factors/component. */
byte mfacmthd; /* PC 170.  Meter factor method. */
struct Fac {
    long nomkfac; /* PC 171, etc.  Component X nominal K-factor. */
    long mstrmfac; /* PC 172, etc.  Component X master meter factor. */
    struct Mtrfac {
        long flwrate; /* PC 174, etc.  Component X flow rate Y. */
        long mfac; /* PC 175.  Component X meter factor Y. */
    } mtrfac[MAXFACS];
} fac[MAXCOMPS];
int mstrmfacpct; /* PC 215.  Master meter factor percentage. */
int adjmfacpct; /* PC 216.  Adjacent meter factor percentage. */
char grsunitsmnemo[7]; /* PC 217.  Gross quantity units mnemonic. */
char stdunitsmnemo[7]; /* PC 218.  Standard quantity units mnemonic. */
unsigned char userestart; /* PC 219.  Use restart quantity. */

/* Alarm Parameters */
int secalrst; /* PC 220.  Secondary alarm reset (secs). */
byte lowflwact; /* PC 221.  Low flow alarm action. */
long lowflwlim; /* PC 222.  Minimum flow rate. */
int lowflwtime; /* PC 223.  Low flow time. */
byte hiflwact; /* PC 224.  High flow alarm action. */
long hiflwlim; /* PC 225.  Maximum flow rate. */
int hiflwtime; /* PC 226.  High flow time. */
int ovrunlim; /* PC 227.  Overrun limit volume. */
byte undrflwact; /* PC 228.  Underflow alarm action. */
int undrflwlim; /* PC 229.  Underflow limit volume. */
byte noflwact; /* PC 230.  No flow alarm action. */
int noflwtime; /* PC 231.  No flow time. */
int unauthflw; /* PC 232.  Unauth. flow limit volume. */
int pserr; /* PC 233.  Pulse security error limit. */
long psresetcnt; /* PC 234.  Pulse security reset count. */
byte dlfailact; /* PC 235.  Data logging alarm action. */
int commtimout[2]; /* PC 236's and 237.  Comms fail time-out per channel. */
byte tempfailact; /* PC 238.  Temperature fail alarm action. */
int tempmin; /* PC 239.  Min. temperature. */
int tempmax; /* PC 240.  Max. temperature. */
byte densfailact; /* PC 241.  Density fail alarm action. */
long densmin; /* PC 242.  Minimum density/gravity. */
long densmax; /* PC 243.  Maximum density/gravity. */
long presmin; /* PC 244.  Minimum pressure. */
long presmax; /* PC 245.  Maximum pressure. */
int adderrlim; /* PC 246.  Additive error limit. */
int addfbckcnt; /* PC 247.  # pulses per additive cycle or # seconds. */
int bvclsdly; /* PC 248.  Block valve time. */

struct Safety {
    byte action; /* PC 249, etc.  Safety circuit X alarm action. */
    char msg[33]; /* PC 250, etc.  Safety ciruit X alarm message. */
} safety[MAXSAFETY];

byte sctype[MAXSAFETY]; /* PC 265, etc.  Safety circuit X type. */

long unauthaddflw; /* PC 271.  Unauthorized additive flow volume. */
int praddminpct; /* PC 272.  Additive volume verification minimum error
                 ** percentage.
                 */
int praddmaxpct; /* PC 273.  Additive volume verification maximum error
                 ** percentage.
                 */

byte storeact; /* PC 274.  Storage alarm action. */
```

```
byte poweract; /* PC 275.  Power fail alarm action. */
byte rampact; /* PC 276.  Ramp down alarm action. */
int ramptime; /* PC 277.  Ramp down time. */
int add1000errlim; /* PC 278.  Additive per 1000 error limit. */
int endtim; /* PC 279.  End time. */

/* I/O Parameters */
unsigned char islot[9]; /* Slots J1 thru J7, DUART and ARCNET. */

struct Ioctl mtrin[MAXMTRIN];
struct Ioctl rtdin[MAXRTDIN];
struct Ioctl anain[MAXANAIN];
struct Ioctl digin[MAXDIGIN];
struct Ioctl digout[MAXDIGOUT];

struct Almout {
    int out; /* PC 287, etc.  Alarm output X. */
    unsigned long mask; /* PC 269, etc.  Alarm output X mask. */
} almout[MAXALMOUT];

int pmpctlout[MAXCOMPS]; /* PC 288, etc.  Component X pump output. */
struct Vlvio {
    int sol1; /* PC 292, etc.  Valve X solenoid 1. */
    int sol2; /* PC 293, etc.  Valve X solenoid 2. */
    int stemsw1; /* PC 294, etc.  Valve X stem switch 1. */
    int stemsw2; /* PC 295, etc.  Valve X stem switch 2. */
    int clsinp; /* PC 296, etc.  Valve X close input. */
} vlvio[MAXVLV];
byte sidemthd; /* PC 312.  Side detect method. */
int dauout[MAXDAU]; /* PC 313, etc.  PPU output X. */
int trip1out; /* PC 315.  Trip 1 output. */
int trip2out; /* PC 316.  Trip 2 output. */
int trip3out; /* PC 317.  Trip 3 output. */
struct Mtrtmp {
    int mtempinp; /* PC 318, etc.  Meter X temperature input. */
    long moffset; /* PC 319, etc.  Meter X offset (Ohms). */
    int mpresinp; /* PC 320, etc.  Meter X pressure input. */
    long pres4; /* PC 464, etc.  Meter X pressure at 4 mA. */
    long pres20; /* PC 465, etc.  Meter X pressure at 20 mA. */
    int mdensinp; /* PC 321, etc.  Meter X density input. */
    long dens4; /* PC 466, etc.  Meter X density at 4 mA. */
    long dens20; /* PC 467, etc.  Meter X density at 20 mA. */
    int flowinp; /* PC 322, etc.  Meter X flow input. */
} mtrtmp[MAXMTR];

int numcpmtszin; /* PC 341.  Number of compartment size inputs. */ /* v5.50 */
byte startmthd; /* PC 342.  Start batch method. */ /* v5.50 */
int autochng; /* PC 343.  Auto/manual change-over input. */
int primalrst; /* PC 344.  Primary alarm reset input. */

int safetyinp[MAXSAFETY]; /* PC 345, etc.  Safety circuit X input. */
byte temprange; /* PC 353.  Range for RTD inputs on analog inputs boards. */
int opdly; /* PC 354.  Delay after outputs, e.g. for DART 2. */

byte pridsptyp; /* PC 355.  Primary display type. */
int priminlt; /* PC 356.  Primary display min. light. */
int primaxlt; /* PC 357.  Primary display max. light. */
byte secdsptyp; /* PC 358.  Secondary display type. */
int secminlt; /* PC 359.  Secondary display min. light. */
int secmaxlt; /* PC 360.  Secondary display max. light. */

int recipeinp[MAXRECIN]; /* PC 361, etc.  Recipe X selection input. */
```

```
int recipeout[MAXRECOUT]; /* PC 367, etc.  Recipe X selection output. */

unsigned char endmthd; /* PC 373.  End output method. */ /* v5.60 */

int levelinp; /* PC 374.  Intermediate level input. */ /* v5.50 */
int hatchinp; /* PC 375.  Hatch sensor input. */ /* v5.50 */
int swing2; /* PC 376.  Swing-arm switch 2. */
int swing1; /* PC 377.  Swing-arm switch 1. */
int loflwinp; /* PC 378.  Stay in low flow input. */

unsigned char fpobasis; /* PC 752.  Factored pulse outputs basis. */
int calmonerr; /* PC 753.  Calibration monitoring error limit. */
int calmonrcnt; /* PC 754.  Calibration monitoring reset count. */
byte calibfailact; /* PC 755.  Calibration failure alarm action for CALMON. */

struct Addinp {
       int ratio; /* PC 380, etc.  Additive X ratio output. */
       int feedback; /* PC 381, etc.  Additive X feedback input. */
       int select; /* PC 382, etc.  Additive X selection input. */
    int pmpctlout; /* PC 800, etc.  Additive X pump control output. */
    int bvout; /* PC 801, etc.  Additive X block valve output. */
} addinp[MAXADDS];

int addclnlinout; /* PC 398.  Additive clean line output. */
int addmtrflushout; /* PC 399.  Additive meter flush output. */

struct Cmpio {
       int bvout; /* PC 400, etc.  Component X block valve output. */
       int bvin; /* PC 401, etc.  Component X block valve input. */
} cmpio[MAXCOMPS];
int combio[15]; /* PC 408, etc.  Component combination X. */
int endout; /* PC 423.  End output. */
int endinp; /* PC 424.  End input. */
unsigned long addmtrflushpulses; /* PC 425.  Additive meter flush pulses. */

long buoyancy; /* PC 426.  Density adjustment for buoyancy in air. */
byte tempunits; /* PC 427.  Temperature units. */
byte densunits; /* PC 428.  Density units. */
byte presunits; /* PC 429.  Pressure units. */
int reftemp; /* PC 430.  Reference temperature. */
long tempsmpl; /* PC 431.  Sample volume. */

struct Cmptemp {
    byte option; /* PC 432, etc.  Component X temperature correction option. */
    long alpha; /* PC 433, etc.  Component X alpha. */
    int backup; /* PC 434, etc.  Component X backup temperature. */
} cmptemp[MAXCOMPS];
struct Cmppres {
    byte option; /* PC 444, etc.  Component X pressure correction option. */
    long pffac; /* PC 445, etc.  Component X F-factor. */
    long backup; /* PC 446, etc.  Component X backup pressure. */
} cmppres[MAXCOMPS];
struct Cmpdens {
    byte option; /* PC 456, etc.  Component X density option. */
    long backup; /* PC 457, etc.  Component X backup density/gravity. */
} cmpdens[MAXCOMPS];

int numrecipes; /* PC 480.  Number of recipes. */
struct Recipe {
    char name[17]; /* PC 481, etc.  Recipe X NUL terminated name. */
    int cmppct[MAXCOMPS]; /* PC 482, etc.  Recipe X component percentages. */
    struct Cmpseq {
```

```
        int cmpno;
    } cmpseq[MAXCOMPS]; /* PC 486, etc.  Recipe X sequence. */
} recipe[MAXRECIPES];

int numaddcalibcycles; /* PC 662.  # additive calibration injection cycles. */

/* v5.30 */
int commsaddr; /* PC 662.  Communications address. */

struct Commun {
    byte mode; /* PC 663, etc.  Channel X mode. */
    byte baud; /* PC 664, etc.  Channel X data rate. */
    byte dbits; /* PC 665, etc.  Channel X word size. */
    byte sbits; /* PC 666, etc.  Channel X stop bits. */
    byte parity; /* PC 667, etc.  Channel X parity. */
} commun[2];

unsigned char commalarmmanb; /* PC 673.  Channel B alarm in manual. */

unsigned char ddmode; /* PC 679.  Dynamic data display mode. */
int ddpc[16]; /* PC 680, etc.  Dynamic data display element X data code. */

byte batch; /* PC 696.  Batch summary. */
byte trans; /* PC 697.  Transaction summary. */
byte almlog; /* PC 698.  Alarm log. */
byte pflog; /* PC 699.  Power fail log. */
byte progmode; /* PC 700.  Program mode entry/exit. */
byte wmsw; /* PC 701.  W&M switch opened/closed. */
byte pcchnglog; /* PC 702.  Program value change. */
byte cfglog; /* PC 703.  Configuration summary data log. */
byte crash; /* PC 704.  Crash memory summary. */
byte totlog; /* PC 705.  Totalizers. */
byte trtkauto; /* PC 706.  Transaction ticket. */
byte trtk; /* PC 708.  Transaction ticket reprint. */
byte thrulog; /* PC 708.  Thruput. */
byte dlogseq; /* PC 709.  Sequence numbers. */
int cuthour; /* PC 710.  Cutoff hour. */

/* Blending. */
int rampdownpct; /* PC 711.  The percentage of the blend's current flow rate, etc. */
int deltarate; /* PC 712.  Change in flow rate for batch fall back. */
long loflwstart; /* PC 713.  Low flow start volume for in-line blend. */
int clnlinvol; /* PC 714.  Clean line volume. */
int lopropfac; /* PC 715.  The meter factor flow rate for low proportion
                ** components of an in-line blend.
                */
int blchkvol; /* PC 716.  Correct after volume.  Volume after which blend
               ** ratio adjustment begins.
               */
int blchkvol2; /* PC 717.  Alarm after volume.  Volume after which blend
                ** ratio alarm is checked.
                */
int vadjust; /* PC 718.  Volume over which blend adjustment is made. */
unsigned char comppctmthd; /* PC 719.  Component % display. */
int addcalibdly; /* PC 720.  Additive calibration delay. */
unsigned char blenderrmthd; /* PC 722.  Blend error method. */
int blnddev; /* PC 723.  Max. dev. % */
int blndtol; /* PC 724.  Blend tol. % */
int blnddevvol; /* PC 725.  Max. dev. volume. */
int blndtolvol; /* PC 726.  Blend tol. volume. */
int blendsmpl; /* PC 727.  Blend sample volume. */
long stoprate[MAXCOMPS]; /* PC 173.  Component X stop rate for in-line blend. */
```

```
long rflwrate[MAXRFLWRATE][2]; /* PC 776, 777, etc.  Recipe "group" low and
                               ** high flow rates.
                               */

unsigned char pttype; /* PC 818.  Pressure transmitter types. */
unsigned char pemthd; /* PC 819.  Pe method. */
unsigned char pinchmthd; /* PC 820.  Pinch back method. */
long pinchpres; /* PC 821.  Pinch back pressure. */
long atmospres; /* PC 822.  Atmospheric pressure. */
long pinchqty; /* PC 823.  Pinch back qty. */
struct Pe_lin {
    int temp; /* PC 824, etc.  Tx. */
    long pres; /* PC 825, etc.  Pe at Tx. */
} pe_lin[MAXPELIN]; /* T/Pe pairs for Pe linearization. */

/* Transaction storage. */

unsigned char trrecl; /* PC 834.  Transaction record length. */
int trnumpgs; /* PC 835.  Transaction file start page (/ 100) and # pages (% 100). */
unsigned char barecl; /* PC 836.  Batch record length. */
int banumpgs; /* PC 837.  Batch file start page (/ 100) and # pages (% 100). */
unsigned char trtkfmt; /* PC 838.  Transaction ticket predefined format. */
unsigned char timouttr; /* PC 839.  Time out transaction. */

/* Transaction storage code configuration. */
struct Sccfg tsccfg[MAXTSC]; /* PC 840, etc.  TSC X offset and row/column. */

/* Batch storage code configuration. */
struct Sccfg bsccfg[MAXBSC];/* PC 880, etc.  TSC X offset and row/column. */

/* v5.30 */
/* Calibration for 4-20 mA and RTD inputs on CPU board. */
unsigned char rtdalpha; /* PC 983.  Alpha for all RTD's. */
int clcalib[4]; /* PC 984, etc.  CPU board 4-20 ma input LOLO, LOHI, HILO and
                ** HIHI counts.
                */
int rtdcalib[4]; /* PC 988, etc. CPU board RTD input LOLO, LOHI, HILO and
                 ** HIHI counts.
                 */

////
```

6.9 DanLoad 6000 v5.70 Database

```
////

/*
**
** Data Base
*/

struct {
    long passcode; /* PC 1, etc.  User X 9-digit passcode. */
    char userid[17]; /* PC 2, etc.  User X NUL terminated ID. */
    byte pwacess; /* PC 3, etc.  User X supervisor privilege. */
} pw[MAXUSER];

int nummtrs; /* PC 50.  Number of meters. */
struct {
    char mtrid[6]; /* PC 51, etc.  Meter X ID. */
    int mtrvlv; /* PC 52, etc.  Meter X flow control valve. */
    int radjfac; /* PC 729, etc.  Meter X ratio adjustment factor. */
    int fadjfac; /* PC 728, etc.  Meter X flow adjustment factor. */
    int ramppct; /* PC 730, etc.  Meter X ramp percentage. */
    int fpo; /* PC 756, etc.  Meter X MPMC factored pulse output. */
    int numblades; /* PC 757, etc.  Meter X CALMON number of blades. */
    long maxmaxchardev; /* PC 758, etc.  Meter X CALMON max. max. characteristic deviation. */
    long maxtotchardevs; /* PC 759, etc.  Meter X CALMON max. total characteristic deviations.
*/
    unsigned char calibsmplstat; /* PC 760, etc.  Meter X calibration sample status. */
    unsigned char calmonanalmthd; /* PC 731, etc.  Meter X CALMON analysis method. */
} mtrparms[MAXMTR];

byte opmode; /* PC 25.  Operating mode. */
byte language; /* PC 28.  Language. */
byte dtefmt; /* PC 38.  Date format. */
byte dtesep; /* PC 39.  Date separator. */
byte decsep; /* PC 40.  Decimal separator. */
byte produnits; /* PC 29.  Product units. */
byte numdataitems; /* PC 30.  Number of data items. */
char dataprompt[5][33]; /* PC 31, etc.  Data item X NUL terminated prompt. */
int pmt_timout; /* PC 36.  Prompt time-out. */
byte chk_disp; /* PC 37.  Check display. */
byte unit_type; /* PC 26.  Unit type. */
byte fcv_type; /* PC 27.  Valve type. */
unsigned char dspdr; /* PC 41.  Display data rate. */
int transeqnum; /* PC 42.  Transaction #. */
int batchseqnum; /* PC 43.  Batch #.  */
unsigned char roundmthd; /* PC 45.  Rounding method. */
int densscale; /* PC 46.  Density/gravity scale. */
int maxbatch; /* PC 47.  Maximum batches/transaction. */

int numvalves; /* PC 48.  Number of (flow control) valves. */
int initflwtime; /* PC 49.  Initial flow time. */

/* Component parameters */
int numcomps; /* PC 65.  Number of components. */
struct {
    char compid[17]; /* PC 66, etc.  Component X NUL terminated ID. */
    int compmtr; /* PC 67, etc.  Component X meter number. */
    long massadj; /* PC 68, etc.  Component X mass adjustment. */
} comp[MAXCOMPS];
```

```
/* Delivery parameters */
long maxpreset; /* PC 78.  Maximum preset volume. */
long minpreset; /* PC 79.  Minimum preset volume. */
byte predeltype; /* PC 80.  Preset/delivery type. */
byte deldisptype; /* PC 81.  Delivery display type. */
byte stopkey; /* PC 82.  Stop key action. */
long fallbackvol; /* PC 83.  Fall back volume. */

struct Compdel {
    long loflwstart; /* PC 84, etc.  Component X low flow start volume. */
    long loflwrestart; /* PC 85, etc.  Component X low flow restart volume. */
    long loflwstop; /* PC 86, etc.  Component X low flow stop volume. */
    int linepackdly; /* PC 87, etc.  Component X line pack delay. */
    int pumpstopdly; /* PC 88, etc.  Component X pump stop delay. */
    int blkvalvedly; /* PC 89, etc.  Component X block valve delay. */
} compdel[MAXCOMPS];
int maxrampclicks; /* PC 108.  Ramp clicks. */
int maxmaintclicks; /* PC 109.  Maintenance clicks. */
int addpmpdly; /* PC 110.  Additive pump stop delay. */
int primcmp; /* PC 111.  Primary component. */

/* Digital Valve Parameters */
struct {
    int dvlferrpct; /* PC 112, etc.  Valve X low flow % error. */
    int dvhferrpct; /* PC 113, etc.  Valve X high flow % error. */
    int dvdly; /* PC 114, etc.  Valve X maint click adjustment. */
    byte openmthd; /* PC 115, etc.  Valve X open method. */
} dvparms[MAXVLV];

/* Pulse per unit (DAU) parameters */
int numdaus; /* PC 128.  Number of PPU's. */
struct Dau {
    struct Dauctl {
        byte gross;
        byte net;
    } dauctl[MAXMTR]; /* PC 129, etc.  PPU X control meters. */
    int daufactor; /* PC 130, etc.  PPU X factor. */
    int daupulwidth; /* PC 131, etc.  PPU X pulse width. */
} dau[MAXDAU];

/* Additive Parameters */
int numadds; /* PC 135.  Number of additives. */
byte addselmthd; /* PC 136.  Additive selection method. */
int injpct; /* PC 137.  Additive inject % */
int addclnlinvol; /* PC 138.  Additive clean line volume. */
byte addunits; /* PC 44. Additive units. */
struct Add {
    struct Addctl {
        byte gross;
        byte net;
    } addctl[MAXMTR]; /* PC 139, etc.  Additive X control meters. */
    int ratiovol; /* PC 140, etc.  Additive X ratio volume in whole units. */
    unsigned char injmthd; /* PC 141, etc.  Additive X injection method. */
    long kfac; /* PC 142, etc.  Additive X volume per pulse or K-factor in
               ** hundredths.
               */
    long reqdvol; /* PC 143, etc.  Additive X volume (in external product
                  ** units x 10,000) per 1000 units of product.
                  */
} add[MAXADDS];

/* Factors  */
```

```
int numfacs; /* PC 169.  Number of factors/component. */
byte mfacmthd; /* PC 170.  Meter factor method. */
struct Fac {
    long nomkfac; /* PC 171, etc.  Component X nominal K-factor. */
    long mstrmfac; /* PC 172, etc.  Component X master meter factor. */
    struct Mtrfac {
        long flwrate; /* PC 174, etc.  Component X flow rate Y. */
        long mfac; /* PC 175.  Component X meter factor Y. */
    } mtrfac[MAXFACS];
} fac[MAXCOMPS];
int mstrmfacpct; /* PC 215.  Master meter factor percentage. */
int adjmfacpct; /* PC 216.  Adjacent meter factor percentage. */
char grsunitsmnemo[7]; /* PC 217.  Gross quantity units mnemonic. */
char stdunitsmnemo[7]; /* PC 218.  Standard quantity units mnemonic. */
unsigned char userestart; /* PC 219.  Use restart quantity. */

/* Alarm Parameters */
int secalrst; /* PC 220.  Secondary alarm reset (secs). */
byte lowflwact; /* PC 221.  Low flow alarm action. */
long lowflwlim; /* PC 222.  Minimum flow rate. */
int lowflwtime; /* PC 223.  Low flow time. */
byte hiflwact; /* PC 224.  High flow alarm action. */
long hiflwlim; /* PC 225.  Maximum flow rate. */
int hiflwtime; /* PC 226.  High flow time. */
int ovrunlim; /* PC 227.  Overrun limit volume. */
byte undrflwact; /* PC 228.  Underflow alarm action. */
int undrflwlim; /* PC 229.  Underflow limit volume. */
byte noflwact; /* PC 230.  No flow alarm action. */
int noflwtime; /* PC 231.  No flow time. */
int unauthflw; /* PC 232.  Unauth. flow limit volume. */
int pserr; /* PC 233.  Pulse security error limit. */
long psresetcnt; /* PC 234.  Pulse security reset count. */
byte dlfailact; /* PC 235.  Data logging alarm action. */
int commtimout[2]; /* PC 236's and 237.  Comms fail time-out per channel. */
byte tempfailact; /* PC 238.  Temperature fail alarm action. */
int tempmin; /* PC 239.  Min. temperature. */
int tempmax; /* PC 240.  Max. temperature. */
byte densfailact; /* PC 241.  Density fail alarm action. */
long densmin; /* PC 242.  Minimum density/gravity. */
long densmax; /* PC 243.  Maximum density/gravity. */
long presmin; /* PC 244.  Minimum pressure. */
long presmax; /* PC 245.  Maximum pressure. */
int adderrlim; /* PC 246.  Additive error limit. */
int addfbckcnt; /* PC 247.  # pulses per additive cycle or # seconds. */
int bvclsdly; /* PC 248.  Block valve time. */

struct Safety {
    byte action; /* PC 249, etc.  Safety circuit X alarm action. */
    char msg[33]; /* PC 250, etc.  Safety ciruit X alarm message. */
} safety[MAXSAFETY];

byte sctype[MAXSAFETY]; /* PC 265, etc.  Safety circuit X type. */

long unauthaddflw; /* PC 271.  Unauthorized additive flow volume. */
int praddminpct; /* PC 272.  Additive volume verification minimum error
                 ** percentage.
                 */
int praddmaxpct; /* PC 273.  Additive volume verification maximum error
                 ** percentage.
                 */

byte storeact; /* PC 274.  Storage alarm action. */
```

```
byte poweract; /* PC 275.  Power fail alarm action. */
byte rampact; /* PC 276.  Ramp down alarm action. */
int ramptime; /* PC 277.  Ramp down time. */
int add1000errlim; /* PC 278.  Additive per 1000 error limit. */
int endtim; /* PC 279.  End time. */

/* I/O Parameters */
unsigned char islot[9]; /* Slots J1 thru J7, DUART and ARCNET. */

struct Ioctl mtrin[MAXMTRIN];
struct Ioctl rtdin[MAXRTDIN];
struct Ioctl anain[MAXANAIN];
struct Ioctl digin[MAXDIGIN];
struct Ioctl digout[MAXDIGOUT];

struct Almout {
    int out; /* PC 287, etc.  Alarm output X. */
    unsigned long mask; /* PC 269, etc.  Alarm output X mask. */
} almout[MAXALMOUT];

int pmpctlout[MAXCOMPS]; /* PC 288, etc.  Component X pump output. */
struct Vlvio {
    int sol1; /* PC 292, etc.  Valve X solenoid 1. */
    int sol2; /* PC 293, etc.  Valve X solenoid 2. */
    int stemsw1; /* PC 294, etc.  Valve X stem switch 1. */
    int stemsw2; /* PC 295, etc.  Valve X stem switch 2. */
    int clsinp; /* PC 296, etc.  Valve X close input. */
} vlvio[MAXVLV];
byte sidemthd; /* PC 312.  Side detect method. */
int dauout[MAXDAU]; /* PC 313, etc.  PPU output X. */
int trip1out; /* PC 315.  Trip 1 output. */
int trip2out; /* PC 316.  Trip 2 output. */
int trip3out; /* PC 317.  Trip 3 output. */
struct Mtrtmp {
    int mtempinp; /* PC 318, etc.  Meter X temperature input. */
    long moffset; /* PC 319, etc.  Meter X offset (Ohms). */
    int mpresinp; /* PC 320, etc.  Meter X pressure input. */
    long pres4; /* PC 464, etc.  Meter X pressure at 4 mA. */
    long pres20; /* PC 465, etc.  Meter X pressure at 20 mA. */
    int mdensinp; /* PC 321, etc.  Meter X density input. */
    long dens4; /* PC 466, etc.  Meter X density at 4 mA. */
    long dens20; /* PC 467, etc.  Meter X density at 20 mA. */
    int flowinp; /* PC 322, etc.  Meter X flow input. */
} mtrtmp[MAXMTR];

int numcpmtszin; /* PC 341.  Number of compartment size inputs. */ /* v5.50 */
byte startmthd; /* PC 342.  Start batch method. */ /* v5.50 */
int autochng; /* PC 343.  Auto/manual change-over input. */
int primalrst; /* PC 344.  Primary alarm reset input. */

int safetyinp[MAXSAFETY]; /* PC 345, etc.  Safety circuit X input. */
byte temprange; /* PC 353.  Range for RTD inputs on analog inputs boards. */
int opdly; /* PC 354.  Delay after outputs, e.g. for DART 2. */

byte pridsptyp; /* PC 355.  Primary display type. */
int priminlt; /* PC 356.  Primary display min. light. */
int primaxlt; /* PC 357.  Primary display max. light. */
byte secdsptyp; /* PC 358.  Secondary display type. */
int secminlt; /* PC 359.  Secondary display min. light. */
int secmaxlt; /* PC 360.  Secondary display max. light. */

int recipeinp[MAXRECIN]; /* PC 361, etc.  Recipe X selection input. */
```

```
int recipeout[MAXRECOUT]; /* PC 367, etc.  Recipe X selection output. */

unsigned char endmthd; /* PC 373.  End output method. */ /* v5.60 */

int levelinp; /* PC 374.  Intermediate level input. */ /* v5.50 */
int hatchinp; /* PC 375.  Hatch sensor input. */ /* v5.50 */
int swing2; /* PC 376.  Swing-arm switch 2. */
int swing1; /* PC 377.  Swing-arm switch 1. */
int loflwinp; /* PC 378.  Stay in low flow input. */

unsigned char fpobasis; /* PC 752.  Factored pulse outputs basis. */
int calmonerr; /* PC 753.  Calibration monitoring error limit. */
int calmonrcnt; /* PC 754.  Calibration monitoring reset count. */
byte calibfailact; /* PC 755.  Calibration failure alarm action for CALMON. */

struct Addinp {
       int ratio; /* PC 380, etc.  Additive X ratio output. */
       int feedback; /* PC 381, etc.  Additive X feedback input. */
       int select; /* PC 382, etc.  Additive X selection input. */
    int pmpctlout; /* PC 800, etc.  Additive X pump control output. */
    int bvout; /* PC 801, etc.  Additive X block valve output. */
} addinp[MAXADDS];

int addclnlinout; /* PC 398.  Additive clean line output. */
int addmtrflushout; /* PC 399.  Additive meter flush output. */

struct Cmpio {
       int bvout; /* PC 400, etc.  Component X block valve output. */
       int bvin; /* PC 401, etc.  Component X block valve input. */
} cmpio[MAXCOMPS];
int combio[15]; /* PC 408, etc.  Component combination X. */
int endout; /* PC 423.  End output. */
int endinp; /* PC 424.  End input. */
unsigned long addmtrflushpulses; /* PC 425.  Additive meter flush pulses. */

long buoyancy; /* PC 426.  Density adjustment for buoyancy in air. */
byte tempunits; /* PC 427.  Temperature units. */
byte densunits; /* PC 428.  Density units. */
byte presunits; /* PC 429.  Pressure units. */
int reftemp; /* PC 430.  Reference temperature. */
long tempsmpl; /* PC 431.  Sample volume. */

struct Cmptemp {
    byte option; /* PC 432, etc.  Component X temperature correction option. */
    long alpha; /* PC 433, etc.  Component X alpha. */
    int backup; /* PC 434, etc.  Component X backup temperature. */
} cmptemp[MAXCOMPS];
struct Cmppres {
    byte option; /* PC 444, etc.  Component X pressure correction option. */
    long pffac; /* PC 445, etc.  Component X F-factor. */
    long backup; /* PC 446, etc.  Component X backup pressure. */
} cmppres[MAXCOMPS];
struct Cmpdens {
    byte option; /* PC 456, etc.  Component X density option. */
    long backup; /* PC 457, etc.  Component X backup density/gravity. */
} cmpdens[MAXCOMPS];

int numrecipes; /* PC 480.  Number of recipes. */
struct Recipe {
    char name[17]; /* PC 481, etc.  Recipe X NUL terminated name. */
    int cmppct[MAXCOMPS]; /* PC 482, etc.  Recipe X component percentages. */
    struct Cmpseq {
```

```
        int cmpno;
    } cmpseq[MAXCOMPS]; /* PC 486, etc.  Recipe X sequence. */
} recipe[MAXRECIPES];

int numaddcalibcycles; /* PC 662.  # additive calibration injection cycles. */

/* v5.30 */
int commsaddr; /* PC 662.  Communications address. */

struct Commun {
    byte mode; /* PC 663, etc.  Channel X mode. */
    byte baud; /* PC 664, etc.  Channel X data rate. */
    byte dbits; /* PC 665, etc.  Channel X word size. */
    byte sbits; /* PC 666, etc.  Channel X stop bits. */
    byte parity; /* PC 667, etc.  Channel X parity. */
} commun[2];

unsigned char commalarmmanb; /* PC 673.  Channel B alarm in manual. */

unsigned char ddmode; /* PC 679.  Dynamic data display mode. */
int ddpc[16]; /* PC 680, etc.  Dynamic data display element X data code. */

byte batch; /* PC 696.  Batch summary. */
byte trans; /* PC 697.  Transaction summary. */
byte almlog; /* PC 698.  Alarm log. */
byte pflog; /* PC 699.  Power fail log. */
byte progmode; /* PC 700.  Program mode entry/exit. */
byte wmsw; /* PC 701.  W&M switch opened/closed. */
byte pcchnglog; /* PC 702.  Program value change. */
byte cfglog; /* PC 703.  Configuration summary data log. */
byte crash; /* PC 704.  Crash memory summary. */
byte totlog; /* PC 705.  Totalizers. */
byte trtkauto; /* PC 706.  Transaction ticket. */
byte trtk; /* PC 708.  Transaction ticket reprint. */
byte thrulog; /* PC 708.  Thruput. */
byte dlogseq; /* PC 709.  Sequence numbers. */
int cuthour; /* PC 710.  Cutoff hour. */

/* Blending. */
int rampdownpct; /* PC 711.  The percentage of the blend's current flow rate, etc. */
int deltarate; /* PC 712.  Change in flow rate for batch fall back. */
long loflwstart; /* PC 713.  Low flow start volume for in-line blend. */
int clnlinvol; /* PC 714.  Clean line volume. */
int lopropfac; /* PC 715.  The meter factor flow rate for low proportion
                ** components of an in-line blend.
                */
int blchkvol; /* PC 716.  Correct after volume.  Volume after which blend
               ** ratio adjustment begins.
               */
int blchkvol2; /* PC 717.  Alarm after volume.  Volume after which blend
                ** ratio alarm is checked.
                */
int vadjust; /* PC 718.  Volume over which blend adjustment is made. */
unsigned char comppctmthd; /* PC 719.  Component % display. */
int addcalibdly; /* PC 720.  Additive calibration delay. */
unsigned char blenderrmthd; /* PC 722.  Blend error method. */
int blnddev; /* PC 723.  Max. dev. % */
int blndtol; /* PC 724.  Blend tol. % */
int blnddevvol; /* PC 725.  Max. dev. volume. */
int blndtolvol; /* PC 726.  Blend tol. volume. */
int blendsmpl; /* PC 727.  Blend sample volume. */
long stoprate[MAXCOMPS]; /* PC 173.  Component X stop rate for in-line blend. */
```

```
long rflwrate[MAXRFLWRATE][2]; /* PC 776, 777, etc.  Recipe "group" low and
                               ** high flow rates.
                               */

unsigned char pttype; /* PC 818.  Pressure transmitter types. */
unsigned char pemthd; /* PC 819.  Pe method. */
unsigned char pinchmthd; /* PC 820.  Pinch back method. */
long pinchpres; /* PC 821.  Pinch back pressure. */
long atmospres; /* PC 822.  Atmospheric pressure. */
long pinchqty; /* PC 823.  Pinch back qty. */
struct Pe_lin {
    int temp; /* PC 824, etc.  Tx. */
    long pres; /* PC 825, etc.  Pe at Tx. */
} pe_lin[MAXPELIN]; /* T/Pe pairs for Pe linearization. */

/* Transaction storage. */

unsigned char trrecl; /* PC 834.  Transaction record length. */
int trnumpgs; /* PC 835.  Transaction file start page (/ 100) and # pages (% 100). */
unsigned char barecl; /* PC 836.  Batch record length. */
int banumpgs; /* PC 837.  Batch file start page (/ 100) and # pages (% 100). */
unsigned char trtkfmt; /* PC 838.  Transaction ticket predefined format. */
unsigned char timouttr; /* PC 839.  Time out transaction. */

/* Transaction storage code configuration. */
struct Sccfg tsccfg[MAXTSC]; /* PC 840, etc.  TSC X offset and row/column. */

/* Batch storage code configuration. */
struct Sccfg bsccfg[MAXBSC];/* PC 880, etc.  TSC X offset and row/column. */

/* v5.30 */
/* Calibration for 4-20 mA and RTD inputs on CPU board. */
unsigned char rtdalpha; /* PC 983.  Alpha for all RTD's. */
int clcalib[4]; /* PC 984, etc.  CPU board 4-20 ma input LOLO, LOHI, HILO and
                ** HIHI counts.
                */
int rtdcalib[4]; /* PC 988, etc. CPU board RTD input LOLO, LOHI, HILO and
                 ** HIHI counts.
                 */

////
```

6.10 Script DanLoad 6000 v1.20 Database

The "database" is the name given to the stored program code values. The database for Script DanLoad 6000 v1.20 is shown below. (A "byte" is an unsigned char.)

```
////

/*
*
*   Data Base
*/

struct {
    long passcode; /* PC 1, etc.  User X 9-digit passcode. */
    char userid[17]; /* PC 2, etc.  User X NUL terminated ID. */
    byte pwacess; /* PC 3, etc.  User X supervisor privilege. */
} pw[MAXUSER];

int  nummtrs; /* PC 50.  Number of meters. */
struct {
    char mtrid[6]; /* PC 51, etc.  Meter X ID. */
    int  mpulsin; /* PC 52, etc.  Meter pulse board input. */
    int  mtrvlv; /* PC 53, etc.  Meter X flow control valve. */
    int  radjfac; /* PC 729, etc.  Meter X ratio adjustment factor. */
    int  fadjfac; /* PC 728, etc.  Meter X flow adjustment factor. */
} mtrparms[MAXMTR];

byte opmode; /* PC 25.  Operating mode. */
byte language; /* PC 28.  Language. */
byte dtefmt; /* PC 38.  Date format. */
byte dtesep; /* PC 39.  Date separator. */
byte decsep; /* PC 40.  Decimal separator. */
byte units; /* PC 29.  Product units. */
byte stndby_prompts; /* PC 30.  Number of data items. */
char dataprompt[5][33]; /* PC 31, etc.  Data item X NUL terminated prompt. */
int  pmt_timout; /* PC 36.  Prompt time-out. */
byte chk_disp; /* PC 37.  Check display. */
byte unit_type; /* PC 26.  Unit type. */
byte fcv_type; /* PC 27.  Valve type. */
unsigned char dspdr; /* PC 41.  Display data rate. */
int  transeqnum; /* PC 42.  Transaction #. */
int  batchseqnum; /* PC 43.  Batch #.  */
unsigned char roundmthd; /* PC 45.  Rounding method. */
int densscale; /* PC 46.  Density/gravity scale. */

int  numvalves; /* PC 48.  Number of (flow control) valves. */
int initflwtime; /* PC 49.  Initial flow time. */

/* Component parameters */
int numcomps; /* PC 65.  Number of components. */
struct {
    char compid[17]; /* PC 66, etc.  Component X NUL terminated ID. */
    int compmtr; /* PC 67, etc.  Component X meter number. */
    long massadj; /* PC 68, Component X mass adjustment factor. */
} comp[MAXCOMPS];

/* Delivery parameters */
long maxpreset; /* PC 78.  Maximum preset quantity. */
long minpreset; /* PC 79.  Minimum preset quantity. */
```

```
byte predeltype; /* PC 80.  Preset/delivery type. */
byte deldisptype; /* PC 81.  Delivery display type. */
byte stopkey; /* PC 82.  Stop key action. */
long fallbackvol; /* PC 83.  Fall back quantity. */

struct Compdel {
    long loflwstart; /* PC 84, etc.  Component X low flow start quantity.*/
    long loflwrestart; /* PC 85, etc.  Component X low flow restart quantity. */
    long loflwstop; /* PC 86, etc.  Component X low flow stop quantity. */
    int  linepackdly; /* PC 87, etc.  Component X line pack delay. */
    int  pumpstopdly; /* PC 88, etc.  Component X pump stop delay. */
    int  blkvalvedly; /* PC 89, etc.  Component X block valve delay. */
} compdel[MAXCOMPS];
int maxrampclicks; /* PC 108.  Ramp clicks. */
int maxmaintclicks; /* PC 109.  Maintenance clicks. */
int addpmpdly; /* PC 110.  Additive pump stop delay. */
int primcmp; /* PC 111.  Primary component. */

/* Digital Valve Parameters */
struct {
    int dvlferrpct; /* PC 112, etc.  Valve X low flow % error. */
    int dvhferrpct; /* PC 113, etc.  Valve X high flow % error. */
    int dvdly; /* PC 114, etc.  Valve X maint click adjustment. */
    byte openmthd; /* PC 115, etc.  Valve X open method. */
} dvparms[MAXVLV];

/* Pulse per unit parameters */
int numdaus; /* PC 128.  Number of PPU's. */
struct Dau {
    struct Dauctl {
        byte gross;
        byte net;
    } dauctl[MAXMTR]; /* PC 129, etc.  PPU X control meters. */
    int daufactor; /* PC 130, etc.  PPU X factor. */
    int daupulwidth; /* PC 131, etc.  PPU X pulse width. */
} dau[MAXDAU];

/* Additive Parameters */
int numadds; /* PC 135.  Number of additives. */
byte addprompt; /* PC 136.  Additive selection method. */
byte addtotmthd; /* PC 137.  Additive totalizing method */
int injlowflw; /* PC 138.  Additive clen line quantity */
byte addunits; /* PC 44.   Additive units. */
struct Add {
    struct Addctl {
        byte gross;
        byte net;
    } addctl[MAXMTR]; /* PC 139, etc. Additive X control meters. */
    int ratiovol; /* PC 140, etc. Additive X ratio qty in whole units.*/
    int offsetvol; /* PC 141, etc. Additive X offset quantity. */
    long totvol; /* PC 142, etc, Additive X totalizing volume */
    byte addavail; /* PC 143, etc, Additive X available */
} add[MAXADDS];

/* Factors  */
int numfacs; /* PC 169.  Number of factors/component. */
byte mfacmthd; /* PC 170.  Meter factor method. */
struct Fac {
    int nomkfac; /* PC 171, etc.  Component X nominal K-factor. */
    long mstrmfac; /* PC 172, etc.  Component X master meter factor. */
    /* byte tempcomp; */
    struct Mtrfac {
```

```
        long flwrate; /* PC 174, etc.  Component X flow rate Y. */
        long mfac; /* PC 175.  Component X meter factor Y. */
    } mtrfac[MAXFACS];
} fac[MAXCOMPS];
int mstrmfacpct; /* PC 215.  Master meter factor percentage. */
int adjmfacpct; /* PC 216.  Adjacent meter factor percentage. */
unsigned char pwpertr; /* PC 217. Password (passcode) per transaction */
char produnitsmnemo[7]; /* PC 218 Product units mnemonic */

/* Alarm Parameters */
unsigned char safetymsg; /* PC 219 Type of safety message */
int secalrst; /* PC 220.  Secondary alarm reset (secs). */
byte lowflwact; /* PC 221.  Low flow alarm action. */
long lowflwlim; /* PC 222.  Minimum flow rate. */
int lowflwtime; /* PC 223.  Low flow time. */
byte hiflwact; /* PC 224.  High flow alarm action. */
long hiflwlim; /* PC 225.  Maximum flow rate. */
int hiflwtime; /* PC 226.  High flow time. */
int ovrunlim; /* PC 227.  Overrun limit volume. *
byte undrflwact; /* PC 228.  Underflow alarm action. */
int undrflwlim; /* PC 229.  Underflow limit volume. */
byte noflwact; /* PC 230.  No flow alarm action. */
int noflwtime; /* PC 231.  No flow time. */
int unauthflwlim; /* PC 232.  Unauth. flow limit volume. *
int pserr; /* PC 233.  Pulse security error limit. */
long psresetcnt; /* PC 234.  Pulse security reset count. */
byte dlfailact; /* PC 235.  Data logging alarm action. */
byte commfailact; /* PC 236.  Comms fail alarm action. */
int commtimout; /* PC 237.  Comms fail time-out. */
byte tempfailact; /* PC 238.  Temperature fail alarm action. */
int tempmin; /* PC 239.  Min. temperature. */
int tempmax; /* PC 240.  Max. temperature. */
byte densfailact; /* PC 241.  Density fail alarm action. */
long densmin; /* PC 242.  Minimum density/gravity. */
long densmax; /* PC 243.  Maximum density/gravity. */
long presmin; /* PC 244.  Minimum pressure. */
long presmax; /* PC 245.  Maximum pressure. */
int adderrlim; /* PC 246.  Additive error limit. */
int feedbacktime; /* PC 247   Feedback time */
int bvclsdly; /* PC 248.  Block valve time. */

struct Safety {
    byte action; /* PC 249, etc.  Safety circuit X alarm action. */
    char msg[33]; /* PC 250, etc.  Safety ciruit X alarm message. */
} safety[MAXSAFETY];

byte sc5type; /* PC 265.  Safety circuit 5 type. */
byte sc6type; /* PC 266.  Safety circuit 6 type. */
byte sc7type; /* PC 267.  Safety circuit 7 type. */
byte sc8type; /* PC 268.  Safety circuit 8 type. */

/* I/O Parameters */
unsigned char islot[9]; /* Slots J1 thru J7, DUART and ARCNET. */

struct Ioctl mtrin[MAXMTRIN];
struct Ioctl rtdin[MAXRTDIN];
struct Ioctl anain[MAXANAIN];
struct Ioctl digin[MAXDIGIN];
struct Ioctl digout[MAXDIGOUT];

int almctlout; /* PC 287, etc.  Alarm output X. */
```

```
int pmpctlout[MAXCOMPS]; /* PC 288, etc.  Component X pump output. */
struct Vlvio {
    int sol1; /* PC 292, etc.  Valve X solenoid 1. */
    int sol2; /* PC 293, etc.  Valve X solenoid 2. */
    int stemsw1; /* PC 294, etc.  Valve X stem switch 1. */
    int stemsw2; /* PC 295, etc.  Valve X stem switch 2. */
    int clsinp; /* PC 296, etc.  Valve X close input. */
} vlvio[MAXVLV];
byte sidemthd; /* PC 312.  Side detect method. */
int dauout[MAXDAU]; /* PC 313, etc.  PPU output X. */
int trip1out; /* PC 315.  Trip 1 output. */
int trip2out; /* PC 316.  Trip 2 output. */
int trip3out; /* PC 317.  Trip 3 output. */
struct Mtrtmp {
    int mtempinp; /* PC 318, etc.  Meter X temperature input. */
    long moffset; /* PC 319, etc.  Meter X offset (Ohms). */
    int mpresinp; /* PC 320, etc.  Meter X pressure input. */
    long pres4; /* PC 464, etc.  Meter X pressure at 4 mA. */
    long pres20; /* PC 465, etc.  Meter X pressure at 20 mA. */
    int mdensinp; /* PC 321, etc.  Meter X density input. */
    long dens4; /* PC 466, etc.  Meter X density at 4 mA. */
    long dens20; /* PC 467, etc.  Meter X density at 20 mA. */
} mtrtmp[MAXMTR];

int startend; /* PC 342.  Off-rack start/end input. */
int autochng; /* PC 343.  Auto/manual change-over input. */
int primalrst; /* PC 344.  Primary alarm reset input. */

int safetyinp[MAXSAFETY]; /* PC 345, etc.  Safety circuit X input. */
byte temprange; /* PC 353.  RTD inputs type. */

byte pridsptyp; /* PC 355.  Primary display type. */
int priminlt; /* PC 356.  Primary display min. light. */
int primaxlt; /* PC 357.  Primary display max. light. */
byte secdsptyp; /* PC 358.  Secondary display type. */
int secminlt; /* PC 359.  Secondary display min. light. */
int secmaxlt; /* PC 360.  Secondary display max. light. */

int recipeinp[MAXRECIN]; /* PC 361, etc.  Recipe X selection input. */
int recipeout[MAXRECOUT]; /* PC 367, etc.  Recipe X selection output. */

struct Addinp {
       int ratio; /* PC 380, etc.  Additive X ratio output. */
       int feedback; /* PC 381, etc.  Additive X feedback input. */
       int select; /* PC 382, etc.  Additive X selection input. */
    int pmpctlout; /* PC 800, etc.  Additive X pump control output. */
    int bvout; /* PC 801, etc.  Additive X block valve output. */
} addinp[MAXADDS];

struct Cmpio {
       int bvout; /* PC 400, etc.  Component X block valve output. */
       int bvin; /* PC 401, etc.  Component X block valve input. */
} cmpio[MAXCOMPS];
int combio[15]; /* PC 408, etc.  Component combination X. */
long buoyancy; /* PC 426   buoyancy */
byte tempunits; /* PC 427.  Temperature units. */
byte densunits; /* PC 428.  Density units. */
byte presunits; /* PC 429.  Pressure units. */
int reftemp; /* PC 430.  Reference temperature. */
long tempsmpl; /* PC 431.  Sample volume. */

struct Cmptemp {
```

```
       byte option; /* PC 432, etc. Component X temp correction option.*/
       long alpha; /* PC 433, etc. Component X alpha. */
       int backup; /* PC 434, etc. Component X backup temperature. */
} cmptemp[MAXCOMPS];
struct Cmppres {
       byte option; /* PC 444, etc.  Component X pressure correction */
       long pffac; /* PC 445, etc.  Component X F-factor. */
       long vp; /* PC 446, etc.  Component X vapor pressure. */
} cmppres[MAXCOMPS];
struct Cmpdens {
       byte option; /* PC 456, etc.  Component X density option. */
       long backup; /* PC 457, etc.  Component X backup density/gravity.*/
} cmpdens[MAXCOMPS];

int numrecipes; /* PC 480.  Number of recipes. */
struct Recipe {
       char name[17]; /* PC 481, etc.  Recipe X NUL terminated name. */
       int cmppct[MAXCOMPS]; /* PC 482, etc.  Recipe X component percentages. */
    struct Cmpseq {
        int cmpno;
    } cmpseq[MAXCOMPS]; /* PC 486, etc.  Recipe X sequence. */
} recipe[MAXRECIPES];

struct Commun {
       byte mode; /* PC 663, etc.  Channel X mode. */
       byte baud; /* PC 664, etc.  Channel X data rate. */
       byte dbits; /* PC 665, etc.  Channel X word size. */
       byte sbits; /* PC 666, etc.  Channel X stop bits. */
       byte parity; /* PC 667, etc.  Channel X parity. */
} commun[2];

unsigned char ddenabled; /* PC 679  Dynamic data display enabled */
int ddpc[16]; /* PC 680, etc. Dynamic data display element X DC. */

byte batch; /* PC 696.  Batch summary. */
byte trans; /* PC 697.  Transaction summary. */
byte almlog; /* PC 698.  Alarm log. */
byte pflog; /* PC 699.  Power fail log. */
byte progmode; /* PC 700.  Program mode entry/exit. */
byte wmsw; /* PC 701.  W&M switch opened/closed. */
byte pcchnglog; /* PC 702.  Program value change. */
byte provelog; /* PC 703.  Meter proving run log. */
byte crash; /* PC 704.  Crash memory summary. */

/* Blending. */
int deltarate; /* PC 712. Change in flow rate for batch fall back.*/
long loflwstart; /* PC 713. Low flow start volume for in-line blend.*/
int clnlinvol; /* PC 714. Clean line volume. */
int lopropfac; /* PC 715. The meter factor flow rate for low
               ** proportion components of an in-line blend.
               */
int blchkvol; /* PC 716. Correct after volume.  Volume after which
              ** blend ratio adjustment begins.
              */
int blchkvol2; /* PC 717.  Alarm after volume.  Volume after which
               ** blend ratio alarm is checked.
               */

int vadjust; /* PC 718.  Volume over which blend adjustment
             ** is made.
             */
```

```
unsigned char comppctmthd; /* PC 719.  Component % display. */
unsigned char blenderrmthd; /* PC 722.  Blend error method. */
int blnddev; /* PC 723.  Max. dev. % */
int blndtol; /* PC 724.  Blend tol. % */
int blnddevvol; /* PC 726.  Blend tol. volume. */
int blndtolvol; /* PC 727.  Blend sample volume. */
int blendsmpl; /* PC 173.  Component X stop rate for in-line blend.*/
long stoprate[MAXCOMPS]; /* PC 173. Component X stop rate for in-line
                         ** blend.
                         */

long rflwrate[MAXRFLWRATE][2]; /* PC 776, 777, etc.  Recipe "group" low and
                               ** high flow rates.
                               */

////
```

6.11 CRC-16 Calculation

Note

This CRC-16 calculation shown is for an Intel-type microprocessor that stores the least significant byte of a 16-bit word at the low address. Some modification may be required for other microprocessors.

```
/*****************************************************************************
**
** Filename     : crc16.c
**
**                Copyright (c) 1992 by Daniel Flow Products
**                                      9753 Pine Lake Drive
**                                      P.O. Box 55435
**                                      Houston
**                                      Texas 77255
**
**                                      Tel. 713-467 6000
**                                      Fax. 713-827 3887
**
** Modified...
** Date         : 6 December 1992
** By           : Daniel Flow Products
** Version      : 1.00
** Description  : Original.
**
**                CRC-16.
**
**                       16    15    2
**                      X   + X   + X  + 1
**
*****************************************************************************/
#include <crc16.h>

/* Combining-value lookup table for classic CRC-16 polynomial.
*/

const unsigned short crc16tab[256] = {

0x0000, 0xc1c0, 0x81c1, 0x4001, 0x01c3, 0xc003, 0x8002, 0x41c2,
0x01c6, 0xc006, 0x8007, 0x41c7, 0x0005, 0xc1c5, 0x81c4, 0x4004,
0x01cc, 0xc00c, 0x800d, 0x41cd, 0x000f, 0xc1cf, 0x81ce, 0x400e,
0x000a, 0xc1ca, 0x81cb, 0x400b, 0x01c9, 0xc009, 0x8008, 0x41c8,
0x01d8, 0xc018, 0x8019, 0x41d9, 0x001b, 0xc1db, 0x81da, 0x401a,
0x001e, 0xc1de, 0x81df, 0x401f, 0x01dd, 0xc01d, 0x801c, 0x41dc,
0x0014, 0xc1d4, 0x81d5, 0x4015, 0x01d7, 0xc017, 0x8016, 0x41d6,
0x01d2, 0xc012, 0x8013, 0x41d3, 0x0011, 0xc1d1, 0x81d0, 0x4010,
0x01f0, 0xc030, 0x8031, 0x41f1, 0x0033, 0xc1f3, 0x81f2, 0x4032,
0x0036, 0xc1f6, 0x81f7, 0x4037, 0x01f5, 0xc035, 0x8034, 0x41f4,
0x003c, 0xc1fc, 0x81fd, 0x403d, 0x01ff, 0xc03f, 0x803e, 0x41fe,
0x01fa, 0xc03a, 0x803b, 0x41fb, 0x0039, 0xc1f9, 0x81f8, 0x4038,
0x0028, 0xc1e8, 0x81e9, 0x4029, 0x01eb, 0xc02b, 0x802a, 0x41ea,
0x01ee, 0xc02e, 0x802f, 0x41ef, 0x002d, 0xc1ed, 0x81ec, 0x402c,
0x01e4, 0xc024, 0x8025, 0x41e5, 0x0027, 0xc1e7, 0x81e6, 0x4026,
0x0022, 0xc1e2, 0x81e3, 0x4023, 0x01e1, 0xc021, 0x8020, 0x41e0,
```

```
0x01a0, 0xc060, 0x8061, 0x41a1, 0x0063, 0xc1a3, 0x81a2, 0x4062,
0x0066, 0xc1a6, 0x81a7, 0x4067, 0x01a5, 0xc065, 0x8064, 0x41a4,
0x006c, 0xc1ac, 0x81ad, 0x406d, 0x01af, 0xc06f, 0x806e, 0x41ae,
0x01aa, 0xc06a, 0x806b, 0x41ab, 0x0069, 0xc1a9, 0x81a8, 0x4068,
0x0078, 0xc1b8, 0x81b9, 0x4079, 0x01bb, 0xc07b, 0x807a, 0x41ba,
0x01be, 0xc07e, 0x807f, 0x41bf, 0x007d, 0xc1bd, 0x81bc, 0x407c,
0x01b4, 0xc074, 0x8075, 0x41b5, 0x0077, 0xc1b7, 0x81b6, 0x4076,
0x0072, 0xc1b2, 0x81b3, 0x4073, 0x01b1, 0xc071, 0x8070, 0x41b0,
0x0050, 0xc190, 0x8191, 0x4051, 0x0193, 0xc053, 0x8052, 0x4192,
0x0196, 0xc056, 0x8057, 0x4197, 0x0055, 0xc195, 0x8194, 0x4054,
0x019c, 0xc05c, 0x805d, 0x419d, 0x005f, 0xc19f, 0x819e, 0x405e,
0x005a, 0xc19a, 0x819b, 0x405b, 0x0199, 0xc059, 0x8058, 0x4198,
0x0188, 0xc048, 0x8049, 0x4189, 0x004b, 0xc18b, 0x818a, 0x404a,
0x004e, 0xc18e, 0x818f, 0x404f, 0x018d, 0xc04d, 0x804c, 0x418c,
0x0044, 0xc184, 0x8185, 0x4045, 0x0187, 0xc047, 0x8046, 0x4186,
0x0182, 0xc042, 0x8043, 0x4183, 0x0041, 0xc181, 0x8180, 0x4040
};
```

```
/*****************************************************************************
**
** Filename     : crc16.h
**
**                Copyright (c) 1992 by Daniel Flow Products
**                                   9753 Pine Lake Drive
**                                   P.O. Box 55435
**                                   Houston
**                                   Texas 77255
**
**                                   Tel. 713-467 6000
**                                   Fax. 713-827 3887
** Modified...
** Date         : 29 April 1994
** By           : Daniel Flow Products, Inc
**                9753 Pine Lake Drive
**                Houston, Texas 77255
** Version      : 3.00
** Description  : Compute CRC-16.
**
**                 16    15    2
**                X   + X   + X  + 1
**
**                C language code example:
**
**                #include <crc16.h>
**
**                extern unsigned char buff[256];
**
**                void cmptcrc(int numchars)
**                {
**                    unsigned short acc;
**                    unsigned char crc_byte1, crc_byte2;
**                    int i;
**
**                    /* Using an array of unsigned chars for the message
**                    ** buffer avoids "sign extension" assignment problems.
**                    */
**                    acc = 0xffff; /* Flush with ones. */
**                    for (i = 0; i < numchars; i++) {
**                        crc16update(buff[i], &acc);
**                    }
**                    crc_byte1 = acc >> 8; /* Typically assigned to
**                                          ** buff[numchars] before
**                                          ** transmitting.
**                                          */
**                    crc_byte2 = acc & 0xff; /* Typically assigned to
**                                            ** buff[numchars + 1] before
**                                            ** transmitting.
**                                            */
**                } /* cmptcrc() */
**
**                For greater efficiency, CRC-16 can be computed
**                character-by-character as a message is transmitted or
**                received.
**
*****************************************************************************/
void crc16update(unsigned short, unsigned short *);
/* Macro to calculate classic CRC-16 on data using lookup table. */
#define crc16update(d, a) *(a) = ((*(a) << 8) ^ crc16tab[(*(a) >> 8) ^ (d)])
extern const unsigned short crc16tab[256];
```

6.12 CRC-16 Computation Method and Example

The following method and example may be useful to someone programming an interface to the DanLoad 6000 who is not familiar with the C language.

Notation

SHL = shift left bits
SHR = shift right bits
XOR = bit-wise exclusive or

All numbers are hexadecimal (base 16) except the "16" in "16-bit" and the row and column numbers indicated in step (5) of the example which are decimal (base 10).

Method

For each data byte perform the following steps:

1) Identify the 16-bit unsigned accumulator (a) and 8-bit unsigned data byte (d). For the first data byte the accumulator is FFFF. For the second, third, fourth, etc. data bytes the accumulator is the result from the previous step (6).

2) Compute w = a SHL 8.

3) Compute x = a SHR 8.

4) Compute y = x XOR d. (y is always greater than or equal to zero and less than or equal to 255.)

5) Compute z = crc16tab[y] via table look-up.

6) Compute w XOR z. If this iteration is for the last data byte the result is the CRC-16 checksum; the CRC-16 checksum should be transmitted high byte first, low byte second.

Example Compute the CRC-16 checksum for the sequence of data bytes:

01 41 02 21

i.e. Start Communications command with function code 41 to DanLoad 6000 with communications address 01.

1) a = FFFF for first data byte and d = 01 (first data byte).
2) FFFF SHL 8 = FF00
3) FFFF SHR 8 = 00FF
4) 00FF XOR 01 = 00FE
5) crc16tab[00FE] = 8180 (row 32, column 7 in table)
6) FF00 XOR 8180 = 7E80

1) a = 7E80 from previous step (6) and d = 41 (second data byte).
2) 7E80 SHL 8 = 8000
3) 7E80 SHR 8 = 007E
4) 007E XOR 41 = 003F
5) crc16tab[003F] = 4010 (row 8, column 8 in table)
6) 8000 XOR 4010 = C010

1) a = C010 from previous step (6) and d = 02 (third data byte).
2) C010 SHL 8 = 1000
3) C010 SHR 8 = 00C0
4) 00C0 XOR 02 = 00C2
5) crc16tab[00C2] = 8191 (row 25, column 3 in table)
6) 1000 XOR 8191 = 9191

1) a = 9191 from previous step (6) and d = 21 (fourth and last data byte).
2) 9191 SHL 8 = 9100
3) 9191 SHR 8 = 0091
4) 0091 XOR 21 = 00B0
5) crc16tab[00B0] = 01B4 (row 23, column 0 in table)
6) 9100 XOR 01B4 = 90B4. This is the last data byte. The CRC-16 checksum should be transmitted as 90 B4, i.e. the complete message with CRC-16 checksum is:

01 41 02 21 90 B4

6.13 Example Program

The following is an example program (in C) that illustrates automation system/DanLoad 6000 communications.

```
////
/* "example.c" */

#include <stdio.h>

#define TRUE 1
#define FALSE 0

/* Max. # bytes in a Modbus RTU message. */
#define MAXMBMSGSZ 256

/* Max. # DanLoad 6000's supported by this program. */
#define MAXDL6000 16

/* Macro to calculate classic CRC-16 on data using lookup table. */
#define crc16update(d, a) *(a) = ((*(a) << 8) ^ crc16tab[(*(a) >> 8) ^ (d)])

/*************************************************************************/

struct Dl6000 {
    unsigned char *omsg; /* Message buffer for sending. */
    unsigned char *imsg; /* Message buffer for receiving. */
    unsigned char addr; /* DanLoad 6000 address. */
    unsigned char fc; /* Modbus RTU function code. */
    unsigned char online; /* Online/offline flag. */
    /*** DanLoad 6000 configuration. ***/
    int nummtrs;
    int numcomps;
    int numvalves;
    int numfacs;
    int numrecipes;
    int numadds;
    unsigned char tempunits;
};

/*************************************************************************/

FILE *port; /* FILE should be typedef in <stdio.c>. */
int baud_rate;
unsigned char mbmsgin[MAXMBMSGSZ];
unsigned char mbmsgout[MAXMBMSGSZ];

struct Dl6000 dl6000[MAXDL6000];

/* Combining-value lookup table for classic CRC-16 polynomial. */
const unsigned short crc16tab[256] = {
    0x0000, 0xc1c0, 0x81c1, 0x4001, 0x01c3, 0xc003, 0x8002, 0x41c2,
    0x01c6, 0xc006, 0x8007, 0x41c7, 0x0005, 0xc1c5, 0x81c4, 0x4004,
    0x01cc, 0xc00c, 0x800d, 0x41cd, 0x000f, 0xc1cf, 0x81ce, 0x400e,
    0x000a, 0xc1ca, 0x81cb, 0x400b, 0x01c9, 0xc009, 0x8008, 0x41c8,
    0x01d8, 0xc018, 0x8019, 0x41d9, 0x001b, 0xc1db, 0x81da, 0x401a,
    0x001e, 0xc1de, 0x81df, 0x401f, 0x01dd, 0xc01d, 0x801c, 0x41dc,
    0x0014, 0xc1d4, 0x81d5, 0x4015, 0x01d7, 0xc017, 0x8016, 0x41d6,
    0x01d2, 0xc012, 0x8013, 0x41d3, 0x0011, 0xc1d1, 0x81d0, 0x4010,
```

```
    0x01f0, 0xc030, 0x8031, 0x41f1, 0x0033, 0xc1f3, 0x81f2, 0x4032,
    0x0036, 0xc1f6, 0x81f7, 0x4037, 0x01f5, 0xc035, 0x8034, 0x41f4,
    0x003c, 0xc1fc, 0x81fd, 0x403d, 0x01ff, 0xc03f, 0x803e, 0x41fe,
    0x01fa, 0xc03a, 0x803b, 0x41fb, 0x0039, 0xc1f9, 0x81f8, 0x4038,
    0x0028, 0xc1e8, 0x81e9, 0x4029, 0x01eb, 0xc02b, 0x802a, 0x41ea,
    0x01ee, 0xc02e, 0x802f, 0x41ef, 0x002d, 0xc1ed, 0x81ec, 0x402c,
    0x01e4, 0xc024, 0x8025, 0x41e5, 0x0027, 0xc1e7, 0x81e6, 0x4026,
    0x0022, 0xc1e2, 0x81e3, 0x4023, 0x01e1, 0xc021, 0x8020, 0x41e0,
    0x01a0, 0xc060, 0x8061, 0x41a1, 0x0063, 0xc1a3, 0x81a2, 0x4062,
    0x0066, 0xc1a6, 0x81a7, 0x4067, 0x01a5, 0xc065, 0x8064, 0x41a4,
    0x006c, 0xc1ac, 0x81ad, 0x406d, 0x01af, 0xc06f, 0x806e, 0x41ae,
    0x01aa, 0xc06a, 0x806b, 0x41ab, 0x0069, 0xc1a9, 0x81a8, 0x4068,
    0x0078, 0xc1b8, 0x81b9, 0x4079, 0x01bb, 0xc07b, 0x807a, 0x41ba,
    0x01be, 0xc07e, 0x807f, 0x41bf, 0x007d, 0xc1bd, 0x81bc, 0x407c,
    0x01b4, 0xc074, 0x8075, 0x41b5, 0x0077, 0xc1b7, 0x81b6, 0x4076,
    0x0072, 0xc1b2, 0x81b3, 0x4073, 0x01b1, 0xc071, 0x8070, 0x41b0,
    0x0050, 0xc190, 0x8191, 0x4051, 0x0193, 0xc053, 0x8052, 0x4192,
    0x0196, 0xc056, 0x8057, 0x4197, 0x0055, 0xc195, 0x8194, 0x4054,
    0x019c, 0xc05c, 0x805d, 0x419d, 0x005f, 0xc19f, 0x819e, 0x405e,
    0x005a, 0xc19a, 0x819b, 0x405b, 0x0199, 0xc059, 0x8058, 0x4198,
    0x0188, 0xc048, 0x8049, 0x4189, 0x004b, 0xc18b, 0x818a, 0x404a,
    0x004e, 0xc18e, 0x818f, 0x404f, 0x018d, 0xc04d, 0x804c, 0x418c,
    0x0044, 0xc184, 0x8185, 0x4045, 0x0187, 0xc047, 0x8046, 0x4186,
    0x0182, 0xc042, 0x8043, 0x4183, 0x0041, 0xc181, 0x8180, 0x4040
};

/************************************************************************/

/* Prototypes. */

unsigned int crc_calc(void *, int);
void delay(unsigned int);
int fclose(FILE *);
int fflush(FILE *);
int fgetc(FILE *);
FILE *fopen(const char *, const char *);
int fputc(int, FILE *);
void logerr(char *);
void mkdlmsg(struct Dl6000 *, unsigned char, unsigned char);
void mkmbmsg(unsigned char, unsigned char, int, unsigned char *);
void start_comms(struct Dl6000 *);
int txdl(struct Dl6000 *);
int ungetc(int, FILE *);

/************************************************************************/

main()
{
    int dlx;

    if ((port = fopen("COM", "r+b")) != NULL) {

        baud_rate = 9600;

        for (dlx = 0; dlx < MAXDL6000; dlx++) {

            /* Set addresses for testing. */
            dl6000[dlx].addr = (unsigned char) dlx;

            /* All DL6000's on the same port, i.e. the same multidrop, can
            ** use the same transmit and receive buffers.  (Master/slave
            ** comms demands one query and
```

```
            ** one response.)
            */
            dl6000[dlx].imsg = mbmsgin;
            dl6000[dlx].omsg = mbmsgout;

            /* Start comms and get version numbers. */
            start_comms(&dl6000[dlx]);
        }

        /* Other start-up processing, e.g. set date and time, request program
        ** code values, audit program codes, get stored transactions, etc.
        */
        /* ... */

        fclose(port);
        return (0);
    }
    else {
        logerr("Couldn't open port");
        return (-1);
    }
} /* main() */

/*****************************************************************************/

/* Start communications and get version number. */
void start_comms(struct Dl6000 *dl)
{
    dl->online = TRUE; /* txdl() will make FALSE if necessary. */

    mkdlmsg(dl, 2, 0x21);
    mkmbmsg(dl->addr, *(dl->omsg + 1), 2, dl->omsg);
    (void) txdl(dl);
    if (dl->online) {

        /* Store configuration information. */
        dl->nummtrs = *(int *) (dl->imsg + 4);
        dl->numcomps = *(int *) (dl->imsg + 6);
        dl->numvalves = *(int *) (dl->imsg + 8);
        dl->numfacs = *(int *) (dl->imsg + 10);
        dl->numrecipes = *(int *) (dl->imsg + 12);
        dl->numadds = *(int *) (dl->imsg + 14);
        dl->tempunits = *(unsigned char *) (dl->imsg + 16);

        /* Get version number. */
        mkdlmsg(dl, 2, 0x2a);
        mkmbmsg(dl->addr, *(dl->omsg + 1), 2, dl->omsg);
        (void) txdl(dl);
        if (*(dl->imsg + 3) == 0x2a) {

            /* Process version number. */
            /* ... */

        }
        else {
            dl->online = FALSE;
            logerr("Unable to get version number");
        }

    } /* if online. */

} /* start_comms() */
```

```
/*****************************************************************************/

/* Make Modbus RTU message frame.  Number of characters to transmit will be
** dcnt + 4.
*/
void mkmbmsg(unsigned char adr, /* Slave address. */
             unsigned char fc, /* Modbus RTU function code. */
             int dcnt, /* Number of characters in data field (0 to 250). */
             unsigned char *mbmsg)
{
    unsigned int crc;

    *(mbmsg) = adr; /* Address. */
    *(mbmsg + 1) = fc; /* Function code. */

    crc = crc_calc(mbmsg, dcnt + 2);
    *(mbmsg + dcnt + 2) = (unsigned char) (crc >> 8);
    *(mbmsg + dcnt + 3) = (unsigned char) (crc & 0xff);

} /* mkmbmsg() */

/*****************************************************************************/

/* Make DanLoad 6000 Modbus RTU message. */
void mkdlmsg(struct Dl6000 *dl,
             unsigned char dfl, /* Data field length. */
             unsigned char cmd) /* DanLoad 6000 command code. */
{
    *(dl->omsg + 1) = dl->fc; /* Use current Modbus RTU function code. */
    /* Set next Modbus RTU function code. */
    switch (dl->fc) {
        case 0x41:
            dl->fc = 0x42;
            break;
        case 0x42:
            dl->fc = 0x41;
            break;
        default:
            logerr("Invalid function code");
            break;
    } /* switch */

    *(dl->omsg + 2) = dfl; /* First byte of data field. */
    *(dl->omsg + 3) = cmd; /* Second byte of data field. */

} /* mkdlmsg() */

/*****************************************************************************/

/* Transmit message to DanLoad 6000 and process response. */
int txdl(struct Dl6000 *dl)
{
    int i, cnt, trycnt;
    int c;
    unsigned int crc;
    unsigned char fc; /* Function code. */

    /* Store function code for checking response. */
    fc = *(dl->omsg + 1);
    /* Initialize try count. */
    trycnt = 0;
    do {
```

```
(void) fflush(port); /* Flush I/O buffers. */

/* Transmit query. */
for (i = 0; i < *(dl->omsg + 2) + 4; i++) {
    fputc(*(dl->omsg + i), port);
}

/* Process response. */
i = 0;
crc = 0xffff; /* Flush with 1's. */

/* Wait up to 1000 ms for first character. */
for (cnt = 0; cnt < 10; cnt++) {

    if ((c = fgetc(port)) != EOF) {
        /* One or more characters received. */
        (void) ungetc(c, port);
        break;
    }
    delay(100);
    cnt++;
}
if (cnt == 10) {
    logerr("No response");
    trycnt++;
    continue;
}

do {
    while ((c = fgetc(port)) != EOF) {

        /* Copy received message into message buffer. */
        *(dl->imsg + i) = (unsigned char) c;

        /* Update checksum as we go. */
        crc16update(*(dl->imsg + i), &crc);

        cnt = 0; /* Reset with each character in. */
        i++;
    }

    /* Delay one character time assuming 10 bits per character. */
    switch (baud_rate) {
        case 110:
            delay(91);
            break;
        case 150:
            delay(67);
            break;
        case 300:
            delay(34);
            break;
        case 600:
            delay(17);
            break;
        case 1200:
            delay(9);
            break;
        case 2400:
            delay(5);
            break;
        case 4800:
```

```
                    delay(3);
                    break;
                case 9600:
                    delay(2);
                    break;
                default:
                    logerr("Invalid baud rate programmed");
                    break;
            } /* switch */

            cnt++;
        } while (cnt < 4); /* Count 4 characters. */

        /* Check checksum. */
        if (crc != 0) {
            logerr("Invalid checksum");
            trycnt++;
            continue;
        }

        /* Check function code.  An exception is a valid response at this
        ** level, but must be dealt with by the application.
        */
        if (*(dl->imsg + 1) != fc) {
            if (*(dl->imsg + 1) == (fc | 0x80)) {
                logerr("Exception");
            }
            else {
                logerr("Invalid function code");
                trycnt++;
                continue;
            }
        }

        return (0); /* Success. */

    } while (trycnt < 3);

    dl->online = FALSE; /* Flag offline on failure. */
    return (-1); /* Failure. */

} /* txdl() */

/*****************************************************************************/

/* System dependent function to delay x milliseconds. */
void delay(unsigned int x)
{
} /* delay() */

/*****************************************************************************/

/* System dependent error logging. */
void logerr(char *s)
{
} /* logerr() */

/*****************************************************************************/

/* End of file. */

////
```

7 Communications Commands

The commands are listed by command code. The command code (in hex) and version number(s) in which the command is available are in parentheses after the command name, e.g. Prompt Recipe (01h, v2.00 and above).

Note

DanLoad 6000 v2.00 was the first version to support automation system communications.

The command codes are documented in the following format:

Note

If you use a C language *struct* or a Pascal *record* to hold the data, your compiler must support "packed structures" if you intend to transfer data directly between the structure and the serial port. It may be more convenient to represent the query and response data as an array of bytes.

Query data: The "data field" transmitted to the DanLoad 6000, i.e. one or more "data values". The first byte of the data field is the "data field length".

The data is represented as a *quasi* C Language structure ("quasi" because the C language does not actually support arrays with their dimension initialized by a variable such as are indicated in some commands!) which may in practice may contain "holes" due to alignment requirements on different hardware platforms. However, the data is transmitted as if the structure were "packed", i.e. no holes.

Data type	# bytes
char	1
int	2
long	4
pointer	2

Counters, such as nummtrs, are always positioned in a message so that their location is independent of any variable length data, i.e. they precede any variable length data in a message.

Implicitly, an array of dimension zero occupies zero bytes.

Response data: The "data field" received from the DanLoad 6000, i.e. one or more "data values".

Note

The data field length is not necessarily constant for any given command code, e.g. the response to the Meter Totalizers command depends on the number of meters configured.

Status flags immediately set:

The status flags immediately set by this query, i.e. prior to the response, if there is no exception.

Status flags immediately cleared:

The status flags immediately cleared by this query, i.e. prior to the response, if there is no exception.

Exceptions: Exception response codes that may be received by the automation system in response to the command.
The DanLoad 6000 responds with exception response code 00h when it receives an invalid command code from the automation system. This exception response code is not included in the list of possible exception response codes for each valid command.

Description: Additional information describing the message structure and the processing that is performed.

Valid modes: The DanLoad 6000's operating modes (see Program Code 025) in which the command can be used.

7.1 Prompt Recipe (01h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int timout; /* Time-out seconds.  0=No time-out.  <0=Use configured
time-out. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

14h Keypad and display locked out

Status flags immediately cleared:

03h Operation time out
04h Recipe selected

Exceptions:

01h Passcode entry in progress
0Ah Primary alarm active
0Ch Transaction authorized
15h Operating mode is manual (low priority, may not be seen!)
20h Number of recipes < 2 **(Not DanLoad 6000 v5.00 and above)**
24h Keypad and display locked out

Description:

This command allows the automation system to let an operator select a recipe at the DanLoad 6000. The keypad and display are locked out to the automation system.

The DanLoad 6000 displays the Recipe Selection Display (starting with the first page with the first recipe highlighted) and prompts the operator to select a recipe.

The DanLoad 6000 sets a flag to indicate that a time-out has occurred. The DanLoad 6000 sets a flag to indicate that a recipe has been selected by the operator. The automation system can obtain the DanLoad 6000's status flags using the Request Status command.

Valid modes: Automatic.

7.2 Request Selected Recipe (02h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int recipenumber; /* 1->30 */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 0Ah Primary alarm active
15h Operating mode is manual (low priority, may not be seen!)
17h No recipe selected

Description: The DanLoad 6000 sets a flag to indicate that a recipe has been selected by the operator. The automation system can request the selected recipe using the Request Selected Recipe command.

The automation system specifies a recipe in the Authorize Transaction command. This can be the recipe selected by the operator or can be determined by the automation system independently.

Valid modes: Automatic.

7.3 Prompt Additives (03h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int timout; /* Time-out seconds.  0=No time-out.  <0=Use configured
time-out. */
    unsigned char addsel; /* Suggested additive selection bit map.  Re.
§8.5. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

14h Keypad and display locked out

Status flags immediately cleared:

03h Operation timed out
05h Additives selected

Exceptions:
09h Transaction in progress
0Ah Primary alarm active
0Ch Transaction authorized
13h No additives configured
15h Operating mode is manual (low priority, may not be seen!)
24h Keypad and display locked out

Description: This is used when the automation system requires the operator to select additives at the keypad.

The DanLoad 6000 displays the Additive Selection Screen (if it is not already displayed) with the suggested additive selection and prompts the operator to select additives. The suggested additive selection can be "no additives".

The DanLoad 6000 sets a flag to indicate that the additive selection is complete or that a time-out occurred, displays "Please wait" and waits for a command.

This command does not result in an additive selection being set in the DanLoad 6000. It just allows the automation system to obtain a possible additive selection from an operator at the DanLoad 6000. The automation system can set the additive selection using the Authorize Transaction command.

Valid modes: Automatic.

7.4 Request Selected Additives (04h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char addsel; /* Additive selection bit map.  Re. §8.5. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 0Ah Primary alarm active
0Ch Transaction authorized
15h Operating mode is manual (low priority, may not be seen!)
18h No additive selection made

Description: This is used following the Prompt Additives command (when the DanLoad 6000 indicates that an additive selection has been made) to obtain the additive selection (possibly none) from the DanLoad 6000.

The automation system specifies an additive selection in the Authorize Transaction command. This can be the additive selection made by the operator or can be determined by the automation system independently.

The additive selection (*addsel*) is described in a table below.

Valid modes: Automatic.

7.5 Time-Out Operation (05h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 01h Passcode entry in progress
08h Batch in progress
15h Operating mode is manual (low priority, may not be seen!)

Description: If the DanLoad 6000's is prompting for a recipe, prompting for additives, prompting for a preset volume, prompting for keypad input, displaying a message or waiting for the operator to start or abort a batch the operation is timed out.

Status flag 03h (Operation timed out) will be set subsequently.

The DanLoad 6000 displays "Please wait" and waits for a further command from the automation system.

Valid modes: Automatic.

7.6 Authorize Transaction (06h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int recipenumber; /* Recipe number 1->30. */
    unsigned char addselmthd; /* Additive selection method. */
    unsigned char addsel; /* Additive selection bit map.  Re. §8.5. */
    unsigned char side; /* Swing-arm side. */
    unsigned char numdataprompts; /* 0->5 */
    long dataitem[numdataprompts]; /* Data items. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int transeqnum; /* Prospective transaction sequence number. */
};
```

Status flags immediately set:

14h Keypad and display locked out

Status flags immediately cleared:

04h Recipe selected
05h Additive selected
0Ch Transaction ended
13h Transaction end requested

Exceptions:

01h Passcode entry in progress
0Ah Primary alarm active
0Ch Transaction authorized
10h Additive not available
15h Operating mode is manual (low priority, may not be seen!)
24h Keypad and display locked out
40h Invalid recipe number
48h Invalid number of data prompts
49h Invalid swing-arm side
4Eh Invalid additive selection method

Description: A transaction for recipe *recipenumber*, side *side* and additives determined by *addsel* and *addselmthd* is authorized but not started.

The automation system indicates on which *side* (1 or 2) loading is authorized to take place. The value 1 should be used for non-swing-arm applications.

*addsel*is the additive selection which determines which additive ratio outputs are cycled during a batch.

The additive selection methods (*addselmthd*) are:

0	Additive selection per *addsel* in the Authorize Transaction command, i.e. selected by automation system.
1	Additive selection per DanLoad 6000's configured additive selection method (program code 136).

Note

A fault in the standard DanLoad 6000 v5.40 (and below) firmware prevents the automation system from selecting the additives through communications if the configured additive selection method is "Inputs".

If additives are enabled via external (to the DanLoad 6000) contacts that complete electrical circuits between the DanLoad 6000's additive ratio outputs and the additive injectors, the automation system should select all (six) additives by setting *addselmthd* to 0 and *addsel* to 3Fh (00111111 binary), though fewer (than six) additives may be configured on the DanLoad 6000 (program code 135). An alternative is for the automation system to set bits in *addsel* based on the configured number of additives in the DanLoad 6000's response to the Start Communications command. Bear in mind that it is possible for each DanLoad to have a different configured number of additives.

The automation system can include up to five data items (each in the range 0 to 99999999) for the DanLoad 6000 to store in transaction storage memory, e.g. previously entered order number or driver number. In the manual operating mode, the DanLoad 6000 can prompt the operator for these data items and store them in transaction storage memory.

The DanLoad 6000 indicates the transaction sequence number (*transeqnum*) that will be used if a transaction is later started by starting the first batch on the transaction. The transaction sequence number rolls from 9999 to 0.

The DanLoad 6000 displays the loading display (if it is not already displayed) and displays "Please wait". (The Authorized Transaction command is the means by which the automation system causes the loading screen to be displayed.) The "large number" preset volume on the loading display is cleared (if it is present). The batch loaded, preset and remaining volumes and the transaction volume are zeroed.

Status flag 12h (Transaction authorized) will be set subsequently.

Valid modes: Automatic.

7.7 End Transaction (07h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char side; /* Swing-arm side. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int transeqnum; /* Transaction sequence number. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 08h Batch in progress
15h Operating mode is manual (low priority, may not be seen!)
22h No transaction authorized
49h Invalid swing-arm side

Description: The *side* should be the swing-arm side on which the transaction was authorized.

If there is a transaction in progress the DanLoad 6000 performs "end of transaction" processing.

If a transaction is authorized but there is no transaction in progress the DanLoad 6000 does not perform "end of transaction" processing but simply deauthorizes the transaction.

The DanLoad 6000 stores further transaction data, i.e. date, time, gross volume, net volume or mass/weight, average temperature, etc. The DanLoad 6000 responds with the transaction sequence number of the transaction that will be been ended.

Status flag 09h (Transaction in progress) is cleared subsequently, 0Ch (Transaction ended) is set subsequently and 12h (Transaction authorized) is cleared subsequently.

The DanLoad 6000 displays "Please wait".

In the manual operating mode a transaction is ended by pressing the "STOP" key when there is a transaction in progress and a batch is stopped or ended or there is no batch in progress. In automatic operating mode pressing the "STOP" key when there is a transaction in progress and a batch is stopped or ended or there is no batch in progress does not end the transaction. Instead, the DanLoad 6000 sets a flag indicating that a request has been made to end the transaction, to which the automation system can send an End Transaction command to one or more DanLoad 6000's (or simply clear or ignore the flag).

Valid modes: Automatic.

7.8 Prompt Preset Volume (08h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    long preset; /* Suggested preset volume. */
    long maxpreset; /* Override maximum preset volume. */
    int timout; /* Time-out seconds.  0=No time-out.  <0=Use configured
time-out. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

14h Keypad and display locked out

Status flags immediately cleared:

03h Operation timed out
06h Preset volume entered

Exceptions:
01h Passcode entry in progress
08h Batch in progress
0Ah Primary alarm active
15h Operating mode is manual (low priority, may not be seen!)
22h No transaction authorized
24h Keypad and display locked out
4Fh Invalid preset volume

Description: (The loading display is already displayed.)

The DanLoad 6000 prompts the operator to enter a preset volume. If the suggested preset volume is greater than zero, the DanLoad 6000 "pre-types" the suggested preset volume and the operator need only press "ENTER", or can press "CLEAR" and enter another volume.

If the override maximum preset volume is greater than zero the DanLoad 6000 ensures that the preset volume is less than or equal to the minimum of the configured maximum preset volume and the override maximum preset volume included in the Prompt Preset Volume command. The configured maximum preset volume is not modified by this command.

The DanLoad 6000 sets a flag to indicate that the preset volume is available or that a time-out occurred, displays "Please wait" and waits for a command.

The Prompt Preset Volume command does not clear the batch ended status flag. However, the Authorize Batch command does clear the batch ended status flag. Thus, depending on the design of the automation system, it may not be necessary for the automation system to clear the batch ended status flag with the Clear Status (13h) command after obtaining batch data with the Batch Data by Component (10h) command.

Valid modes: Automatic.

7.9 Request Preset Volume (09h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    long preset; /* Preset volume. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 0Ah Primary alarm active
15h Operating mode is manual (low priority, may not be seen!)
16h No preset volume entered

Description: The DanLoad 6000 responds with volume entered by the operator in response to the Prompt Preset Volume command.

The automation system can perform further validation on the preset volume entered by the operator.

A preset volume is set in the DanLoad 6000 using the Authorize Batch command.

Valid modes: Automatic.

7.10 Authorize Batch (0Ah, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    long preset; /* Preset volume. */
    int numcomps; /* Configured number of components. */
    int timout; /* Time-out seconds.  0=No time-out.  <0=Use configured
time-out. */
    struct {
        unsigned char use_gord; /* Use or don't use this backup gravity
or density.                                                    */
        long gord; /* Gravity or density. */
            unsigned char use_temp; /* Use or don't use this backup
temperature. */
        int temp; /* Temperature. */
    } comp[numcomps];
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int batchseqnum; /* Prospective batch sequence number. */
};
```

Note

With the weights and measures switch closed, backup temperatures and densities/gravities can be modified only if the corresponding program codes' weights and measures attributes are not set.

Status flags immediately set:

14h Keypad and display locked out

Status flags immediately cleared:

03h Operation timed out
06h Preset volume entered
0Dh Batch ended
0Eh Batch aborted (not started)

Exceptions:

01h Passcode entry in progress
04h Program code value is Weights and Measures (backup density/gravity or temperature)
0Ah Primary alarm active
0Bh Batch authorized
11h Program code value is read only
15h Operating mode is manual (low priority, may not be seen!)
22h No transaction authorized
24h Keypad and display locked out
45h Invalid program code value (backup density/gravity or temperature)
47h Invalid number of components
4Fh Invalid preset volume

Description:

The preset volume is "set" in the DanLoad 6000 so that it can be loaded. The DanLoad 6000 displays "Press START when ready or STOP to cancel". The "START" and "STOP" keys are enabled. The operator has *timout* seconds to press the "START" key or the batch is automatically aborted.

The preset volume must not be greater than the configured maximum preset volume (program code 078) and must not be less than the configured minimum preset volume (program code 079).

The preset volume is in the units of volume implied by the component's nominal K-factors and meter factors.

If the *use_gord* flag is set the configured backup density or gravity is replaced by the value in the message. The units of gravity or density should be the same as those in use by the DanLoad 6000 (program code 432, etc.)

If the *use_temp* flag is set the configured backup temperature is replaced by the *temp* in the message. The units of temperature should be the same as those in use by the DanLoad 6000 (program codes 427, 432, etc.)

The DanLoad 6000 indicates the (prospective) batch sequence number (*batchseqnum*) that will be used if a batch is started. The batch sequence number rolls from 9999 to 0.

Status flag 11h (Batch authorized) will be set.

Valid modes: Automatic.

7.11 Set Densities/Gravities (0Bh, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int numcomps; /* Configured number of components. */
    struct {
        unsigned char use_gord; /* Use or don't use this backup gravity
or density.                                          */
        long gord; /* Gravity or density. */
    } comp[numcomps];
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Note

With the weights and measures switch closed, backup temperatures and densities/gravities can be modified only if the corresponding program codes' weights and measures attributes are not set.

Status flags immediately set:

Status flags immediately cleared:

Exceptions:
04h Program code value is Weights and Measures
0Ah Primary alarm active
0Bh Batch authorized
11h Program code value is read only
15h Operating mode is manual (low priority, may not be seen!)
45h Invalid program code value
47h Invalid number of components

Description: The DanLoad 6000 uses a component's "backup density/gravity" (program code 457, etc.) when the "gravity/density" option (program code 456, etc.) is "Off". The automation system can download the latest known density/gravity prior to authorizing a batch.

If the *use_gord* flag is set the configured backup density or gravity is replaced by the value in the message. The units of gravity or density should be the same as those in use by the DanLoad 6000 (program code 432, etc.)

The backup densities or gravities can also be updated using the Authorized Batch command which is useful if there is an on line density input to the automation system. However, if densities or gravities are updated manually, e.g in the automation system's tank or product files, the Set Densities/Gravities command can be used to send the updated densities or gravities to the appropriate DanLoad 6000's.

Valid modes: Automatic.

7.12 Set Temperatures (0Ch, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int numcomps; /* Configured number of components. */
    struct {
       unsigned char use_temp; /* Use or don't use this backup
       temperature. */
        int temp; /* Temperature. */
    } comp[numcomps];
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Note

With the weights and measures switch closed, backup temperatures and densities/gravities can be modified only if the corresponding program codes' weights and measures attributes are not set.

Note

The Set Temperatures automation system command (0Ch) does not work correctly for components 2, 3 and 4 *in v2.0 thru v5.5*. There are two work-arounds: 1) Use the Set Program Code Value command (23h) to set the backup temperatures. 2) Use the Authorize Batch command (0Ah) to set the backup temperatures. (The backup temperatures are not used, if they are used at all, until a batch is in progress.)

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 04h Program code value is Weights and Measures

0Ah Primary alarm active
0Bh Batch authorized
11h Program code value is read only
15h Operating mode is manual (low priority, may not be seen!)
45h Invalid program code value
47h Invalid number of components

Description: The automation system can set the DanLoad 6000's backup temperatures.

If the *use_temp* flag is set the configured backup temperature is replaced by the *temp* in the message. The units of temperature should be the same as those in use by the DanLoad 6000 (program codes 427, 432, etc.)

The automation system could send this command, e.g. when an operator updates the backup temperatures in the automation system's meter or temperature probe file.

Valid modes: Automatic.

7.13 End Batch (0Dh, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int batchseqnum; /* Batch sequence number. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 06h No batch in progress
15h Operating mode is manual (low priority, may not be seen!)

Description: If there is a batch is progress and stopped it is ended. The DanLoad 6000 sets a flag to indicate that the batch has ended (which implies it is non-restartable). This is a signal for the automation system to request batch data. The DanLoad 6000 stores batch data and transaction data in "transaction storage memory".

The DanLoad 6000 displays "Please wait".

Status flag 0Ah (Batch in progress) is cleared, 0Dh (Batch ended) is set and 15h (Batch stopped) is cleared.

This is not the same as the Stop Batch command, which simulates the "STOP" key being pressed, and therefore potentially allows the batch to be restarted.

Under normal circumstances a batch ends (by itself) when the preset volume has been delivered.

Use the Time-Out Operation command to abort a batch that has been authorized but not yet started.

Valid modes: Automatic.

7.14 Start Batch (Remote "START") (0Eh, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int batchseqnum; /* Batch sequence number. */
};
```

Status flags immediately set:

0Ah Batch in progress

Status flags immediately cleared:

15h Batch stopped (resumable)

Exceptions: 0Ah Primary alarm active
14h No batch authorized
15h Operating mode is manual (low priority, may not be seen!)

Description: A batch must previously have been authorized using the Authorize Batch command. The DanLoad 6000 starts delivery of the batch. (This is the same as pressing the "START" key at the appropriate time.)

This is used to start a batch that has been authorized but is not yet in progress or to restart a batch in progress that has been stopped.

This is used for "remote control" of the DanLoad 6000, where there cannot be an operator in proximity of the DanLoad 6000. In these circumstances the DanLoad 6000 may be controlled entirely from the automation system, and the DanLoad 6000 may not even be fitted with a display or keypad. In this way, barges or railcars can be loaded from a control room.

Valid modes: Automatic.

7.15 Stop Batch (Remote "STOP") (0Fh, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int batchseqnum; /* Batch sequence number. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 06h No batch in progress
15h Operating mode is manual (low priority, may not be seen!)

Description: The batch in progress is stopped. This simulates the "STOP" key being pressed. The manner in which the batch is stopped is determined by the programmed "STOP" key action.

The DanLoad 6000 sets a flag to indicate that the batch has stopped (and is restartable) or that the batch has ended. If the batch has stopped (and is restartable) the DanLoad 6000 displays "Press START when ready or STOP to cancel".

The DanLoad 6000 displays "Please wait".

Valid modes: Automatic.

7.16 Batch Data by Component (10h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int batchseqnum; /* Batch sequence number. */
    int transeqnum; /* Transaction sequence number. */
    int recipenumber; /* Recipe number. */
    unsigned char side; /* Swing-arm side. */
    struct Ba_start_date_time {
        unsigned char year;
        unsigned char month;
        unsigned char day;
        unsigned char hours;
        unsigned char mins;
        unsigned char secs;
    } start;
    struct Ba_end_date_time {
        unsigned char year;
        unsigned char month;
        unsigned char day;
        unsigned char hours;
        unsigned char mins;
        unsigned char secs;
    } end;
    int nummtrs; /* Configured number of meters. */
    int numcomps; /* Configured number of components. */
    int numadds; /* Number of additives. */
    unsigned char numdataprompts; /* 0->5 */
    struct Totalizer {
        long grstotstrt; /* Gross meter totalizer at start of batch. */
       long nettotstrt; /* Net volume or mass/weight meter totalizer at
start of batch. */
        long grstotend; /* Gross meter totalizer at start of batch. */
         long nettotend; /* Net volume or mass/weight meter totalizer at
end of batch. */
    } totalizer[nummtrs];
    struct {
        long grs; /* Gross batch whole units. */
        long net; /* Net volume or mass/weight batch whole units. */
        int avetemp; /* Average temperature. */
        long avedens; /* Average density. */
        long avepres; /* Average pressure XXXX.XX. */
        int pct100; /* Actual % in batch, e.g. 60.03. */
    } comp[numcomps];
    struct {
         long grs100; /* Additive batch volume.  Implied scaling depends
on additive units. */
    } add[numadds];
    long dataitem[numdataprompts]; /* Data items. */
```

```
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 15h Operating mode is manual (low priority, may not be seen!)
26h No batch ended

Description: The DanLoad 6000 sets a flag to indicate that a batch has ended and cannot be restarted. The automation system can request data for the batch.

The DanLoad 6000's response include the configured number of components *numcomps* (program code 065) and the configured number of additives *numadds* (program code 135).

A component's batch net equals its batch gross and its temperature and density are "not defined", i.e. they are garbage, if a C_{tl} method (temperature option) is not specified for the component. (Program code 432, etc.)

Valid modes: Automatic.

7.17 Additive Totalizers (11h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int numadds; /* Number of additives. */
    struct {
         long grstot100; /* Gross totalizer.  Implied scaling depends on
additive units. */
    } add[numadds];
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: The DanLoad 6000's response include the configured number of additives *numadds* (program code 135). Up to six additives can be configured.

The DanLoad 6000's response includes the additive totalizers of all configured additives independent of whether the additive was selected during the last batch.

Valid modes: Manual.
Automatic.

7.18 Request Status (12h, v2.00 and above)

Note

See also Enhanced Request Status (3Ch, v5.11 and above).

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned long status; /* Status bits 31 thru 0.  Re. §8.1. */
    unsigned char side; /* Swing-arm side (0->2). */
    long grsvol; /* Batch gross volume whole units. */
    long netvol; /* Batch net volume or mass/weight whole units. */
    unsigned char safety; /* Safety circuits status, including those with
alarm action "Off" for DanLoad 6000 v5.11 and above. */
     unsigned char almcd; /* Alarm code of oldest active primary alarm.
See §8.3. */
    unsigned char alarm_byte_9; /* Alarm bits 79 thru 72.  Re. §8.3. */
    unsigned char alarm_byte_8; /* Alarm bits 71 thru 64. */
    unsigned char alarm_byte_7; /* Alarm bits 63 thru 56. */
    unsigned char alarm_byte_6; /* Alarm bits 55 thru 48. */
    unsigned char alarm_byte_5; /* Alarm bits 47 thru 40. */
    unsigned char alarm_byte_4; /* Alarm bits 39 thru 32. */
    unsigned char alarm_byte_3; /* Alarm bits 31 thru 24. */
    unsigned char alarm_byte_2; /* Alarm bits 23 thru 16. */
    unsigned char alarm_byte_1; /* Alarm bits 15 thru 8. */
    unsigned char alarm_byte_0; /* Alarm bits 7 thru 0. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: *status* is a binary representation of the DanLoad 6000's 32 status flags.

safety is a binary representation of the current logical state ("open" or "closed") of the DanLoad 6000's eight general-purpose safety

circuits (program codes 345 through 352). The least significant bit is safety circuit 1, etc. Safety circuits that are not configured on the DanLoad 6000 are indicated to be "open", i.e. the corresponding bit is 0.

alarm_byte_0 through *alarm_byte_9* indicate active (raised and not reset) primary alarms. The automation system can maintain a copy of the "alarm bytes" so that it can determine when primary alarms are raised and cleared for logging purposes.

Valid modes: Manual.
Automatic.

7.19 Clear Status (13h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned long status; /* Status bits mask. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

Status flags immediately cleared:

Various

Exceptions: 12h Status not set or cannot be reset
15h Operating mode is manual

Description: This command allows the automation system to clear certain status flags. See §8.1 for the status flags (marked "α") that can be cleared using the Clear Status command.

*status*is a binary representation of the DanLoad 6000's status flags. A "1" in any bit indicates a request to clear the corresponding status flag on the DanLoad 6000.

Valid modes: Automatic.

7.20 Reset Primary Alarms (14h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char alarm_byte_9; /* Alarm bits 79 thru 72.  Re. §8.3. */
    unsigned char alarm_byte_8; /* Alarm bits 71 thru 64. */
    unsigned char alarm_byte_7; /* Alarm bits 63 thru 56. */
    unsigned char alarm_byte_6; /* Alarm bits 55 thru 48. */
    unsigned char alarm_byte_5; /* Alarm bits 47 thru 40. */
    unsigned char alarm_byte_4; /* Alarm bits 39 thru 32. */
    unsigned char alarm_byte_3; /* Alarm bits 31 thru 24. */
    unsigned char alarm_byte_2; /* Alarm bits 23 thru 16. */
    unsigned char alarm_byte_1; /* Alarm bits 15 thru 8. */
    unsigned char alarm_byte_0; /* Alarm bits 7 thru 0. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 01h Passcode entry in progress
15h Operating mode is manual (low priority, may not be seen!)

Description: This command allows multiple active primary alarms to be reset. If the chronologically oldest active primary alarm is reset this command has the same effect as a primary alarm reset via the alarm reset screen or the alarm reset switch (program code 344).

Valid modes: Automatic.

7.21 Meter Totalizers (15h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int nummtrs; /* Configured number of meters. */
    struct {
        long grstot; /* Gross totalizer rolls 999999999 to 0. */
           long nettot; /* Net volume or mass/weight totalizer rolls
999999999 to 0. */
    } meter[nummtrs];
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: In a blending configuration, it is not always possible to identify which component generates unauthorized flow. More than one component may flow through a meter. The DanLoad 6000 maintains running meter totalizers (which include unauthorized flow) and running component totalizers (which exclude unauthorized flow).

Typically, the Request Meter Totalizers command would be used at the end of each shift or business day to read all meters. Thus, total meter throughput for the period can be determined and reconciled.

See the "Manual of Petroleum Measurement Standards", Chapter 12, "Calculation of Petroleum Quantities". When a load ticket is required, the loaded volume should be calculated as the truncated stop reading minus the truncated start reading from a non-resettable totalizer. "Show meter readings from the non-resettable readout, and truncate (that is, disregard) all values less than a whole barrel or gallon."

Valid modes: Manual.
Automatic.

7.22 Component Totalizers (16h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int numcomps; /* Configured number of components. */
    struct {
        long grstot; /* Gross totalizer rolls 999999999 to 0. */
           long nettot; /* Net volume or mass/weight totalizer rolls
999999999 to 0. */
    } comp[numcomps];
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: In a blending configuration, it is not always possible to identify which component generates unauthorized flow. More than one component may flow through a meter. The DanLoad 6000 maintains running meter totalizers (which include unauthorized flow) and running component totalizers (which exclude unauthorized flow).

See the "Manual of Petroleum Measurement Standards", Chapter 12, "Calculation of Petroleum Quantities". When a load ticket is required, the loaded volume should be calculated as the truncated stop reading minus the truncated start reading from a non-resettable totalizer. "Show meter readings from the non-resettable readout, and truncate (that is, disregard) all values less than a whole barrel or gallon."

Valid modes: Manual.
Automatic.

7.23 Unauthorized Flow (17h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int nummtrs; /* Configure number of meters. */
    struct {
        long ufgtot10; /* Gross unauthorized flow 10ths rolls 999999.9 to
0.0. */
         long ufntot10; /* Net volume or mass/weight unauthorized flow
10ths rolls 999999.9 to 0.0. */
    } meter[nummtrs];
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: Unauthorized flow, i.e. flow when the flow control valve has not been commanded to be open, is accumulated for each meter. Unauthorized flow is included in the individual meter totalizers. Unauthorized flow is not included in the individual component totalizers.

The automation system may wish to request unauthorized flow "between batches", i.e. just before authorizing a batch, and during "end of day" processing. Requesting unauthorized flow does not zero the unauthorized flow counters or reset an unauthorized flow alarm. (An unauthorized flow alarm can be reset at the DanLoad 6000 or by using the Alarm Reset command.)

Valid modes: Manual.
Automatic.

7.24 Data Code Value (18h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int dc; /* Data code. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int dc; /* Data code. */
    unsigned char dcvl; /* Data code value length. */
    unsigned char dcv[dcvl]; /* Data code value. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 52h Invalid data code

Description: See the list of data codes in the "DanLoad 6000 Reference Manual".

dcvl is the number of characters required to store the data code value.

dcv is a binary representation of the data code value.

Valid modes: Manual.
Automatic.

7.25 Request Meter Values (19h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int mtrnum; /* Meter number (1->4). */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int mtrnum; /* Meter number. */
    long grstot; /* Gross totalizer rolls 999999999 to 0. */
    long nettot; /* Net volume or mass/weight totalizer rolls 999999999
to 0. */
    long grs; /* Batch gross whole units. */
    long net; /* Batch net volume or mass/weight whole units. */
    long rate3; /* Average over one second flow rate. */
    long rate5; /* Average batch flow rate.  Not implemented. */
    long rate4; /* Maximum average over one second flow rate. */
    long mtrfac; /* Current meter factor. */
    unsigned int plscnt; /* Current pulse count rolls 65535 to 0. */
    int numfacs; /* Number of factors or flow rates. */
    struct {
        unsigned long pls; /* Number of pulses at flow rate. */
    } pls[numfacs];
    long ufg10; /* Gross unauthorized flow 10ths since start of last
batch. */
    long ufn10;      /* Net volume or mass/weight unauthorized flow
                        10ths since start of last batch.  Net = gross for
                        unauth. flow volume. */
    int temp10; /* Current temperature. */
    long dens; /* Current density. */
    long pres; /* Current pressure XXXX.XX. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 41h Invalid meter number

Description: The DanLoad 6000 responds with the current values of various data.

Valid modes: Manual.
Automatic.

7.26 Request Component Values (1Ah, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int compnum; /* Component number. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int compnum; /* Component number. */
    long grstot; /* Gross totalizer rolls 999999999 to 0. */
     long nettot; /* Net volume or mass/weight totalizer rolls 999999999
to 0. */
    long grs; /* Batch gross whole units. */
    long net; /* Batch net volume or mass/weight whole units. */
    int avetemp; /* Average batch temperature. */
    long avedens; /* Average density/gravity. */
    long avepres; /* Average pressure XXXX.XX. */
    int avecls5; /* Average valve closure pulses. */
    int percent; /* Component batch percentage. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 42h Invalid component number

Description: The DanLoad 6000 responds with the current values of various data.

Valid modes: Manual.
Automatic.

7.27 Request Power Fail Date and Time (1Bh, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    struct Power_fail_date_time {
        unsigned char year;
        unsigned char month;
        unsigned char day;
        unsigned char hours;
        unsigned char mins;
        unsigned char secs;
    } pwrfail;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: *pwrfail* is the date and time at which the DanLoad 6000 last lost power.

The automation system may wish to log this information each time a DanLoad 6000 is brought on line.

Valid modes: Manual.
Automatic.

7.28 Display Message (1Ch, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    char msgtxt[129]; /* Null terminated message text. */
    int pmptfw; /* Prompt field width (0->8).  pmptfw < 0 => 0.  pmptfw
> 8 => 8. */
    unsigned char inpctl; /* Input control. */
    int timout; /* Time-out # seconds. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

14h Keypad and display locked out

Status flags immediately cleared:

03h Operation timed out
07h Keypad data available

Exceptions:

01h Passcode entry in progress
08h Batch in progress
0Ah Primary alarm is active
15h Operating mode is manual (low priority, may not be seen!)
24h Keypad and display locked out

Description:

The DanLoad 6000 displays *msgtxt* on the bottom line of the display. Invalid characters in *msgtxt* are replaced by an asterisk. If *pmptfw* is zero all forty columns are used. If *pmptfw* is non-zero the first thirty two columns are used. If necessary, the message text is scrolled within the appropriate number of columns. One space is inserted at the end of the message for scrolling purposes.
If *pmptfw* is less than zero then zero is assumed. If *pmptfw* is greater than eight then eight is assumed.

If *pmptfw* is non-zero the DanLoad 6000 prompts for *pmptfw* input characters starting at column thirty three. The "CLEAR" key clears the input field and restarts input entry. The "ENTER" key terminates input entry. If "ENTER" is pressed when the input field is blank the keypad data available to the automation system has a value of zero.

Valid input control (*inpctl*) values are:

0	Echo numeric keys on the display.
1	Display an asterisk "*" for each numeric key pressed.

If *pmptfw* is greater than zero the time-out specified by *timout* is reset whenever a key is pressed. If *pmptfw* is zero a time-out occurs after the specified *timout*; this allow the automation system to display a message "for a few seconds". If *timout* is zero the automation system can terminate the operation with a Time-Out Operation command. If a time-out occurs the keypad data (see Request Keypad Data command) is not available to the automation system since the timed-out status flag is set and the keypad data available status flag is clear.

Valid time-out (*timout*) values are:

-1	Use the configured time-out, i.e. the number of seconds specified by program code 036.
0	No time-out, i.e. display/prompt until a Time-Out Operation command.
> 0	Use the supplied time-out, i.e. the number of seconds specified in the message.

Valid modes: Automatic.

7.29 Request Keypad Data (1Dh, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    long keypad_data;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 08h Batch in progress
0Ah Primary alarm active
0Eh No keypad data available
15h Operating mode is manual (low priority, may not be seen!)

Description: This command is used to retrieve input requested via the Display Message command. When keypad input is terminated the DanLoad 6000 sets a flag indicating that keypad data is available.

The *keypad_data* is a binary representation of the number entered by the operator at the DanLoad 6000.

Valid modes: Automatic.

7.30 Request Transaction Storage Status (1Eh, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;

    int oldest_trseq; /* Oldest transaction seq # stored (0-9999). */
    int numtr; /* Number of transactions stored (0-10,000). */
    int maxnumtr; /* Maximum number of transactions that can be stored.
*/
    unsigned char trcfgerr; /* Transaction configuration error. */

    int oldest_baseq; /* Oldest batch seq # stored (0-9999). */
    int numba; /* Number of batches stored (0->10,000). */
    int maxnumba; /* Maximum number of batches that can be stored. */
    unsigned char bacfgerr; /* Batch configuration error. */

};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: A transaction/batch is not considered "stored" ("on file") until it has ended. (Thus, there may be batches on file for a transaction which is not yet on file.)

The transaction/batch file is considered "empty" if and only if the number of transactions/batches on file is zero.

The transaction/batch configuration flag is cleared on a "warm start" (power-up) and is set if a transaction/batch storage code's offset is configured such that the associated value does not fit in the configured transaction/batch record length.

The transaction/batch sequence numbers are non-resetable, 4-digit numbers that roll from 9999 to 0. Fewer than 10,000

transactions/batches may be stored, depending on the configuration; the transaction/batch sequence numbers still roll from 9999 to 0.

The sequence number of the newest transaction stored ("on file") can be computed as...

(oldest_trseq + numtr - 1) % 10000

The sequence number of the newest batch stored ("on file") can be computed as...

(oldest_baseq + numba - 1) % 10000

...where "%" is the C language modulus operator.

Valid modes: Manual.
Automatic.

7.31 Transaction Data by Component (1Fh, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int transeqnum; /* Transaction sequence number. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int transeqnum; /* Transaction sequence number. */
    int recipenumber; /* Recipe number. */
    unsigned char side; /* Swing-arm side. */
    long gross; /* Transaction gross volume. */
    long net; /* Transaction net volume or mass/weight. */
    struct Tr_start_date_time {
        unsigned char year;
        unsigned char month;
        unsigned char day;
        unsigned char hours;
        unsigned char mins;
        unsigned char secs;
    } start; /* Transaction start date and time. */
    struct Tr_end_date_time {
        unsigned char year;
        unsigned char month;
        unsigned char day;
        unsigned char hours;
        unsigned char mins;
        unsigned char secs;
    } end; /* Transaction end date and time. */
    int nummtrs; /* Configured number of meters. */
    unsigned char numdataprompts; /* 0->5 */
    struct Totalizer {
        long grstotstrt; /* Gross meter totalizer at start of batch. */
       long nettotstrt; /* Net volume or mass/weight meter totalizer at
start of batch. */
        long grstotend; /* Gross meter totalizer at end of batch. */
       long nettotend; /* Net volume or mass/weight totalizer at end of
batch. */
    } totalizer[nummtrs];
    long dataitem[numdataprompts]; /* Data items entered by operator in
manual or                                    downloaded by automation
system in automatic. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 02h No transaction ended

43h Invalid transaction sequence number

Description: When a transaction ends the DanLoad 6000 sets a flag indicating that the transaction has ended. This command is used to request transaction data from the DanLoad 6000 when a transaction ends.

In manual, a transaction is ended by pressing the "STOP" key when there is a transaction in progress and no batch in progress. In automatic, a transaction is ended by the automation system via the End Transaction command.

Valid modes: Manual.
Automatic.

7.32 Initialize Transaction Storage (20h, v3.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 09h Transaction in progress

Description: The DanLoad 6000 responds immediately and then initializes transaction storage (the transaction file and the batch file) using the configured start pages, numbers of pages and record lengths.

Transaction and batch records "on file" are deleted. The "oldest record on file" for each file is indeterminate until another record is put on file.

Transaction storage initialization takes several seconds to complete.

The rolling, 4-digit transaction and batch sequence numbers are not modified by this command.

Valid modes: Manual.
Automatic.

7.33 Start Communications (21h, v2.00 and above)

Note

See also Enhanced Start Communications (3Bh, v5.11 and above).

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int nummtrs; /* Configured number of meters. */
    int numcomps; /* Configured number of components. */
    int numvalves; /* Configured number of valves. */
    int numfacs; /* Configured number of factors/component. */
    int numrecipes; /* Number of recipes configured. */
    int numadds; /* Number of additives configured. */
    unsigned char tempunits; /* 0=Celsius.  1=Fahrenheit. */
    struct {
        unsigned char temp_option; /* Temperature correction option. */
        unsigned char pres_option; /* Pressure correction option. */
    } comp[numcomps];
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: When the DanLoad 6000 is powered-up/reset, exits program mode or has detected an automation system communications failure, communications can be started with this command.

The DanLoad 6000 will not respond to any command from the automation system until communications have been started.

If communications are already started when this command is received then the communications link is reinitialized but operations at the DanLoad 6000 are unaffected.

When a communications failure occurs on a communications channel then communications on that channel can be started using this command.

The alternating function code sequence is reset. Either function code (41h or 42h) can be used for this query. The next (non-retry) query from the automation system should use the other function code.

The query is six characters long; address (one character), function code (one character), data field length (one character), command code (one character) and checksum (two characters). Valid Start Communications commands for a DanLoad 6000 with communications address 01h are:

01 41 02 21 90 B4

and:

01 42 02 21 60 B4

The temperature correction options (per component) follow program codes 432, 435, etc., with zero being "Off". Refer to the "DanLoad 6000 Reference Manual".

The pressure correction options (per component) follow program code 444, 447, etc., with zero being "Off". Refer to the "DanLoad 6000 Reference Manual".

Valid modes: Manual.
Automatic.

7.34 Request Program Code Values and Attributes (22h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int pc1;
    int pcN;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char pcv1l; /* Program code value 1 length. */
    unsigned char pcv1[sizeof(pcv1)];
    unsigned char pca1; /* Re. §8.6. */
...
    unsigned char pcvNl; /* Program code value N length. */
    unsigned char pcvN[sizeof(pcvN)];
    unsigned char pcaN; /* Re. §8.6. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 03h Response's data field too long
44h Invalid program code number

Description: There are many program codes. A program code is a number. A program code identifies a variable (parameter) in the DanLoad 6000. A program code value is the value of the variable identified by the program code. A program code value can be a number, a character or a string of characters. A program code's attributes are properties associated with that program code.

Since the automation system can request a range of program code values and attributes the response's data field is variable-length. The maximum response message length for Modbus RTU is 256 bytes (including check characters). If the automation system requests a range of program code values and attributes whose combined length is too large to fit in a single response the DanLoad 6000 responds with an exception. The automation system should request the required program code values and attributes by making multiple

requests for smaller ranges of program code values and attributes, e.g. ten at a time or compute the response's length.

This can be used to find out if a program code value is stored in transaction storage memory.

When a program code value is changed (in "Setup"), the program code's attribute "value changed" is set and the DanLoad 6000's (master) "value changed" flag is set. This can be used by an automation system to find out if a program code value has been modified while the DanLoad 6000 was "offline" or in "Setup". (The automation system can detect exit from "Setup".) An automation system can reset the "value changed" flag and check "value changed bits" to determine which program code values have changed. An automation system can use the Modify Program Code Attribute command to reset the "value changed bit". Thus, it is not possible to change the DanLoad 6000's setup parameter without the automation system being aware of it.

The response contains a binary representation of the program code's attribute byte. This can be used to determine a program code value has been changed by an operator.

This command can be used to request all recipe configuration values, all additive configuration values, etc.

Valid modes: Manual.
Automatic.

7.35 Set Program Code Value (23h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int pc; /* Program code. */
    unsigned char pcv[sizeof(pcv)]; /* Program code value. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

08h Program code value(s) changed

Status flags immediately cleared:

Exceptions:
04h Program code value is Weights and Measures
09h Transaction in progress
0Ah Primary alarm active
11h Program code value is read only
15h Operating mode is manual (low priority, may not be seen!)
44h Invalid program code
45h Invalid program code value

Description:
The query's data field is variable-length.

This command can be used to configure the DanLoad 6000 from an automation system. The DanLoad 6000 validates the program code value being set and responds with exception code 45h if it is not valid. If the program code value is valid, the program code's "value changed" attribute and the "global" program code value(s) changed status flag are set. The program code value's size can be found using the Request Program Code Values and Attributes (22h) command.

The Configure Recipe command (27h) provides a quick, convenient way to configure recipes dynamically.

Note

Some program code values take effect immediately. Other program code values do not take effect until the DanLoad 6000 is reset. The DanLoad 6000 should be reset after program code values have been modified. The Reset Unit (30h) command can be used.

Note

Program code values can be modified only one at a time using command code 23h. When downloading a "complete configuration", i.e. all program codes, to a DanLoad 6000 is is usually quickest to request all program values (using command code 22h) and to download only those that differ.

Valid modes: Automatic.

7.36 Modify Program Code Attribute (24h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int pc; /* Program code. */
    unsigned char bit; /* Attribute bit (0->7). */
    unsigned char state; /* TRUE to set, FALSE to clear. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 04h Program code value is Weights and Measures
09h Transaction in progress
0Ah Primary alarm active
15h Operating mode is manual (low priority, may not be seen!)
44h Invalid program code
55h Invalid bit #

Description: This command can be used to modify a program code's attributes, i.e. read only, Weights and Measures, data logging and value changed.

Only one attribute bit can be set or cleared at a time, i.e. *bit* equal to 3 means change bit 3 (which is the Value Changed attribute).

The Weights and Measures attribute can be changed only if the Weights and Measures switch is open.

Valid modes: Automatic.

7.37 Request Value Changed Attributes (25h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char valchg[104]; /* Sufficient for MAXPC bits. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 0Ch Transaction authorized
15h Operating mode is manual (low priority, may not be seen!)

Description: This command can be used to find out which program code values have been modified since their "value changed" attributes were last cleared. *valchg[0]* is the "value changed" attribute of program code 001, etc. The automation system need only look at program codes in which it is interested.

Valid modes: Automatic.

7.38 Clear Value Changed Attributes (26h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

Status flags immediately cleared:

08h Program code value(s) changed

Exceptions: 0Ch Transaction authorized
15h Operating mode is manual (low priority, may not be seen!)

Description: This command can be used to clear every program code's "value changed" attribute.

Valid modes: Automatic.

7.39 Configure Recipe (27h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int recipenumber; /* Recipe number (1->30). */
    int numcomps; /* Configured number of components. */
    char name[17]; /* Null terminated recipe name of length 16. */
    int cmppct[numcomps]; /* Component percentages in hundredths. */
    int cmpseq[numcomps]; /* Component sequence or low/high proportion.
*/
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

08h Program code value(s) changed

Status flags immediately cleared:

Exceptions:
0Ah Primary alarm active
0Ch Transaction authorized
15h Operating mode is manual (low priority, may not be seen!)
40h Invalid recipe number
47h Invalid number of components

Description: The is a short-cut for configuring recipes (instead of setting the individual program code values).

In the case of an in-line blender unit type there are some other considerations. The low/high proportion flags should be configured carefully; it is possible that a particular component is high proportion (flag = 0) for some recipes and low proportion (flag = 1) for different recipes *on the same DanLoad 6000*. This is a function of the size of the meter (and thus the usable range of flow rates) through which the component flows. The overall recipe flow rates (program codes 776, 777, etc.) *may* need to be modified if component percentages are modified via communications. (The component stop rates and

calibrated meter factor flow rates should not normally be modified dynamically.)

The configured number of recipes (program code 480) is not automatically increased or decreased by the Configure Recipe command.

Valid modes: Automatic.

7.40 Get Date and Time (28h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    struct Op_date_time {
        unsigned char year;
        unsigned char month;
        unsigned char day;
        unsigned char hours;
        unsigned char mins;
        unsigned char secs;
    } op_date_time;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: The DanLoad 6000 has a battery-backed real-time clock which maintains the date and time even when the DanLoad 6000 is powered down.

The DanLoad 6000 responds with the *year*, *month*, *day*, *hours*, *mins*, and *secs* from its battery-backed real-time clock.

Valid modes: Manual.
Automatic.

7.41 Set Date and Time (29h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    struct Set_date_time {
        unsigned char year;
        unsigned char month;
        unsigned char day;
        unsigned char hours;
        unsigned char mins;
        unsigned char secs;
    } set_date_time;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 0Ch Transaction authorized
50h Invalid date
51h Invalid time

Description: The DanLoad 6000 validates the date and time from the automation system.

If both the date and time are valid the DanLoad 6000 sets its battery-backed real-time clock.

Valid modes: Manual.
Automatic.

7.42 Request Firmware Versions (2Ah, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
     char cp1fwver[strlen(cp1fwver) + 1]; /* CPU-1 program EPROM null
terminated. */
     char cp2fwver[strlen(cp2fwver) + 1]; /* CPU-2 program EPROM null
terminated. */
      char msgver[strlen(msgver) + 1]; /* CPU-1 message EPROM null
terminated. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: This can be used to verify the installed firmware version. The firmware versions are in the format "1.23". The automation system may wish to prefix a lowercase "v", i.e. "v1.23", to match the format displayed by the DanLoad 6000.

The automation system should ensure that the length of each version number string does not exceed its available storage space.

Valid modes: Manual.
Automatic.

7.43 Read Input (2Bh, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int ioptp; /* I/O point type. */
    int ioptnum; /* I/O point number. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char val; /* Discrete input value. */
};

struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned int val; /* Meter, RTD or current loop value. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 4Ah Invalid I/O point type
4Bh Invalid I/O point number

Description: Valid I/O point types (*ioptp*) are:

0	Meter input.
1	RTD input.
2	Current loop input.
3	Discrete input. NOTE: The automation system should specify the actual discrete input (rather than the "inverted discrete input", #51 and above) for the I/O point number.

The format of the response depends on *ioptp*.

Valid modes: Manual.
Automatic.

7.44 Write Output (2Ch, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int ioptp; /* I/O point type. */
    int ioptnum; /* I/O point number. */
    unsigned int val; /* Value. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 4Ah Invalid I/O point type
4Bh Invalid I/O point number
4Ch Invalid output value

Description: Valid I/O point types (*ioptp*) are:

4	Discrete output.

Valid modes: Manual.
Automatic.

7.45 DUART Diagnostic (2Dh, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char instruction; /* 0=start diagnostic, 1=request results.
*/
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char busy_status; /* 0x10=Ch B busy, 0x01=Ch A busy. */
    struct {
        int locerr;
        int exterr;
        int bauderr;
        unsigned char fmterr;
        unsigned char interr;
    } results;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 0Ch Transaction authorized
1Bh Diagnostic not started
1Ch Diagnostic running

Description: *This command is for chamber testing or other diagnostic purposes only.*

At least one channel (the which receives this command) is busy, so at most one channel (the channel that is not busy) can be tested.

This command can be used to start a DUART diagnostic and subsequently to request the results of the diagnostic. If the results of the diagnostic are requested before the diagnostic is complete or if the results of the diagnostic are requested without the diagnostic having been started an exception response is given.

When the DUART diagnostic has completed, the response should be interpreted as follows:

locerr is the number of failures out of 256 characters transmitted during the local loopback test. (Thus, *locerr == 0* means "passed" and *locerr > 0* means "failed".)

exterr is the number of failures out of 256 characters transmitted during an external loopback test. (Thus, *exterr == 0* means "passed" and *exterr > 0* means "failed".)

*bauderr*is the baud rate error mask for eight configurable baud rates from 300 thru 38400, corresponding to bits 0 (least significant) thru bit 7 of *bauderr*. A bit being set (1) means that the DUART "failed" at the corresponding baud rate. A bit being clear (0) means that the DUART "passed" at the corresponding baud rate, e.g. *bauderr & 0x04 == 0x04* means that the DUART failed the 1200 bps test.

fmterr is the data format error mask for the "parity", "stop bit" and "data bit" tests, corresponding to bits 0 (least significant) thru bit 2 of *fmterr*. A bit being set (1) means that the DUART failed the corresponding test. A bit being clear (0) means that the DUART passed the corresponding test, e.g. *fmterr & 0x07 == 0* means that all tests passed.

*interr*is the interrupt error mask for the receive ready, FIFO full and transmit ready interrupts. This is not implemented via automation system communications.

Valid modes: Manual.
Automatic.

7.46 ARCNET Diagnostic (2Eh, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char dsreg; /* COM20020's Diagnostic Status Register. */
    unsigned char cfgreg; /* COM20020's Configuration Register. */

};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 09h Transaction in progress

Description: *This command is for chamber testing or other diagnostic purposes only; the DanLoad 6000 responds only after several seconds.*

THIS COMMAND WILL BE AVAILABLE IN A FUTURE RELEASE.

Valid modes: Manual.
Automatic.

7.47 Request Crash Data (2Fh, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int cpu; /* 1 or 2. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned int stack_pointer;
    unsigned int program_counter;
    unsigned int interrupt_mask;
    unsigned int interrupt_mask1;
    unsigned int win_sel_reg;
    unsigned int cur_task;
    int val; /* Error code. */
    void *task_adr;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 09h Transaction in progress
46h Invalid CPU number

Description: The crash data can assist Daniel Industries engineers in the analysis of "fatal errors" caused by hardware or software malfunction.

The crash data contains (in order); stack pointer at time of failure, program counter at time of failure, interrupt mask at time of failure, interrupt mask 1 at time of failure, Window Select Register at time of failure, O/S task number at time of failure, error code and O/S task address (task entry point) at time of failure.

Valid modes: Manual.
Automatic.

7.48 Reset Unit (30h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 09h Transaction in progress

Description: The DanLoad 6000 responds to the automation system and then resets (similar to power up).

The DanLoad 6000's CPU-1 issues a reset instruction which resets both CPU-1 and CPU-2.

Valid modes: Manual.
Automatic.

Note

Previouly documented as being valid only in the automatic operating mode and giving an exception in the manual operating mode. Refer to standard DanLoad 6000 fault 222.

7.49 Last Key Pressed (31h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char keycode;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 1Ah No key pressed, i.e. no key code available

Description: The DanLoad 6000 responds with the key code of the keypad key most recently pressed.

In DanLoad 6000 v5.11 and above, the Key Pressed (0Bh) status flag (in the response to the Request Status (12h) and Enhanced Request Status (3Ch) commands) indicates if a key code is available. The Key Pressed status flag can be cleared by the automation system using the Clear Status (13h) command if a "new" key code is required.

Valid modes: Manual.
Automatic.

7.50 RAM Tests (32h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int cpu; /* 1 or 2. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char ok; /* 0 <=> failed */
    unsigned char *addr; /* Address of first failure. */
    unsigned char pg; /* Data page. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 09h Transaction in progress
46h Invalid CPU number

Description: This command is for chamber testing or other diagnostic purposes only; the DanLoad 6000 responds with the results of the last RAM test that ran, i.e. on power-up or via diagnostics, on the specified CPU. In order to force RAM tests to be performed through communications, first send a Reset Unit command, wait until the DanLoad 6000 responds to a Start Communications command and then send the RAM Tests command for one or both CPU's.

Valid modes: Manual.
Automatic.

7.51 Swing-arm Side (33h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char side; /* Swing-arm side (0->2). */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: The DanLoad 6000 responds with the swing-arm side (0, 1 or 2) as determined by the swing-arm switches on the CPU board and the configured "Side detect method".

Valid modes: Manual.
Automatic.

7.52 Installed Boards (34h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char btp[7]; /* I/O slot array. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: The DanLoad 6000 responds with the (currently installed) I/O board types detected at reset.

The board types are:

0	No board installed
1	DC I/O
2	AC I/O
3	8-Channel Analog Inputs
4	2-Channel Meter Pulse
5	Enhanced I/O
6	2-Channel Analog Inputs
7	4-Channel Meter Pulse

btp[0]	Type of board in slot J1
btp[1]	Type of board in slot J2
btp[2]	Type of board in slot J3
btp[3]	Type of board in slot J4
btp[4]	Type of board in slot J5
btp[5]	Type of board in slot J6
btp[6]	Type of board in slot J7

Valid modes: Manual.
Automatic.

7.53 Configure (35h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char btp[7]; /* I/O slot array. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

08h Program code value(s) changed

Status flags immediately cleared:

Exceptions: 0Ch Transaction authorized
54h Invalid board type

Description: This command is for chamber testing or other diagnostic purposes only and is not intended for use by an automation system.

The DanLoad 6000's database is updated and I/O points are configured using the I/O board types from the automation system.

Valid modes: Manual.
Automatic.

7.54 Weights and Measures Switch (36h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char wm; /* Weights and measures switch. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: The DanLoad 6000 responds with the current state of the Weights and Measures switch (0=Open, 1=Closed). Program codes defined as "Weights and Measures" cannot be modified when the Weights and Measures switch is closed.

Valid modes: Manual.
Automatic.

7.55 Change Operating Mode (37h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char opmode; /* Operating mode.  0=Auto, 1=Manual. */
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 27h Operating mode cannot be changed
4Dh Invalid operating mode

Description: The Change Operating Mode command allows the automation system to change the DanLoad 6000's operating mode if the auto/manual change-over input is not configured. The Change Operating Mode command is the communications equivalent of the auto/manual change-over input.

If the DanLoad 6000's operating mode is controlled by the auto/manual change-over input an exception response is generated.

If the DanLoad 6000 is already in the desired operating mode then the command is simply acknowledged.

The operating mode does not change until the transaction has ended.

Status flag 00h (Operating mode is manual) is set or cleared subsequently.

Valid modes: Manual.
Automatic.

7.56 Clear Display (38h, v2.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 01h Passcode entry in progress
15h Operating mode is manual (low priority, may not be seen!)
24h Keypad and display locked out

Description: This command clears the DanLoad 6000's multi-line display. The DanLoad 6000 displays the date and time in the upper right hand corner and "Please wait" on the bottom line.

This command is command can be used to erase the recipe menu after a Prompt Recipe command, the additive menu after a Prompt Additive command or the loading screen after an Authorize Transaction command, a Prompt Preset Volume command or an Authorized Batch command.

Valid modes: Automatic.

7.57 Request Stored Transaction (39h, v3.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int trseq;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char transaction_record[trrecl]; /* trrecl ∈ [0, 250] */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 1Dh Transaction not on file (transaction storage)

Description: This command is used to request a transaction record from the (transaction storage) transaction file. The format of the transaction record is configurable.

Valid transaction sequence numbers are 0 through 9999. In practice, fewer than 10,000 transactions may actually be stored.

Valid modes: Manual.
Automatic.

7.58 Request Stored Batch (3Ah, v3.00 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int baseq;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char batch_record[barecl]; /* barecl ∈ [0, 250] */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 1Eh Batch not on file (transaction storage)

Description: This command is used to request a batch record from the (transaction storage) batch file. The format of the batch record is configurable.

Valid batch sequence numbers are 0 through 9999. In practice, fewer than 10,000 batches may actually be stored.

Valid modes: Manual.
Automatic.

7.59 Enhanced Start Communications (3Bh, v5.11 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    int nummtrs; /* Configured number of meters. */
    int numcomps; /* Configured number of components. */
    int numvalves; /* Configured number of valves. */
    int numfacs; /* Configured number of factors/component. */
    int numrecipes; /* Number of recipes configured. */
    int numadds; /* Number of additives configured. */
    unsigned char tempunits; /* 0=Celsius.  1=Fahrenheit. */
    /* NOTE: Same as command code 21h as far as here. */
    unsigned char numdataitems; /* Number of data items, 0 to 5. */
    unsigned char language; /* Language. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: This is an alternative to the Start Communications (21h) command. This command has the following advantages over the Start Communications (21h) command:

- It has a fixed length response.
- Its response contains all values necessary for an automation system to predict the lengths of all other responses from the DanLoad 6000.
- Its response informs the automation system of the configured language.

This command has the following disadvantages over the Start Communications (21h) command:

- It is available only on DanLoad 6000 v5.11 and above; its use does not allow an automation system to be backward compatible with earlier DanLoad 6000 versions.

When the DanLoad 6000 is powered-up/reset, exits program mode or has detected an automation system communications failure, communications can be started with this command.

The DanLoad 6000 will not respond to any command from the automation system until communications have been started.

If communications are already started when this command is received then the communications link is reinitialized but operations at the DanLoad 6000 are unaffected.

When a communications failure occurs on a communications channel then communications on that channel can be started using this command.

The alternating function code sequence is reset. Either function code (41h or 42h) can be used for this query. The next (non-retry) query from the automation system should use the other function code.

The query is six characters long; address (one character), function code (one character), data field length (one character), command code (one character) and checksum (two characters). Valid Enhanced Start Communications commands for a DanLoad 6000 with communications address 01h are:

01 41 02 3B 11 7F

and:

01 42 02 3B E1 7F

The *language* can be used by the automation system to select an appropriate "message file" for sending text to the DanLoad 6000. (NOTE: The language can be modified by the automation system prior to authorizing a transaction on the DanLoad 6000.)

Valid modes: Manual.
Automatic.

7.60 Enhanced Request Status (3Ch, v5.11 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned long status; /* Status bits 31 thru 0. */
    unsigned char side; /* Swing-arm side (0->2). */
    long grsvol; /* Gross batch volume whole units. */
    long netvol; /* Net batch volume whole units. */
    unsigned char safety; /* Safety circuits status, including those with
alarm action "Off" for DanLoad 6000 v5.11 and above. */
     unsigned char almcd; /* Alarm code of oldest active primary alarm.
See §8.3. */
    unsigned char alarm_byte_9; /* Alarm bits 79 thru 72. */
    unsigned char alarm_byte_8; /* Alarm bits 71 thru 64. */
    unsigned char alarm_byte_7; /* Alarm bits 63 thru 56. */
    unsigned char alarm_byte_6; /* Alarm bits 55 thru 48. */
    unsigned char alarm_byte_5; /* Alarm bits 47 thru 40. */
    unsigned char alarm_byte_4; /* Alarm bits 39 thru 32. */
    unsigned char alarm_byte_3; /* Alarm bits 31 thru 24. */
    unsigned char alarm_byte_2; /* Alarm bits 23 thru 16. */
    unsigned char alarm_byte_1; /* Alarm bits 15 thru 8. */
    unsigned char alarm_byte_0; /* Alarm bits 7 thru 0. */
    /* NOTE: Same as command code 21h as far as here. */
     int recipenumber; /* Recipe number (1 to 30) entered or authorized.
*/
    unsigned char addsel; /* Additive selection bit map.  Re. §8.5. */
    long preset; /* Preset volume entered or authorized. */
    unsigned char spare[8]; /* Not used yet! */
    /* NOTE: Same as command code 1Eh from here on. */
    int oldest_trseq; /* Oldest transaction seq # stored (0-9999). */
    int numtr; /* Number of transactions stored (0-10,000). */
     int maxnumtr; /* Maximum number of transactions that can be stored.
*/
    unsigned char trcfgerr; /* Transaction configuration error. */
    int oldest_baseq; /* Oldest batch seq # stored (0-9999). */
    int numba; /* Number of batches stored (0->10,000). */
    int maxnumba; /* Maximum number of batches that can be stored. */
    unsigned char bacfgerr; /* Batch configuration error. */
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions:

Description: This is an alternative to the Request Status (12h) command. This command has the following advantages over the Request Status (12h) command:

- Its response contains the recipe number, additive selection and preset volume. Request Selected Recipe, Request Selected Additives and Request Preset Volume commands may no longer be required; the relevant data are available to the automation system immediately when the corresponding status bits change. Response times are improved.
- Its response contains the transaction storage status. Request Transaction Storage Status commands may no longer be required. This is particulary useful for "ticket printing" systems that upload batch and transaction data by watching for new records being written into the files.

This command has the following disadvantages over the Request Status (21h) command:

- Its response time is slightly slower, but not significantly.
- Since there are no exceptions, it does not protect the automation system from making mistakes, e.g. using the recipe number when no recipe is selected and no transaction is authorized.
- It is available only on DanLoad 6000 v5.11 and above; its use does not allow an automation system to be backward compatible with earlier DanLoad 6000 versions.

status is a binary representation of the DanLoad 6000's 32 status flags.

safety is a binary representation of the current logical state ("open" or "closed") of the DanLoad 6000's eight general-purpose safety circuits (program codes 345 through 352). The least significant bit is safety circuit 1, etc. Safety circuits that are not configured on the DanLoad 6000 are indicated to be "open", i.e. the corresponding bit is 0.

alarm_byte_0 through *alarm_byte_9* indicate active (raised and not reset) primary alarms. The automation system can maintain a copy

of the "alarm bytes" so that it can determine when primary alarms are raised and cleared for logging purposes.

The *recipe number* is valid while the recipe selected or the transaction authorized status bits are set. (The recipe number authorized for the transaction need not be the same as the recipe number selected by the operator.)

The *additive selection* is valid while the additives selected status bit is set.

The preset volume is valid while the preset volume entered or batch authorized status bits are set. For the former, it is the preset volume entered by the operator. For the latter, it is the preset volume authorized for the batch. (The preset volume authorized for the batch need not be the same as the preset volume entered by the operator.)

Valid modes: Manual.
Automatic.

7.61 Report Alarm (3Dh, v5.60 and above)

Query data:

```
struct Tasq {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
    unsigned char almcd; /* Alarm code of alarm to be raised.  See §8.3.
*/
};
```

Response data:

```
struct Tasr {
    unsigned char dfl; /* Data field length. */
    unsigned char cmdcode;
};
```

Status flags immediately set:

Status flags immediately cleared:

Exceptions: 55h Invalid bit #, i.e. the alarm code in the query is invalid.

Description: This command is used to raise a DanLoad 6000 alarm (typically with a primarly alarm action) via communications. (The term "report alarm" is used since internal to the DanLoad 6000 an alarm must be "reported" in order to be "raised".)

Possible uses of the Report Alarm command include:

- Raising an alarm based on conditions that are either not input to or detectable by the DanLoad 6000, e.g. a fire detection system or a PLC being used to a monitor a rail car's position.
- Stopping a batch (possibly temporarily) with a situation-specific message. In this case, the automation system might configure a safety circuit alarm message via communications and then raise the corresponding safety circuit alarm. The automation system can reset the alarm using the Reset Primary Alarms command.

It is emphasized that an automation system should not depend on DanLoad 6000 alarms to prevent or control hazardous or life-threatening conditions.

Valid modes: Manual.
Automatic.

8 Codes and Flags

8.1 Status Flags

The Request Status command requests the DanLoad 6000's 32 status flags as an unsigned long, i.e. binary data. The status flags indicate the operational state of the DanLoad 6000. When the DanLoad 6000 is in automatic, the automation system influences the operational state of the DanLoad 6000. However, status flags are updated by the DanLoad 6000 in both automatic and manual.

Bits marked "*" are cleared on power-up/reset. Bits marked "#" are cleared by a Start Communications command. Bits marked "α" can be cleared using the Clear Status command. Bits marked "ß" indicated statuses which prevent the automation system from displaying messages on the DanLoad 6000. All flags are cleared on a "cold start".

Bit (hex)	Interpretation
00hß	Operating mode is manual
01hß	Primary alarm
02h*	Passcode entry in progress
03h*#α	Operation timed out
04h*#α	Recipe selected
05h*#α	Additives selected, i.e. additive selection complete
06h*#α	Preset volume entered
07h*#α	Keypad data available
08hα	Program code value(s) changed
09h*	Transaction in progress
0Ah*ß	Batch in progress (doesn't necessarily imply delivery in progress)
0Bh*α	Key pressed (DanLoad 6000 v5.11 and above)
0Chα	Transaction ended
0Dhα	Batch ended
0Eh*α	Batch aborted (not started)
0Fh*α	Intermediate level input alarm stopped batch (v5.50 and above)
10h	RESERVED
11h	Batch authorized
12h	Transaction authorized
13hα	Transaction end requested ("STOP" key in auto)
14h	Keypad and display locked out to automation system
15h	Batch stopped (resumable)
16hα	Program mode
17h*	Flowing, i.e. batch in progress and not stopped
18h	RESERVED
19h	RESERVED

1Ah	RESERVED
1Bh	RESERVED
1Ch	RESERVED
1Dh	RESERVED
1Eh	RESERVED
1Fh	RESERVED

The DanLoad 6000's status flags are set and cleared as follows.

Note

"Receipt of a TAS command" assumes that a valid TAS command was received, i.e. the DanLoad 6000 did NOT respond with an exception code.

Note

⌂ indicates that the status flag is set or cleared after processing occurs that is caused by the listed condition, i.e. there may be a delay between the occurrence of the condition and the status flag changing.

00h Operating mode is manual

SET:

-On power-up when the operating mode (program code 025) is "Manual".

⌂-DanLoad 6000 switches to manual mode via user command at keypad or via configured "Auto/manual change-over" input when no load is in progress or via TAS Set Program Code Value command

CLEARED:

-On power-up when the operating mode (program code 025) is "Automatic".

⌂-DanLoad 6000 switches to automatic mode or to stand-alone mode at keypad or via configured "Auto/manual change-over" input or via TAS Set Program Code Value command

01h Primary alarm

SET:

⌂-Primary alarm occurs on the DanLoad 6000

CLEARED:

-Receipt of a Reset Primary Alarms command which clears the only active primary alarm

-the only active primary alarm is cleared by user via keypad or via configured primary alarm reset switch

02h Passcode entry in progress

SET:

-ALT+ENTER pressed at keypad from load mode

CLEARED:

-User completes passcode entry from the keypad.

-Power-up of unit

03h Operation timed out

SET:

-in automatic mode, user times-out after receipt of a Prompt Recipe, Prompt Additives, Prompt Preset Volume or Display Message command.

-in manual mode, user times-out at the Recipe, Additives, Preset Volume, or data item prompts

⌂-following receipt of Time-Out Operation command while DanLoad 6000 is prompting for a recipe, additives, a preset volume, or Display Message (not necessarily immediately) or Authorize Batch.

CLEARED:

-receipt of Prompt Recipe, Prompt Additives, Prompt Preset Volume, or Display Message, or Authorize Batch command in automatic.

-when the DanLoad 6000 initiates any prompting operation in manual.

-receipt of a Clear Status command with data bit 03h set in the automatic operating mode.

-change of operation mode between manual and automatic

-power-up of DanLoad 6000

-receipt of Start Communications command.

04h Recipe selected

SET:

-user selects a recipe via the keypad

CLEARED:

-receipt of Authorize Transaction command in automatic.

-when a transaction is authorized in manual.

-receipt of Prompt Recipe command in automatic or when a recipe is prompted for in manual.

-receipt of a Clear Status command with data bit 04h set in the automatic operating mode.

-change of operation mode between manual and automatic

-power-up of DanLoad 6000
-receipt of Start Communications command.

05h Additives selected

SET:

-user selects additives via the keypad

CLEARED:

-receipt of Authorize Transaction command in automatic or when a transaction is authorized in manual.
-receipt of Prompt Additives command in automatic or when an additive selection is prompted for in manual.
-receipt of a Clear Status command with data bit 05h set in the automatic operating mode.
-change of operation mode between manual and automatic
-power-up of DanLoad 6000
-receipt of Start Communications command.

06h Preset volume entered

SET:

-user presses ENTER key after entering valid preset volume

CLEARED:

-receipt of Prompt Preset Volume command in automatic or when a preset volume is prompted for in manual.
-receipt of Authorize Batch command in automatic or when a batch is authorized in manual.
-receipt of Clear Status command with data bit 06h set in the automatic operating mode.
-change of operation mode between manual and automatic
-power-up of DanLoad 6000
-receipt of Start Communications command.

07h Keypad data available

SET:

-user presses ENTER key after entering data following receipt of a Display Message command with pmptfw > 0

CLEARED:

-receipt of a Clear Status command with data bit 07h set in the automatic operating mode.
-receipt of a Display Message command
-change of operation mode between manual and automatic

-power-up of DanLoad 6000
-receipt of Start Communications command.

08h Program code values changed

SET:

-DanLoad 6000 switches between automatic and manual via the Change Operating Mode command or the auto/manual change-over switch with the operating mode (program code 025) not read-only.
-receipt of a Set Program Code Value command which changes a program code value
-receipt of a Configure Recipe command
-receipt of a Configure command
-user changes a program code value in Setup mode via the keypad
-when a new transaction or a new batch is started and the transaction sequence number of batch sequence number are not read-only.

CLEARED:

-receipt of a Clear Status command with data bit 08h set in the automatic operating mode.
-receipt of a Clear Value Changed Attributes command

09h Transaction in progress

SET:

-first batch is started via START key or Start Batch command after a transaction has been authorized

CLEARED:

-when the transaction ended status flag (0Ch) is set.
-power-up of DanLoad 6000

0Ah Batch in progress

SET:

-receipt of Start Batch command
-user presses START key after batch has been authorized

CLEARED:

-Batch ended bit is set
-power-up of DanLoad 6000
-the operating mode change to/from manual/automatic.

0Bh Key Pressed (DanLoad 6000 v5.11 and above)

SET:

-when a key (except the "ALT" key by itself) is pressed.

CLEARED:

-power-up/reset of DanLoad 6000.

-receipt of a Clear Status command with data bit 0Bh set in the automatic operating mode.

0Ch Transaction ended

SET:

-when an operating mode change to/from automatic/manual forces a transaction to end.

-when the DanLoad 6000 is powered-up and a transaction was in progress when the DanLoad 6000 was powered-down.

⌂-following receipt of End Transaction command and completion of end transaction processing.

⌂-in manual mode, any action which ends a transaction

CLEARED:

-

-receipt of Clear Status command with data bit 0Ch set in the automatic operating mode.

-on receipt of Authorized Transaction command in automatic or when a transaction is authorized in manual.

0Dh Batch ended

SET:

⌂-receipt of End Batch command when a batch is in progress and the batch is stopped

⌂-preset volume has been delivered under "normal" circumstances

⌂-receipt of Stop Batch command or user presses STOP key when a batch is in progress and the batch becomes non-restartable

⌂-primary or secondary alarm occurs while a batch is in progress and the batch is stopped and becomes non-restartable

⌂-when an operating mode change to/from manual/automatic forces a batch to end.

-when the DanLoad 6000 is powered-up and a batch was in progress when the DanLoad 6000 was powered-down.

⌂-Timeout of a secondary alarm when a batch is in progress

-STOP key pressed when there is a stopped batch in progress

CLEARED:

-receipt of Authorize Batch command in automatic or when a batch is authorized in manual.

-receipt of Clear Status command with data bit 0Dh set in the automatic operating mode.

0Eh Batch aborted (not started)

SET:

-in automatic mode, user presses STOP key after an Authorize Batch command instead of the START key or Start Batch command

-in manual mode, user presses STOP key after entering a preset volume instead of the START key

-when the DanLoad 6000 is powered-down/up while a batch is authorized but not in progress.

CLEARED:

-receipt of Clear Status command with data bit 0Eh set in the automatic operating mode.

-receipt of Authorize Batch command in automatic or when a batch is authorized in manual.

-when the DanLoad 6000 is powered-up, unless there was a batch authorized but not in progress when the DanLoad 6000 was powered-down.

0Fh Intermediate level input alarm stopped batch

SET:

- When a batch is stopped due to an intermediate level input alarm. (The batch may also be ended.)

CLEARED:

- Power-up/reset of DanLoad 6000.
- Batch started or restarted.
- Receipt of Clear Status command with data bit 0Fh set in the automatic operating mode.

10h RESERVED

11h Batch authorized

SET:

-receipt of Authorize Batch command in automatic or when a batch is authorized in manual.

CLEARED:

-Batch ended bit is set

-when the batch aborted status flag is set.

-power up of DanLoad 6000

12h Transaction authorized

SET:

⌂-following receipt of Authorize Transaction command (not necessarily immediately)

CLEARED:

-power up of DanLoad 6000

-when the transaction ended status flag is set.

-when there is an operating mode change when the transaction authorized status flag is set.

⌂-when an End Transaction command is received while the transaction authorized status flag is set but there is not a transaction in progress in automatic.

-when the operator presses the "STOP" key when a transaction is authorized but there is no transaction in progress in manual.

13h Transaction end requested (stop key in auto)

SET:

-in automatic mode, STOP key pressed by user when a transaction is authorized and there is no batch in progress or a batch is stopped.

CLEARED:

-receipt of Authorize Transaction command

-receipt of Clear Status command with data bit 13h set in the automatic operating mode.

14h Keypad and display locked out to the automation system

SET:

-receipt of Prompt Recipe, Prompt Additives, Prompt Preset Volume, or Display Message command in automatic.

-receipt of Authorize Batch command

-while an Authorize Transaction command is being processed (cleared at end of processing)

CLEARED:

-user selects recipe following receipt of Prompt Recipe command

-user selects additives following receipt of Prompt Additives command

-user selects preset volume following receipt of Prompt Preset Volume command

-user inputs response following receipt of a Display Message command

-when the Batch ended or Batch aborted flag(s) are set.

-when the Operation Timed out bit is set while the keypad/display is locked out.

⌂-when a primary alarm occurs which ends a Prompt Recipe, Prompt Additive, Prompt Preset Volume or Display Message in automatic.

-when the DanLoad 6000 is powered-up.

15h Batch stopped (resumable)

SET:

⌂-receipt of Stop Batch command when a batch is in progress and the batch becomes restartable

⌂-user presses STOP key when a batch is in progress and the batch becomes restartable

⌂-primary or secondary alarm occurs when a batch is in progress, and the batch becomes restartable

-(This flag is set only if batch in progress is set.)

CLEARED:

-receipt of Start Batch command or when the operator presses "START" after an Authorize Batch command.

-when the operator presses the "START" key to restart a stopped batch.

-when the batch ended status flag is set.

-power up of DanLoad 6000

16h Program Mode

SET:

-when the operator presses "ENTER" after typing a valid passcode when attempting to enter program mode.

CLEARED:

-receipt of a Clear Status command with data bit 16h set in the automatic operating mode.

17h Flowing

SET:

-when a batch delivery begins, i.e. flow control valve is opened.

CLEARED:

-on receipt of an Authorized Transaction command in automatic or when a transaction is authorized in manual.

-when the batch stopped or batch ended status flags are set.

-when the DanLoad 6000 is powered-up.

18h RESERVED
19h RESERVED
1Ah RESERVED
1Bh RESERVED
1Ch RESERVED
1Dh RESERVED
1Eh RESERVED
1Fh RESERVED

8.2 DanLoad 6000 Exception Response Codes

The Modbus function code for normal queries and responses are 41h and 42h. (The Modbus protocol reserves function codes 65 through 72 for user-defined "custom functions".) The DanLoad 6000 uses function codes C1h and C2h to indicate an "exception response", i.e. invalid data field, in which case the response's data field contains a DanLoad 6000 Exception Response Code. (These are not the standard Modbus exception response codes.)

Code (hex)	Interpretation
00h	Invalid command code
01h	Passcode entry in progress
02h	No transaction ended
03h	Response's data field too long
04h	Program code value is Weights and Measures
05h	RESERVED
06h	No batch in progress
07h	No transaction in progress
08h	Batch in progress
09h	Transaction in progress
0Ah	Primary alarm active
0Bh	Batch authorized
0Ch	Transaction authorized
0Dh	RESERVED
0Eh	No keypad data available
0Fh	Component not available
10h	Additive not available
11h	Program code value is read only
12h	Status not set or cannot be reset
13h	No additives configured
14h	No batch authorized
15h	Operating mode is manual
16h	No preset volume entered
17h	No recipe selected
18h	No additive selection made
19h	Data items not entered
1Ah	No key pressed
1Bh	Diagnostic not started
1Ch	Diagnostic running
1Dh	Transaction not on file (transaction storage)
1Eh	Batch not on file (transaction storage)

1Fh	RESERVED
20h	Number of recipes < 2 **(Not DanLoad 6000 v5.00 and above)**
21h	RESERVED
22h	No transaction authorized
23h	RESERVED
24h	Keypad and display locked out
25h	No batch stopped
26h	No batch ended
27h	Operating mode cannot be changed
28h	RESERVED
29h	RESERVED
2Ah	RESERVED
2Bh	RESERVED
2Ch	RESERVED
2Dh	RESERVED
2Eh	RESERVED
2Fh	RESERVED
30h	RESERVED
31h	RESERVED
32h	RESERVED
33h	RESERVED
34h	RESERVED
35h	RESERVED
36h	RESERVED
37h	RESERVED
38h	RESERVED
39h	RESERVED
3Ah	RESERVED
3Bh	RESERVED
3Ch	RESERVED
3Dh	RESERVED
3Eh	RESERVED
3Fh	RESERVED
40h	Invalid recipe number
41h	Invalid meter number
42h	Invalid component number
43h	Invalid transaction sequence number
44h	Invalid program code
45h	Invalid program code value
46h	Invalid CPU number
47h	Invalid number of components

48h	Invalid number of data items
49h	Invalid swing-arm side
4Ah	Invalid I/O point type
4Bh	Invalid I/O point number
4Ch	Invalid output value
4Dh	Invalid operating mode
4Eh	Invalid additive selection method
4Fh	Invalid preset volume
50h	Invalid date
51h	Invalid time
52h	Invalid data code
53h	Invalid override maximum preset volume
54h	Invalid board type
55h	Invalid bit #

8.3 Alarm Flags

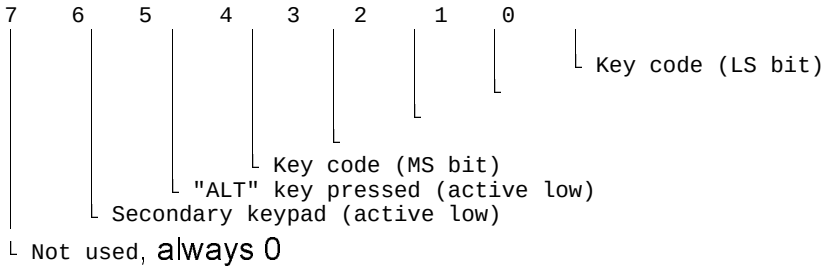
The alarm code is the alarm bit plus one.

Bit	Interpretation	Action
00	Primary display failure	Primary
01	Secondary display failure	
02	Comms failure channel A	Configurable
03	Comms failure channel B	
04	Unable to maintain blend	Primary
05	Flow rate too low meter 1	Configurable
06	Flow rate too low meter 2	
07	Flow rate too low meter 3	
08	Flow rate too low meter 4	
09	Flow rate too high meter 1	Configurable
10	Flow rate too high meter 2	
11	Flow rate too high meter 3	
12	Flow rate too high meter 4	
13	Valve closed early meter 1	Configurable
14	Valve closed early meter 2	
15	Valve closed early meter 3	
16	Valve closed early meter 4	
17	Time-out - no flow detected meter 1	Configurable
18	Time-out - no flow detected meter 2	
19	Time-out - no flow detected meter 3	
20	Time-out - no flow detected meter 4	
21	Unauthorized flow exceeds limit meter 1	Primary
22	Unauthorized flow exceeds limit meter 2	
23	Unauthorized flow exceeds limit meter 3	
24	Unauthorized flow exceeds limit meter 4	
25	Pulse security error meter 1	Primary
26	Pulse security error meter 2	
27	Pulse security error meter 3	
28	Pulse security error meter 4	
29	Temperature failure meter 1	Configurable
30	Temperature failure meter 2	
31	Temperature failure meter 3	
32	Temperature failure meter 4	
33	Pressure failure meter 1	Primary
34	Pressure failure meter 2	
35	Pressure failure meter 3	
36	Pressure failure meter 4	
37	Unable to close valve meter 1	Primary
38	Unable to close valve meter 2	
39	Unable to close valve meter 3	
40	Unable to close valve meter 4	
41	Density failure component 1	Configurable
42	Density failure component 2	
43	Density failure component 3	
44	Density failure component 4	
45	Component 1 block valve not closed	Primary
46	Component 2 block valve not closed	
47	Component 3 block valve not closed	
48	Component 4 block valve not closed	
49	Additive 1 failure	Primary
50	Additive 2 failure	
51	Additive 3 failure	
52	Additive 4 failure	
53	Additive 5 failure	
54	Additive 6 failure	

55	Safety circuit 1	Configurable
56	Safety circuit 2	Configurable
57	Safety circuit 3	Configurable
58	Safety circuit 4	Configurable
59	Safety circuit 5	Configurable
60	Safety circuit 6	Configurable
61	Safety circuit 7	Configurable
62	Safety circuit 8	Configurable
63	Data logging memory full	Configurable
64	Memory check failure	Primary
65	Storage memory full	Configurable
66	Power failure	Configurable
67	Unable to ramp down	Configurable
68	MPMC 1 failure	Primary
69	MPMC 2 failure	Primary
70	Calibration failure meter 1	Primary
71	Calibration failure meter 2	Primary
72	Calibration failure meter 3	Primary
73	Calibration failure meter 4	Primary
74	Intermediate level input	Secondary
75	RESERVED	
76	RESERVED	
77	RESERVED	
78	RESERVED	
79	RESERVED	

8.4 Key Codes

There is a key code for each of the thirty four keys, i.e. seventeen keys each with an "ALT" function. (See table below.) The key codes correspond to the six least significant bits (bits 0 through 5) of the DanLoad 6000's internal key codes and, therefore, do not indicate if the key was pressed at the primary or at the secondary keypad.



Note

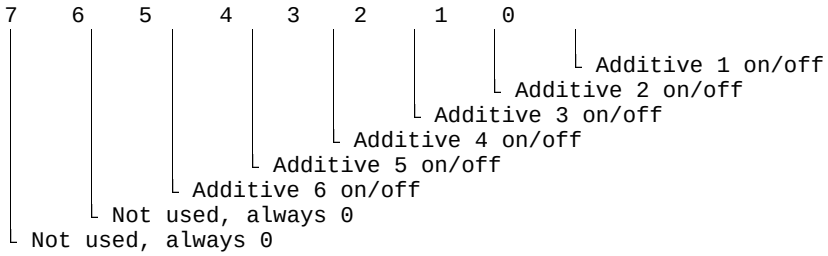
Seven bits (6=most significant to 0=least significant) are used. Bits 4 to 0 indicate which key. Bit 5 indicates "ALT" key active (active low). Bit 6 indicates secondary keypad (active low).

Key code (hex)	Key
20	"0"
21	"1"
22	"2"
23	"3"
24	"4"
25	"5"
26	"6"
27	"7"
28	"8"
29	"9"
2A	"CLEAR"
2B	"SELECT"
2C	"↑"
2D	"↓"
2E	"ENTER"
2F	"STOP"
30	"START"

00	"ALT"+"0"
01	"ALT"+"1"
02	"ALT"+"2"
03	"ALT"+"3"
04	"ALT"+"4"
05	"ALT"+"5"
06	"ALT"+"6"
07	"ALT"+"7"
08	"ALT"+"8"
09	"ALT"+"9"
0A	"ALT"+"CLEAR"
0B	"ALT"+"SELECT"
0C	"ALT"+"↑"
0D	"ALT"+"↓"
0E	"ALT"+"ENTER"
0F	"ALT"+"STOP"
10	"ALT"+"START"

8.5 Additive Selection Codes

The additive selection code is a binary representation of the DanLoad 6000's additive selection byte.



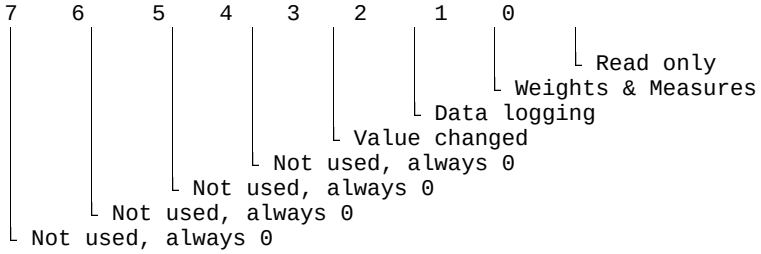
Code (hex) Interpretation

	Additive #					
	1	2	3	4	5	6
00	Off	Off	Off	Off	Off	Off
01	On	Off	Off	Off	Off	Off
02	Off	On	Off	Off	Off	Off
03	On	On	Off	Off	Off	Off
04	Off	Off	On	Off	Off	Off
05	On	Off	On	Off	Off	Off
06	Off	On	On	Off	Off	Off
07	On	On	On	Off	Off	Off
08	Off	Off	Off	On	Off	Off
09	On	Off	Off	On	Off	Off
0A	Off	On	Off	On	Off	Off
0B	On	On	Off	On	Off	Off
0C	Off	Off	On	On	Off	Off
0D	On	Off	On	On	Off	Off
0E	Off	On	On	On	Off	Off
0F	On	On	On	On	Off	Off
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
1A						
1B						
1C						
1D						
1E						
1F						
20						
21						
22						

```
23
24
25
26
27
28
29
2A
2B
2C
2D
2E
2F
.
.
.
3E      Off On  On  On  On  On
3F      On  On  On  On  On  On
```

8.6 Program Code Attributes Code

The program code attributes code is a representation of the DanLoad 6000's program code attributes byte, which is explained in detail in the section on Program Code Attributes.



Code (hex)	Interpretation
00	No attributes set
01	*Read only*
02	*Weights and Measures*
03	*Weights and Measures* and *Read only*
04	*Data logging*
05	*Data logging* and *Read only*
.	
.	
.	
0F	*Value changed*, *Data logging*, *Weights and Measures* and *Read only*

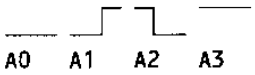
8.7 Valid Characters in Alphanumeric Strings

The unshaded characters in the following table can be displayed on the DanLoad 6000's display using the Display Message (1Ch) command. Certain "special characters" cannot be represented in WordPerfect 5.1 so...

- a character followed by "-" is used to represent that character with a line through it, e.g. character code ED (O-),
- a character followed by "/" is used to represent that character with an acute accent, e.g. character code F1 (c/),
- a character followed by "^{V}" (superscript "V") is used to represent that character with an upside down circumflex, e.g. character code F2 (c^{V}),
- a character followed by "$^{+}$" (superscript "+") is used to represent that character with a "horn", and
- a character followed by "$_{o}$" (subscript "o") is used to represent that character with a dot below it.

Any invalid characters in a message to be displayed are "filtered out" and displayed as asterisks by the DanLoad 6000.

Characters codes A0, A1, A2 and A3 are the "waveform drawing" characters; "low", "low to high", "high to low" and "high", i.e.



Characters codes CC to CF are reserved for a large character (double-sized) "space".

HI / LO	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	`	p	Ç	É		3	7			L-
1			!	1	A	Q	a	q	ü	æ		3	7			c/
2			"	2	B	R	b	r	é	Æ		3	7			c^{v}
3			#	3	C	S	c	s	â	ô		3	7			d-
4			$	4	D	T	d	t	ä	ö	0	4	8		↑	s^{v}
5			%	5	E	U	e	u	à	ò	0	4	8		↓	z^{v}
6			&	6	F	V	f	v	å	û	0	4	8		á	D-
7			'	7	G	W	g	w	ç	ù	0	4	8			o^{+}
8			(	8	H	X	h	x	ê	ij	1	5	9		í	u^{+}
9			)	9	I	Y	i	y	ë	Ö	1	5	9		ó	a_{o}
A			*	:	J	Z	j	z	è	Ü	1	5	9		ú	o^{+}_{o}
B			+	;	K	[	k	{	ï	¢	1	5	9		ñ	
C			,	<	L	\	l	|	î	£	2	6	_		o-	
D			-	=	M	]	m	}	ì	¥	2	6	_		O-	
E			.	>	N	^	n	~	Ä	₧	2	6	_		I-	FE
F			/	?	O	_	o		Å	ƒ	2	6	_		s/	FF

In DanLoad 6000 v5.00 and above, characters codes E6 to FD are defined as follows when the configured language (program code 28) is "Greek".

Character code (hex)	Greek character
E6	Alpha
E8	Gamma

Character code (hex)	Greek character
E9	Delta
ED	Theta
F0	Lambda
F3	Xi
F5	Pi
F7	Sigma
FA	Phi
FC	Psi
FD	Omega

The complete Greek alphabet is defined by the following character codes.

Greek character	Character code (hex)
Alpha	E6
Beta	42
Gamma	E8
Delta	E9
Epsilon	45
Zeta	5A
Eta	48
Theta	ED
Iota	49
Kappa	4B
Lambda	F0
Mu	4D
Nu	4E
Xi	F3
Omicron	4F
Pi	F5
Rho	50
Sigma	F7
Tau	54
Upsilon	59
Phi	FA
Chi	58

Greek character	Character code (hex)
Psi	FC
Omega	FD

When the language is "Russian", the complete Russian (Cyrillic) alphabet is defined by the following characters codes.

Transcribed name of Russian character	Russian character looks like... (* = the usual Roman alphabetic character is used)	Character code (hex)
ah	"A"*	41
beh	"b" pointing right	E5
veh	"B"*	42
geh	Greek gamma	E8
deh	Box with legs	E7
yeh	"E"*	45
yoh	"E" umlaut	E9
zheh	"X" with vertical line through it	EA
zeh	Special "3"	EB
ee	Backwards "N"	EC
ee KRAHT-kuh-yuh	Backwards "N" with line above	ED
kah	"K"*	4B
ehl	Greek pi with foot ("tap dancer")	EE
ehm	"M"*	4D
ehn	"H"*	48
oh	"O"*	4F
peh	Greek pi	F5
ehr	"P"*	50

Transcribed name of Russian character	Russian character looks like... (* = the usual Roman alphabetic character is used)	Character code (hex)
ehs	"C"*	43
teh	"T"*	54
oo	Special "Y"	EF
ehf	Greek phi	FA
khah	"X"*	58
tseh	Square "U" with tail	F0
cheh	Backwards Greek mu	F1
shah	Square "W"	F2
shhah	Square "W" with tail	F3
TV^{y}OHR-dïee znakh	"b" pointing left	F4
yih-RIE	"bl"	F6
M^{y}AHKH-k^{y}ee znakh	Special "b"	F7
eh ah-bah-ROHT-nuh-yuh	Backwards epsilon	F8
yoo	"I-O"	F9
yaa	Backwards "R"	FB

In the Thai DanLoad 6000, character codes FE and FF are reserved for "script on" and "script off" control respectively, so that Roman alphabetic characters and Thai "script" characters can be mixed in a message.

9 Connection to an Automation System

9.1 DUART Pin-Outs

The DanLoad 6000 is connected to an automation system via its DUART. The DUART is manufactured with either a) one RS-232 and one RS-485 ports, or b) two RS-485 ports. For the v1 CPU board, the DUART is a "daughter board" installed on top of the CPU board. For the v2 CPU board, the DUART is built-in, i.e. integral to the CPU board. For DanLoad 6000 v5.30 and above, program code 662 must be used to configure the unit's communications address; the DUART's address switch is no longer used. The DUART board's address switches should be set to something other than zero. NOTE: For RS-485 interfaces, some vendors use the symbol "B" instead of "+" and "A" instead of "-".

RS-485/RS-232 DUART			
Channel A (RS-485)		Channel B (RS-232)	
Pin	Usage	Pin	Usage
1	RXD+/B	5	RXD
2	RXD-/A	6	CTS
3	TXD+/B	7	TXD
4	TXD-/A	10	SGND

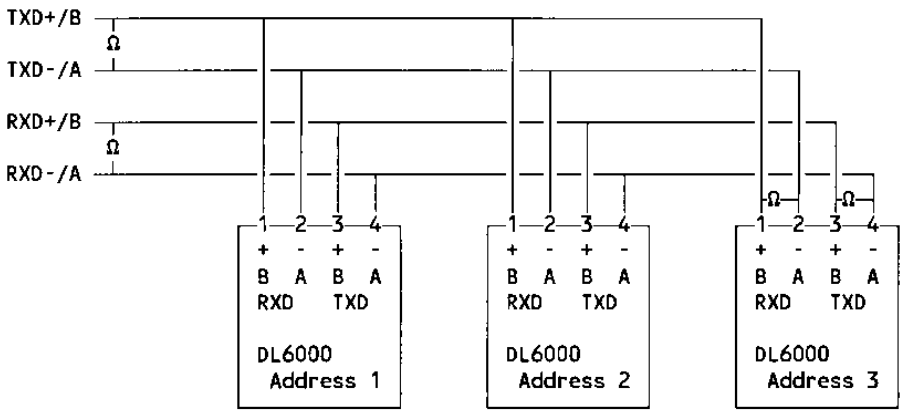
Note

To perform the "external loopback" diagnostic on channel A, pin 1 should be looped to pin 3 and pin 2 should be looped to pin 4.

CTS on channel B is used only when the channel is configured for data logging.

RS-485/RS-485 DUART			
Channel A (RS-485)		Channel B (RS-485)	
Pin	Usage	Pin	Usage
1	RXD+/B	5	RXD+/B
2	RXD-/A	6	RXD-/A
3	TXD+/B	7	TXD+/B
4	TXD-/A	8	TXD-/A

DanLoad 6000 Channel "A" RS-485 Multidrop Connection



NOTE

FOR LONG CABLE RUNS (OVER 1000 FEET), 120Ω RESISTORS SHOULD BE PLACED ACROSS TXD+/B AND TXD-/A AND ACROSS RXD+/B AND RXD-/A ON THE MASTER STATION AND ON THE LAST SLAVE STATION AS INDICATED ABOVE.

9.2 Redundant Communications Link

In some installations, it is desirable to have a "redundant communications link" to the DanLoad 6000, meaning that loading should continue even if the communications multidrop is broken (snapped, corroded, dug up) or a DanLoad 6000 communications port fails (lightning strike, electronics component failure).

A redundant communications link can be implemented by doing *all* of the following:

- Configure both DanLoad 6000's communications ports for automation system communications. NOTE: This prevents the use of "data logging" or a backup/parallel ticket printing system since the DanLoad 6000 has only two serial ports.
- Double-up the multidrop links from the automation system to the DanLoad 6000's, i.e. use two serial ports on the automation system and two multidrop cables for each "group" of DanLoad 6000's.
- Program the automation system automatically to change from communications on a "bad" multidrop (detected as no response from the DanLoad 6000) to the other multidrop. This is explained in more detail below.

The DanLoad 6000 is quite "happy" to receive automation system queries on both communications ports, it is simply unwise to try to *control* authorization via both ports simultaneously. The Start Communications command does not interfere with the DanLoad 6000's operation; it is primarily a method for synchronizing the alternating function codes (41h and 42h).

A workable automation system redundant communications link implementation would be as follows. The automation system would (initially) Start Communications with both DanLoad 6000 ports (channel "A" and channel "B"), and would then proceed to control the DanLoad 6000 via one port while sending an occasional "poll" command (such as Request Status) on the other port (to let the automation system know that the multidrop link and DanLoad 6000 port are operational, and to let the DanLoad 6000 know that the automation system is "alive"). If the automation system detects a communications failure on the control multidrop, it can simply switch control to the other multidrop and send the command with the appropriate (synchronization) function code (41h or 42h) to the DanLoad 6000. This "multidrop change-over" would be entirely transparent to an operator at the DanLoad 6000. Obviously, the automation system should report the failed *multidrop link* or *DanLoad 6000 port* so that the fault can be diagnosed and corrected.

Automation system implementors may also wish to note that while both multidrops are available, performance improvements can be achieved by making one multidrop the primary control multidrop for some of the DanLoad 6000's, and making the other multidrop the primary control multidrop for the other DanLoad 6000's.

This page intentionally left blank.

10 Revision Information

10.1 Revision 1.1 (4 May 1993)

Many modifications following in-house review. The document should be re-read carefully!

The command required to request stored transaction data is not defined. (Transaction storage is not defined.)

10.2 Revision 1.2 (10 June 1993)

Various modifications and clarifications following review. The data fields of some commands have changed. The document should be re-read carefully!

The CRC-16 accumulator should be initially flushed with 1's.

There are additional exception response codes (starting from 40h) for invalid values in the data field.

The *numcomps*, *nummtrs*, etc. in the data fields are the configured number of components, the configured number of meters, etc.

The DanLoad 6000 does not display any particular screen by default in automatic. The DanLoad 6000 displays the loading screen when it receives the Authorized Transaction command.

The DanLoad 6000's keypad and display are locked out to the automation system during certain operations.

The Prompt Additive command include a suggested additive selection.

The Authorize Batch command and the Set Densities/Gravities command include flags indicating whether or not the backup density or gravity should be updated by the DanLoad 6000.

10.3 Revision 1.3 (13 September 1993)

The gross and net batch volumes in the Request Status response are in whole units.

Prompt Recipe (01h). Exception 09h (Transaction in progress) has been replaced by exception 0Ch (Transaction authorized).

Time-Out Operation (05h). Status flag 03h (Operation timed out) is not set immediately.

Authorize Transaction (06h). Status flag 12h (Transaction authorized) is not set immediately.

End Transaction (07h). Status flags 09h, 0Ch and 12h are not set/cleared immediately.

Authorize Batch (0Ah). Status flag 11h (Batch authorized) is not set immediately. Exception 04h (program code value is Weights and Measures) has been added.

End Batch (0Dh). Status flags 0Ah, 0Dh and 15h are not set/cleared immediately.

Batch Data by Component (10h). Average temperature and average density are *long*. Added actual %age per component (*pct100*).

Set Date and Time (29h). Exception 0Ah (Primary alarm active) has been removed.

Added exception code 10h.

Added exception code 03h.

Added exception code 04h.

Added exception code 54h.

Deleted exception code 21h. (Use exception code 16h.)

Exception code 44h is "Invalid program code". Exception code 45h is "Invalid program code value".

10.4 Revision 1.4 (24 January 1994)

The maximum query/response message length for Modbus RTU is 256 bytes (including check characters.)

Batch Data by Component (10h), Request Meter Values (19h) and Request Component Values (1Ah). The DanLoad 6000 responds with component batch and meter load gross and net volumes in whole units.

Clear Display (38h). This is a new command.

Corrections made in documentation that describes when status flags are SET and CLEARED.

Added a simple schematic showing typical RS-485 multidrop connection with terminating resistors.

10.5 Revision 1.5 (11 May 1994)

New/modified commands to support transaction storage: Request Transaction Storage Status (1Eh), Initialize Transaction Storage (20h), Request Stored Transaction (39h) and Request Stored Batch (3Ah).

New C_{tl} options: "Old 24" and "Old 54".

New alarms: "Memory check failure", "Storage memory full" and "Power failure". There is an alarm bit for each one.

The "large number" preset volume on the loading display is cleared (if it is present) when a transaction is authorized.

Included example Start Communications command messages for a DanLoad 6000 with communications address 01h.

Corrected the CRC-16 macro example to show that the accumulator should be flushed with sixteen ones, i.e. *acc = 0xffff*, rather than eight ones, i.e. *acc = 0xff*.

The "Additive I/O board" is now called the "Enhanced I/O board".

10.6 Revision 1.6 (17 October 1994)

The DanLoad 6000 software version number(s) in which each command is available are shown in §6 and in the Table of Contents.

Format of pressure values changed from XXXX.X to XXXX.XX throughout, i.e. allow from 0.00 to 9999.99 psi, kPa, etc. (Size of data is unchanged, i.e. *long*.)

New "Unable to ramp down" alarm with alarm code 68 (alarm bit 67).

Pin-outs for channel A (RS-485) external loopback diagnostic.

The "Primary internal temperature" and "Secondary internal temperature" alarms (bits 0 and 1) are now called the "Primary display failure" and "Secondary display failure" alarms. (Logic has been added to detect when a keypad/display panel has been disconnected.)

Installed Boards (34h). A new board type, i.e. 4-channel meter pulse board, can be detected. The existing meter pulse board is now called the 2-channel meter pulse board.

The "transaction storage" program code attribute is no longer used. The values to store in transaction storage memory are configured via transaction storage codes (TSC's) and batch storage code (BSC's).

The C language database definitions for DanLoad 6000 v4.00 have been included.

10.7 Revision 1.7 (17 November 1995)

Exception code 15h ("Operating mode is manual") is low priority, and may not be seen if other (higher priority) status bits (being either set or clear) generate an exception for a given command, e.g. command code 06h ("Authorize Transaction") typically generates exception code 24h ("Keypad and display locked out") in the manual operating mode.

For DanLoad 6000 v5.00 and above, the Prompt Recipe (01h) command no longer gives the number of recipes < 2 (20h) exception.

New "alarm bits" for MPMC failure alarms. (The MPMC is the "smart" daughter board for the 4-channel meter pulse board.) New "alarm bits" for calibration failure alarms. (Support for future MPMC v2.00.)

10.8 Revision 1.8 (9 July 1996)

The Enhanced Start Communications (3Bh) and Enhanced Request Status (3Ch) commands (DanLoad 6000 v5.11 and above) are described.

The Key Pressed (0Bh) status flag (DanLoad 6000 v5.11 and above), which simplifies the process of detecting a key pressed for the automation system, is described.

The Russian (Cyrillic) character codes are documented.

Various clarifications have been made based on comments from automation system programmers, e.g. cross-reference "bit maps" to §8, query and response structures must be "packed", etc. Comments have been added to indicate that net volumes may be interpreted as mass/weight (standard DanLoad 6000 v5.11 upgrade), i.e. the DanLoad 6000 uses the same storage for either one or the other. Subsection numbering has been corrected.

10.9 Revision 1.9 (5 Dec 1997)

The Report Alarm (3Dh) and command is described.

Corrected and clarified use of RAM Tests command. Clarified use of Configure Recipe command.

Clarified that query and response struct's in §7 are only "quasi" C language.

Included DanLoad 6000 v5.30 database which has support for the v2 CPU board. For DanLoad 6000 v5.30 and above, program code 662 must be used to configure the unit's communications address; the DUART's address switch is no longer used.

Included DanLoad 6000 v5.40 database.

Added information about "inverted discrete inputs" (v5.30 and above feature).

Added information about "redundant communications link" implementation.

Included "Intermediate level input" alarm (v5.50 and above). The "intermediate level input alarm stopped batch" (0Fh) status flag (DanLoad 6000 v5.50 and above), which simplifies processing an intermediate level input alarm for the automation system, is described.

10.10 Revision 2.0 (16 Mar 1998)

Described DanLoad 6000 internal processing in the case where two automation systems are connected.

Modified table of "valid characters in alphanumeric strings" to include characters added for "Vietnamese" (standard DanLoad 6000 v5.60).

Clarified meaning of the invalid program code value exception to command code 23h.

Documented problem with command code 0Ch in v2.0 thru v5.5.

10.11 Revision 2.1 (10 Sep 1998)

Updated for standard DanLoad 6000 v5.70 software development (project 6006, file EPR-00004, ECO 9497).

The Reset Unit command (command code 30h) does work in the manual operating mode. See note in §7.48.

WARRANTY CLAIM REQUIREMENTS

To make a warranty claim, you, the Purchaser, must:

1. Provide Daniel with proof of the Date of Purchase and proof of the Date of Shipment of the product in question.

2. Return the product to Daniel within twelve (12) months of the date of original shipment of the product, or within eighteen (18) months of the date of original shipment of the product to destinations outside of the United States. The Purchaser must prepay any shipping charges. In addition, the Purchaser is responsible for insuring any product shipped for return, and assumes the risk of loss of the product during shipment.

3. To obtain Warranty service or to locate the nearest Daniel office, sales, or service center call (713) 467-6000, Fax (281) 897-2901, or contact:

   Daniel Measurement and Control
   P. O. Box 55435
   Houston, Texas 77255

   When contacting Daniel for product service, the purchaser is asked to provide information as indicated on the following "Customer Problem Report".

   Daniel Measurement and Control offers both on call and contract maintenance service designed to afford single source responsibility for all its products.

   Daniel Industries, Inc. reserves the right to make changes at any time to any product to improve its design and to insure the best available product.

DANIEL INDUSTRIES, INC.
CUSTOMER PROBLEM REPORT

FOR FASTEST SERVICE, COMPLETE THIS FORM, AND RETURN IT ALONG WITH THE AFFECTED EQUIPMENT TO CUSTOMER SERVICE AT THE ADDRESS INDICATED BELOW.

COMPANY NAME:_____

TECHNICAL CONTACT:_____ PHONE:_____

REPAIR P. O. #:_____ IF WARRANTY, UNIT S/N:_____

INVOICE ADDRESS:_____

_____

_____

SHIPPING ADDRESS:_____

_____

_____

RETURN SHIPPING METHOD:_____

EQUIPMENT MODEL #:_____ S/N:_____FAILURE DATE:_____

DESCRIPTION OF PROBLEM:_____

_____

_____

WHAT WAS HAPPENING AT TIME OF FAILURE?_____

_____

ADDITIONAL COMMENTS:_____

_____

_____

REPORT PREPARED BY:_____ TITLE:_____

IF YOU REQUIRE TECHNICAL ASSISTANCE, PLEASE FAX OR WRITE THE MAIN CUSTOMER SERVICE DEPARTMENT AT:

DANIEL MEASUREMENT AND CONTROL
ATTN: CUSTOMER SERVICE
19203 HEMPSTEAD HIGHWAY
HOUSTON, TEXAS 77065

PHONE: (281) 897-2900
FAX: (281) 897-2901

The sales and service offices of Daniel Industries, Inc. are located throughout the United States and in major countries overseas. Please contact Daniel Measurement and Control at P. O. Box 55435, Houston, Texas 77255, or phone (713) 467-6000 for the location of the sales or service office nearest you. Daniel Measurement and Control offers both on-call and contract maintenance service designed to provide single-source responsibility for all Daniel Measurement and Control products.

DANIEL