User Guide

CAN

Option Module for
Unidrive

Part Number: 0460-0063
Issue Number: 2

SAFETY INFORMATION

Persons supervising and performing the electrical installation or maintenance of a Drive and/or an external Option Unit must be suitably qualified and competent in these duties. They should be given the opportunity to study and if necessary to discuss this User Guide before work is started.

The voltages present in the Drive and external Option Units are capable of inflicting a severe electric shock and may be lethal. The Stop function of the Drive does not remove dangerous voltages from the terminals of the Drive and external Option Unit. Mains supplies should be removed before any servicing work is performed.

The installation instructions should be adhered to. Any questions or doubt should be referred to the supplier of the equipment. It is the responsibility of the owner or user to ensure that the installation of the Drive and external Option Unit, and the way in which they are operated and maintained complies with the requirements of the Health and Safety at Work Act in the United Kingdom and applicable legislation and regulations and codes of practice in the UK or elsewhere.

The Drive software may incorporate an optional Auto-start facility. In order to prevent the risk of injury to personnel working on or near the motor or its driven equipment and to prevent potential damage to equipment, users and operators, all necessary precautions must be taken if operating the Drive in this mode.

The Stop and Start inputs of the Drive should not be relied upon to ensure safety of personnel. If a safety hazard could exist from unexpected starting of the Drive, an interlock should be installed to prevent the motor being inadvertently started.

GENERAL INFORMATION

The manufacturer accepts no liability for any consequences resulting from inappropriate, negligent or incorrect installation or adjustment of the optional operating parameters of the equipment or from mismatching the Drive with the motor.

The contents of this User Guide are believed to be correct at the time of printing. In the interests of a commitment to a policy of continuous development and improvement, the manufacturer reserves the right to change the specification of the product or its performance, or the contents of the User Guide, without notice.

Copyright:	© May 00 Control Techniques SSPD
Author:	Paul Bennett
Issue Code:	2
System File:	V2.07.05
Firmware (UD77)	V1.00.00

Contents

1 Introduction

NOTE

A brand new Unidrive CAN interface will NOT communicate with any other devices via the CAN interface. A DPL program MUST be written and downloaded to the module before any communications over a CAN network can be achieved.

SYPT is required to write a program that can control communications via the CAN interface. The basic functions used to communicate between the DPL program and the CAN controller are provided as function blocks, and come as part of the SYPT Function Block Library.

To download a DPL program from SYPT via the PC RS232 port, configure the COMMUNICATIONS to use the MD29MON protocol.

Although the DPL Toolkit cannot be used to write programs for the CAN interface, it can still be used to download BIN files created using SYPT. The FLASHER download utility program (supplied as part of SYPT and the DPL Toolkit) will also download pre-compiled BIN files to the CAN module.

NOTE

Drive parameters are denoted in this manual by "#MM.PP", where MM refers to the menu number, and PP refers to the parameter number within that menu. Please refer to the Unidrive manual for parameter definitions.

1.1 Unidrive CAN Interface

The Unidrive CAN interface for Unidrive is supplied in a large option module package. The CAN interface uses the UD70 Applications card as a host.

The UD70 retains full functionality, allowing the user to download normal DPL application programs. No program modifications are required to allow existing DPL programs to run. A different UD70 operating system file ("DPLCAN.SYS") is used, and the UD70 has this system file pre-loaded.

NOTE

System file V2.07.05 or later must be loaded to support the Unidrive CAN interface.

1.2 CAN V2.0 Part B Passive

When CAN was originally developed the CAN identifier field was specified as 11 bits. Subsequent developments of CAN allowed identifiers with an additional 18 bits, vastly increasing the number of available identifiers. Bus arbitration is determined over the full 29-bit identifier in such systems.

The Unidrive CAN interface is "CAN V2.0 Part B Passive". This means that although 29 bit identifiers are not supported, it can co-exist on a network with nodes that do use 29-bit identifiers (known as Part B Active) without generating errors.

2 Mechanical Installation

NOTE

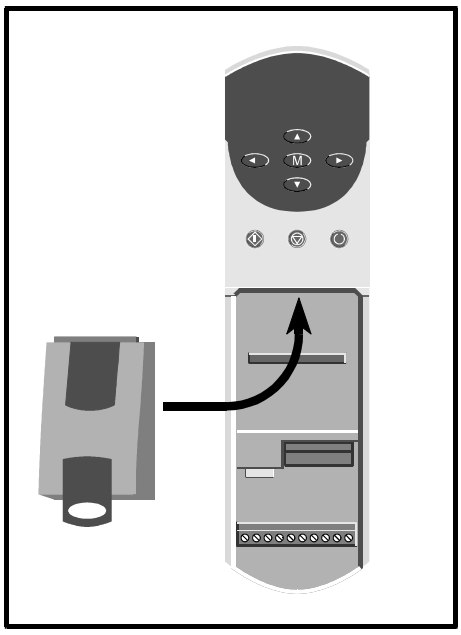
The Unidrive must be disconnected from the main supply before installing or removing an option module.

2.1 Unidrive

Isolate the Drive from the main supply and allow 5 minutes for the DC Bus capacitors to discharge.

Insert the Unidrive CAN interface module as shown below. Ensure that it is correctly inserted. The module will click firmly into place.

To remove the module, pull on the black tab, and the module will disengage from the connector and pull out of the Drive.

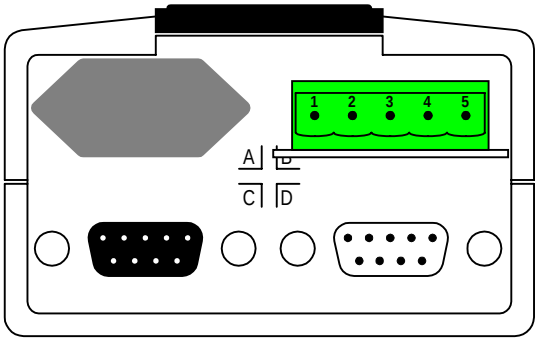


3 Electrical Installation

3.1 CAN Connector

The Unidrive CAN Interface can be connected to a CAN network directly onto the 9-way D-type connector (B), or by a standard 5-way screw terminal socket (B), using the small converter board provided with the Unidrive CAN interface.

Connectors C and D on the Unidrive CAN interface are the RS232 programming port (C) and RS485 general purpose communications port (D) of the UD70.



The pin connections for the D-type connector and the 5-way terminal block connector are given in the table below.

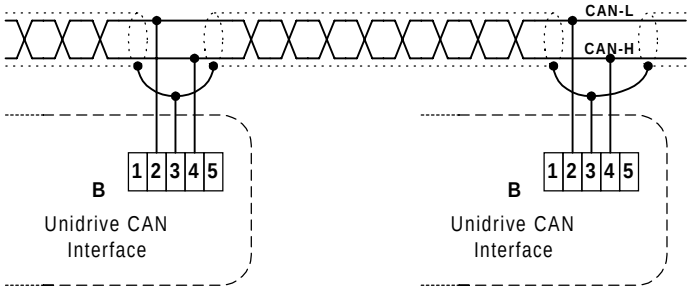
Signal	Terminal Block	D-type Terminal	Function
0v	1	3	Power supply 0V (Black)
CAN-L	2	2	Data signal low (Blue)
SHIELD	3	5	Shield
CAN-H	4	7	Data signal high (White)
V_{DC}	5	9	Power Supply +24V (Red)

NOTE

All diagrams in this manual illustrate connections using the 5-way screw terminal connector block.

3.2 CAN Connections

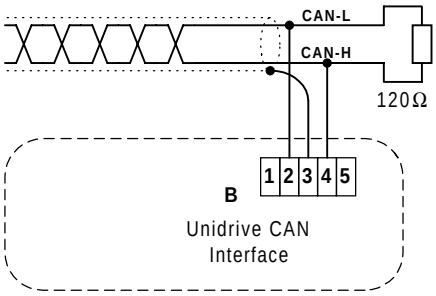
To connect a node to the CAN network, make the connections as shown in the diagram below. The cable screen must be connected to the middle pin of the CAN connector.



3.3 CAN Network Termination

There is no termination resistor supplied on the Unidrive CAN Interface. It is the user's responsibility to ensure that BOTH ends of each section of network cable are correctly terminated.

A 120Ω 0.25W resistor should be connected between the CAN_H and CAN_L lines on the node at each end of the main trunk cable, as shown in the diagram below. When the system is powered down, approximately 60Ω should be measured between the CAN_H and CAN_L lines. Resistor tolerance must be ±10% or better.



NOTE

The above method of connecting the termination resistor ensures that the network remains terminated when the CAN connector is disconnected from the node.

It is very important in high-speed communications networks that the network is correctly terminated. Failure to terminate the network properly may mean that the network operates with substantially reduced noise immunity, or in the worst case, the network doesn't work at all.

3.4 Maximum Network Length

The maximum network length depends on the data rate required. The maximum number of nodes on a network without a repeater is 32.

Data Rate	Maximum Bus Length (m)
10 kbits/sec	1000
20 kbits/sec	1000
50 kbits/sec	1000
125 kbits/sec	500
250 kbits/sec	250
500 kbits/sec	100
800 kbits/sec	50
1.0 Mbits/sec	30

3.5 External Power Supply

The Unidrive CAN interface chipset is powered from the internal supply of the host Drive. However, an external +24V power supply can be connected to power the transceiver circuitry if required. This may be desirable to keep the physical characteristics of the network the same if nodes are to be switched on and off during network operation. Each Unidrive CAN interface will draw 5mA from the external supply.

4 Node Configuration

4.1 CAN Enable

#20.01 -1 = CAN disabled ≥ 0 = CAN enabled

The CAN interface is enabled by setting #20.01 to a value of 0 or higher. If circumstances require that the CAN interface be disabled under certain circumstances, this can be achieved by setting #20.01 to -1, and setting #17.19 to 1 to completely reset the UD70 and CAN controller.

CAN does not actually implement any protocol on top of the CAN hardware data-link layers. However, virtually every protocol in existence does require each node to have a unique address or code.

To provide some common ground for people implementing CAN protocols, it is recommended that #20.01 should be used as the node address. #20.01 can also be used to simply specify the CAN identifier to be used by the node. The DPL program can simply copy the value directly into the 11-bit identifier of the outgoing CAN frame.

4.2 Data Rate

#20.02 Data Rate

The data rate configured at each node must be the same. There is no automatic detection of the data rate on CAN.

#20.02	Data Rate		#20.02	Data Rate
0	10 kbits/sec		4	250 kbits/sec
1	20 kbits/sec		5	500 kbits/sec
2	50 kbits/sec		6	800 kbits/sec
3	125 kbits/sec		7	1.0 Mbits/sec

Setting any other value in #17.14 will result in trip "tr62", if run-time trips have been enabled. (#17.14 = 1)

4.3 Event Task Control

#17.23 Event Task Trigger Source

#20.03 Event Task Trigger Slot

The EVENT task can be triggered by the timer/counter function inside the UD70, or it can be triggered by the CAN network. When a CAN data frame arrives in the slot specified in #20.03, this will trigger the EVENT task. The slot mask must also be configured to enable the slot.

#17.23	#20.03	EVENT Task Trigger Source	Comment
0	X	Internal UD70 timer/counter	Default.
1	0 to 15	CAN Network	Slot mask must be configured in DPL with **PUTCAN** command.

#20.03 will be defaulted to 0 if an invalid value is detected during initialisation.

4.4 Initialising Set-up Changes

CAN configuration parameters are only read during the initialisation sequence of the CAN interface, thus preventing unpredictable network behaviour while parameters are being edited. When parameters have been configured, the Unidrive CAN interface must be reset to implement the changes in network set-up.

The UD70 can be reset from the Unidrive keypad in 2 ways.

- Set #MM.00 to 1070 and press the red RESET button. This will implement any changes made to the CAN configuration, but the changes will NOT be stored. If power is lost to the Drive, the changes made will be lost, and the UD70 will revert to the stored configuration.
- Set #17.19 to 1. This causes a full reset of the UD70, and implements any changes made to the CAN configuration. It will also force the UD70 to store the #20.PP parameters in FLASH memory, thus ensuring that the changes will not be lost when power is removed from the Drive. The UD70 will reset #17.19 to 0, when the reset sequence is complete.

5 CAN Commands

The Unidrive CAN interface does not directly support any CAN protocol that may be available. When used in conjunction with SYPT, five function blocks (**PUTCAN**, **GETCAN, CANSTATUS, ENABLECANTRIPS** and **RESETCANTIMER**) are provided that allow the DPL programmer to have full control over the transmit/receive slots in the CAN controller. This means that any valid CAN message can be configured using DPL code and sent out over the CAN network.

To communicate with other CAN-based equipment, the user must have details of the protocol that the equipment uses, and write a DPL program to construct the necessary CAN message attributes, before data can be transferred over the CAN network. Similarly, DPL code must also be written to decode and handle the data contained within messages received from other equipment on the network.

The table below gives a brief summary of the commands available for use in controlling the CAN interface from the DPL program.

Command	Description
PUTCAN	Used to download data frame information, configure CAN slot identifier masks, and update the Remote Transmit Request response message from the DPL program.
GETCAN	Used to upload a data frame from the specified CAN slot.
CANSTATUS	Used to check the status of a CAN slot, to see if the specified slot has a message loaded (either a new data frame received, or a data frame waiting to be transmitted), or if the slot is empty.
ENABLECANTRIPS	Enables certain automatic trips in the event of network loss, bus-off condition, or invalid configuration parameters.
RESETCANTIMER	Resets the CAN controller's internal timer. This timer is used to return the time stamp (Time) argument from the **GETCAN** function.

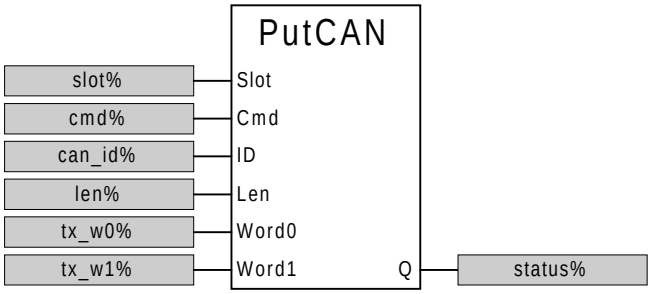
5.1 PUTCAN

This function block is used to transmit a CAN data frame, set a message filter for message reception, send a Remote Transmit Request (RTR) frame, and configure and RTR response data frame.

DPL CODE

```
status% = PUTCAN(slot%, cmd%, can_id%, len%, tx_w0%, tx_w1%)
```

FUNCTION BLOCK DIAGRAM



Argument	Range	Description
slot%	0 to 15	Destination message slot in the CAN controller.
cmd%	0 to 3	Command 0 = Transmit message 1 = Configure mask for specified CAN slot. 2 = Send RTR message 3 = Configure RTR response message
can_id%	0 to 2047	11bit CAN ID
len%	0 to 8	Number of data bytes
tx_w0%	$\pm 2^{31}$	32bit data word containing byte 0 (LSB) through to byte 3 (MSB) of the CAN data field. Unused bytes should be set to zero
tx_w1%	$\pm 2^{31}$	32bit data word containing byte 4 (LSB) through to byte 7 (MSB) of the CAN data field. Unused bytes should be set to zero
status%	0 or 1	Error code 0 = Command failed 1 = Command successfully completed

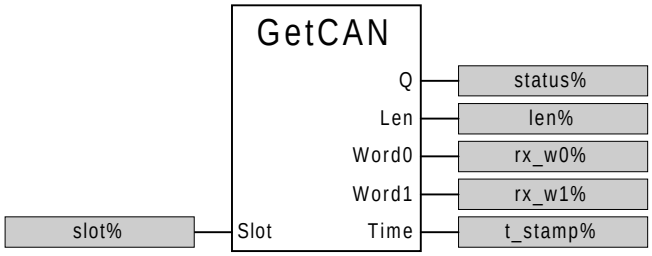
5.2 GETCAN

This function block examines the specified message slot, and returns the CAN Identifier, Data Length Code and the data bytes. Unused data bytes will be set to 0.

DPL CODE

```
(status%, len%, rx_w0%, rx_w1%, t_stamp%) = GETCAN(slot%)
```

FUNCTION BLOCK DIAGRAM



Argument	Range	Description
slot%	0 to 15	Slot from which the message is to be read.
status%	0 or 1	Error code 0 = Command failed – slot is empty 1 = Message read successfully - all output arguments are valid
len%	0 to 8	Number of data bytes
rx_w0%	$\pm 2^{31}$	32bit data word containing bytes 0 (LSB) through 3 (MSB) of the CAN data field. Unused bytes will be set to zero
rx_w1%	$\pm 2^{31}$	32bit data word containing bytes 4 (LSB) through 7 (MSB) of the CAN data field. Unused bytes will be set to zero
t_stamp%	0 to 255	8-bit time-stamp. 1 unit represents 32 * t_{bit}. The value rolls over to zero when the counter reaches 255. Only slots 0 to 7 will return a time-stamp value, slots 8 to 15 will always return -1. This value can be used to determine the order in which data frames were actually received.

5.3 CANSTATUS

This function block returns the status of the specified message slot.

DPL CODE

```
status% = CANSTATUS(slot%)
```

FUNCTION BLOCK DIAGRAM

Argument	Range	Description
`slot%`	0 to 15	Slot Number
`status%`	0 to 2	0 = Slot is empty 1 = Message transmission in progress 2 = New message has been received

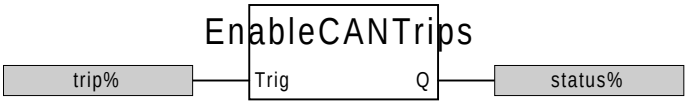
5.4 ENABLECANTRIPS

This function block (with #17.14) enables the CAN system trips. This will cause the Unidrive CAN interface to trip the Drive if certain network error conditions are detected.

DPL CODE

```
status% = ENABLECANTRIPS(trip%)
```

FUNCTION BLOCK DIAGRAM



Argument	Range	Description
trip%	0 to 2	0 = Trips disabled 1 = Bus Off trip only (tr61) 2 = All trips enabled (tr60 and tr61)
status%	0 or 1	0 = Operation Failed 1 = Operation Successful

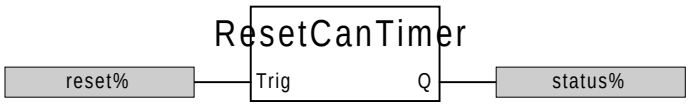
5.5 RESETCANTIMER

This function block resets the internal timer in the CAN controller to 0. When messages are received in slots 0 to 7, they are time-stamped with the contents of the CAN controller's internal timer. The GETCAN function block returns the time-stamp value.

DPL CODE

```
status% = RESETCANTIMER(reset%)
```

FUNCTION BLOCK DIAGRAM



Argument	Range	Description
reset%	0 or 1	0 = No action 1 = Reset CAN timer
status%	0 or 1	0 = Operation Failed 1 = Operation Successful

6 Using the CAN Commands

6.1 What are CAN Slots?

"CAN Slots" are areas in the CAN controller where data frames are stored temporarily. They can be configured either to receive certain CAN frames, or to hold a CAN frame for transmission.

Each slot must be configured to receive a data frame by defining the "slot mask" with an 11-bit CAN identifier. When a data frame has been passed the error checking, the CAN controller compares CAN identifier with the "slot mask" for each slot. If a slot is found where the slot mask identifier matches the data frame identifier, the data frame is placed in this slot and the slot status register is updated. The slot will keep the message (rejecting subsequent messages that match the slot mask identifier) until the CPU reads the message form the slot, and resets the slot status register.

To transmit a CAN data frame, the CPU must load the CAN identifier, data length code and the data bytes to be sent into a slot. The CAN controller will generate the necessary control bits and error code to construct the complete data frame, and will start to "arbitrate" for the bus. Bus arbitration is handled automatically by the CAN controller, and requires no further action from the CPU.

The Unidrive CAN interface has 16 slots in the CAN controller, and all slots can be configured to transmit or receive data frames. In general, most slots will be configured to receive messages with specified CAN Ids, with one or two slots reserved solely for transmitting messages. If a slot that has been configured to receive data frames with a specific CAN identifier, it will lose that configuration if it is used to transmit a data frame. Slot configuration is totally flexible, and slots can be re-configured at any time.

6.2 Transmitting a CAN Frame

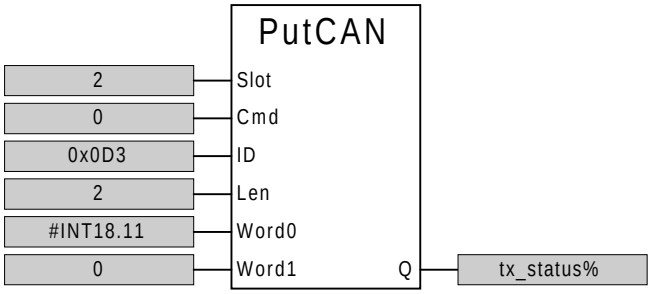
Send the CAN message identifier and data bytes to the CAN controller. Message will be queued in the specified slot for transmission. The CAN controller will automatically handle the arbitration for the network. If more than one message is waiting to be transmitted, the message with the highest priority (lowest identifier number) will be sent out when the node next wins control of the bus.

The example below shows how to put a 16-bit data value (taken from a Drive parameter) into two 8-bit data bytes, and transmit the message out of slot 2. The CAN identifier is 0x0D3 (211 decimal).

DPL CODE

```
tx_status% = PUTCAN(2, 0, 0x0D3, 2, #18.11, 0)
```

FUNCTION BLOCK DIAGRAM



6.3 Configuring a Slot Mask

Most CAN protocols derive the message identifier from local node data. Virtually all bus systems require each node to have a unique node address. If the node address is included as part of the CAN identifier code, this guarantees that no two nodes can produce messages with the same CAN identifier code.

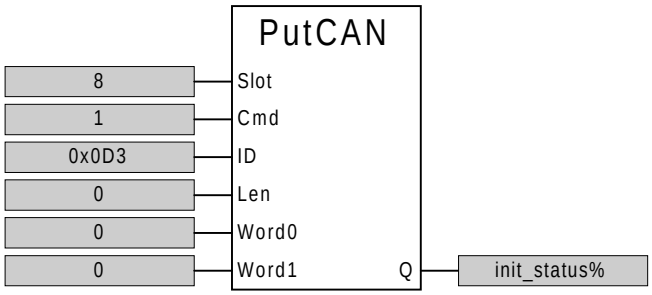
When the slot masks are configured, it is best to use identical code in each node and get certain information from the Drive itself, rather than use fixed numbers. This means that when a program is installed into another node on the network, it will produce messages with unique CAN identifier codes.

The example below shows how to configure slot 8 to receive the message transmitted in section 6.2. Typically, this command would be issued in the INITIAL task, but this does not have to be the case.

DPL CODE

```
init_status% = PUTCAN(8, 1, 0x0D3, 0, 0, 0)
```

FUNCTION BLOCK DIAGRAM



To receive a message from the CAN network, a slot must be configured with a mask for the CAN ID. If a message is received by the CAN controller, with an identifier that matches the mask of one of the slots, the message is put into that slot. Otherwise, the CAN controller discards the message.

The mask for each slot should be configured during the INITIAL task of the DPL program. (Setting cmd% to 1 configures the slot mask) This allows the CAN controller to filter out messages that are not intended for the node. By configuring the slot masks in the INITIAL task, this guarantees that the CAN controller is always re-configured after a RESET.

When the UD70 is reset, all CAN slots are initialised to mask identifier 0. This means that only messages with an identifier of 0 (usually reserved) will be received by the CAN controller.

CAN slots should be configured with unique mask ids. If 2 or more slots have the same mask id, generally the slot with the highest slot number (0 to 15) will receive the message, although this is not guaranteed to be the case.

A CAN slot should not be used for both transmitting and receiving messages. If a slot has been set up to receive a message (i.e. the mask has been set) and it is subsequently used to transmit a message, the mask information will be lost, and the slot will not receive any more messages. The mask must be re-configured before the slot can be used to receive messages again.

6.4 CAN Slot Status

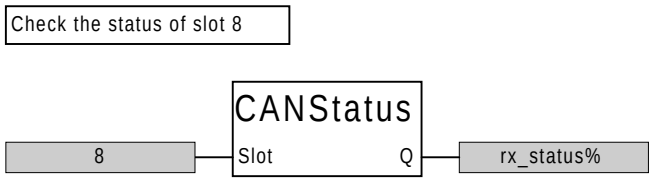
The status of each CAN slot can be determined using the CANSTATUS function block. This function is used to find slots in the CAN controller that have received new data frames, or slots which are waiting to transmit data frames over the CAN network.

When a slot has received a new data frame, it will reject all subsequent data frames with matching CAN identifiers until the data frame held is unloaded from the slot.

DPL CODE

```
; check the status of slot 8
rx_status% = CANSTATUS(8)
```

FUNCTION BLOCK DIAGRAM



rx_status%	0 = Slot is empty 1 = Message transmission in progress 2 = New message has been received

6.5 Receiving a CAN Frame

The CAN controller will not notify the DPL program that a new message has arrived. The program must use the **CANSTATUS** command to check each slot and see if a new message has arrived, and use **GETCAN** to retrieve the data values from the CAN message.

CANSTATUS returns the slot status very quickly, so a check of all slots configured to receive data frames, using **CANSTATUS** in a **DO…WHILE** loop, is an efficient way of determining if any new data frames have been received. **GETCAN** can then be used to retrieve data from slots with new data frames.

When a message arrives, it is placed into the slot with a matching identifier mask. Subsequent messages with the same identifier will be discarded until the DPL program has retrieved the message already in the slot.

The example below shows how to check for receipt of the message transmitted in section 6.2. The slot was configured transmit a message over the network in section 6.3, with the following attributes:

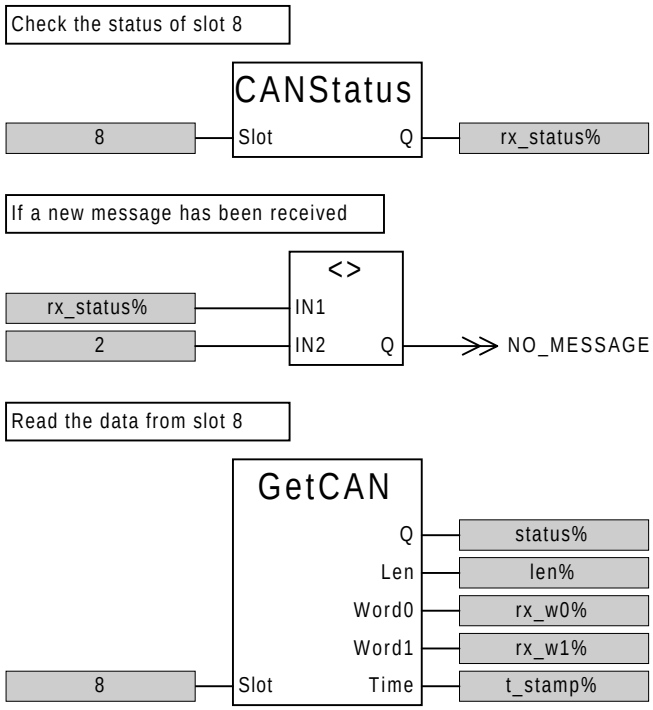
DPL CODE

```
; check the status of slot 8
rx_status% = CANSTATUS(8)

; if a new message has been received
IF rx_status% = 2 THEN

       ; read the data frame from slot 8
       (status%, len%, rx_w0%, rx_w1%, t_stamp%) = GETCAN(8)
ENDIF
```

FUNCTION BLOCK DIAGRAM



NO_MESSAGE:

6.6 Automatic Network Error Trips

Certain error conditions with the CAN network can be detected, and the Drive can be tripped automatically when an error condition is detected.

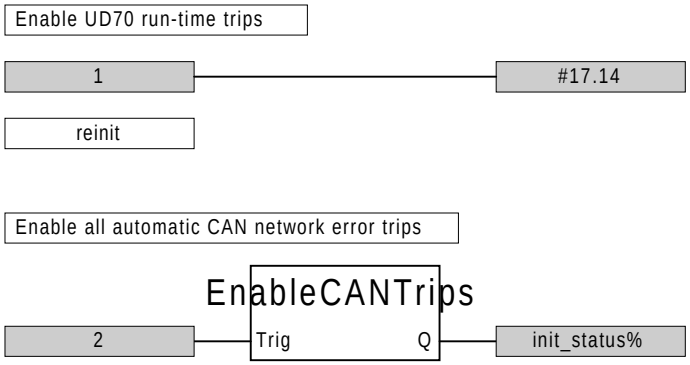
The example code listed below would normally be part of the INITIAL task, but this does not have to be the case.

DPL CODE

```
#17.14 = 1     ; enable UD70 run-time trips
REINIT

; enable all automatic CAN network error trips
init_status% = ENABLECANTRIPS(2)
```

FUNCTION BLOCK DIAGRAM



By setting #17.14 to 1 and executing the **REINIT** command, this will force the UD70 to read the UD70 configuration parameters (#17.PP) again, and enable the UD70 run-time trips.

6.7 Remote Transmit Request

A "Remote Transmit Request" frame allows a node to request a data frame from another node. The data frame that is returned is the RTR response frame. The "remote" node will load the RTR response frame into the CAN controller, and transmission of the RTR response frame is handled automatically in hardware by the CAN controller. The CPU in the "remote" node is does not even need to know that the RTR response frame has been transmitted.

A node can update its RTR response frame at any time. Depending on the nature of the data being sent, it could be updated on a regular basis or when a change of data occurs.

6.7.1 Requesting Node

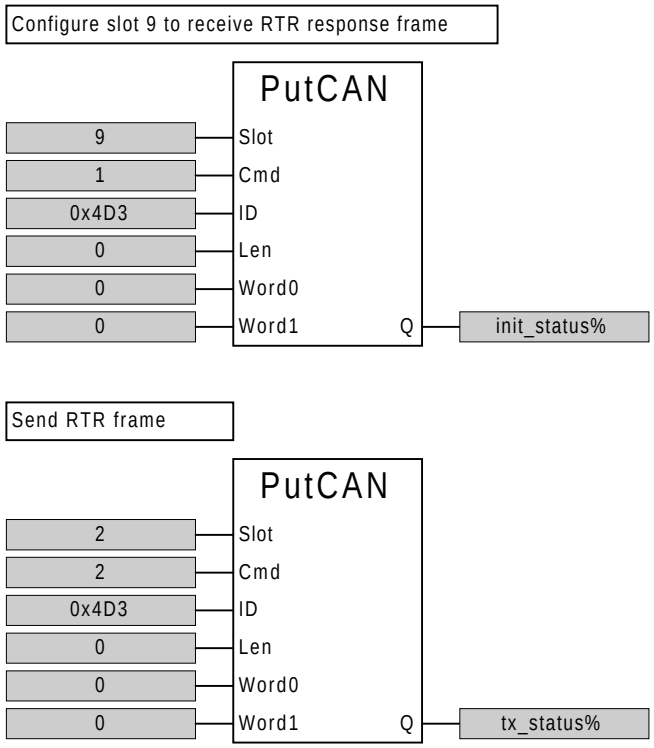
The "requesting" node must transmit an RTR frame from one of the allocated transmit slots. The CAN identifier of the RTR frame MUST match the CAN identifier of the RTR response message expected back from the "remote" node.

The "remote" node will reply with the pre-configured RTR message, so the "requesting" node MUST configure a receive slot with the slot mask set to the CAN identifier of the transmitted RTR frame. IF this is not done, the "requesting" node will discard the RTR response frame.

DPL CODE

```
; configure slot 9 to receive RTR response frame
init_status% = PUTCAN(9, 1, 0x4D3, 0, 0, 0)
tx_status% = PUTCAN(2, 2, 0x4D3, 0, 0, 0)  ; send RTR frame
```

FUNCTION BLOCK DIAGRAM



Use **CANSTATUS** and **GETCAN** to detect and retrieve the RTR response frame. (See section 6.5). Usually, only one slot needs to be allocated to receive RTR response frames, as the slot mask can be re-configured with the appropriate CAN identifier before the RTR frame is actually sent.

6.7.2 Remote Node

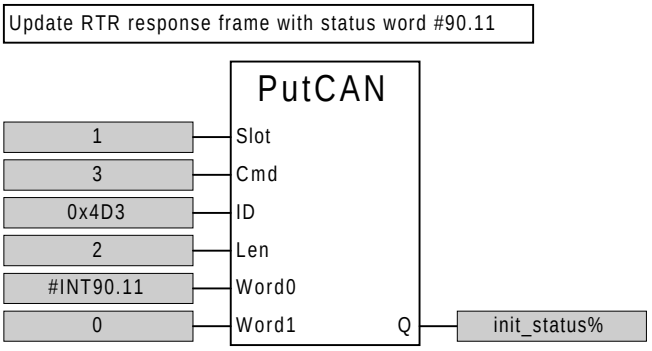
The "remote" node will automatically respond to an RTR frame if the CAN identifier of the RTR frame matches the CAN identifier of the RTR response frame held in the CAN controller. The RTR response frame can be transmitted using one of the slots reserved for transmitting data frames.

Use the **PUTCAN** command with **cmd% = 3** to define the RTR response frame. The CAN identifier, data length code and data bytes must all be specified. The RTR response frame is updated in the CAN controller every time this command is executed in the DPL program.

DPL CODE

```
; update RTR response frame with status word
init_status% = PUTCAN(1, 3, 0x4D3, 2, #90.11, 0)
```

FUNCTION BLOCK DIAGRAM



A typical use for RTR messages may be for a "master" node to read the status of other nodes on the network. If every node was to transmit a status word to the master, it could easily be overloaded. Instead, every node simply updates the status word in the RTR slot, and the master cycles through each node, asking for the RTR message. This prevents overload of the master, as it only asks for new data when it is ready to receive it.

6.8 Event Task Trigger

The Unidrive CAN interface provides a facility to trigger the EVENT task from the CAN network, instead of from the internal TIMER unit. To trigger the EVENT task, #17.23 must be set to 1, and a valid slot number specified in #20.03. The specified slot must be configured with the required mask as normal.

NOTE

The UD70 must be reset by setting #17.19 to 1 before any changes in the configuration of #17.23 and #20.03 will take effect.

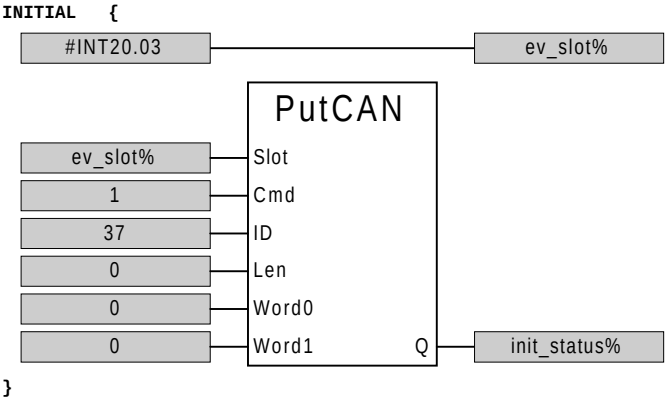
When the EVENT task is triggered, **GETCAN** must be used to retrieve the message, and empty the slot. Failure to do this means that any subsequent messages arriving for this slot will be discarded. **CANSTATUS** is not necessary, as the EVENT task is only be triggered when a data frame is received in the slot specified by #20.03.

DPL CODE

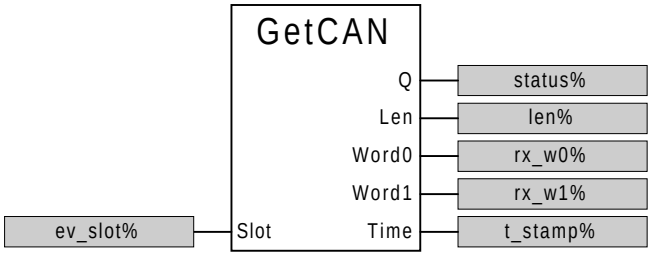
```
INITIAL   {
ev_slot% = #20.03
init_status% = PUTCAN(ev_slot%, 1, 37, 0, 0, 0)
}

EVENT   {
(status%, len%, rx_w0%, rx_w1%, t_stamp%) = GETCAN(ev_slot%)
}
```

FUNCTION BLOCK DIAGRAM



```
EVENT    {
```




```
}
```

6.8.1 Example Use of EVENT Task Trigger

This feature allows a simple strategy to be implemented to overcome problems with data skew, particularly on a network running at a comparatively slow data rate.

If 20 new speed references must be sent to 20 drives with minimal data skew, the values can be sent over the network, and buffered in each Drive when it receives the appropriate CAN message. When all messages have been sent, a SYNC message (with a unique identifier code) can be sent over the network to trigger the EVENT task in each Drive, which in turn transfers the speed reference from the temporary buffer area to the speed reference parameter.

There will still be some data skew, mainly due to fact that the speed loop cycles in the drives will not be synchronous with each other. However, the maximum skew will 1 speed loop cycle time (1.38ms), and almost certainly lower than the time taken to transmit all speed references over the CAN network.

For example, to transmit a message with a 16-bit speed reference (2 data bytes) over a CAN network running at 50 Kbits/sec will take approximately 1.4ms. With 20 drives on the network, total transmission time is 28ms. When compared with a maximum skew time of 1.38ms, the data skew is reduced by 95%.

The EVENT task has a higher priority than the SPEED and ENCODER tasks. If the EVENT task trigger function is being used to implement a fast update of parameters in the position loop, this could cause a problem. If the position loop code is interrupted and a new value forced into certain registers, the position loop will continue with a different value to the starting value, and this may cause problems.

To guarantee that this does not happen, the data should be passed to a temporary variable in the EVENT task. The SPEED or ENCODER task should then be used to transfer the value to the position loop parameters. The SPEED and ENCODER task code does not run until all system file operations attached to these tasks has been completed, so the new value will be picked up on the next task.

7 Diagnostic Information

7.1 Fieldbus Code

Unidrive: #20.14

The fieldbus code identifies the hardware level in the option module. This information is vital when trying to determine what upgrades can be performed on older modules.

The identification of the high-speed communications option module can be read from #20.14 on the Unidrive display. (The code is also available as #89.04.) This number is shown in the form XYZ, where X is the fieldbus type, Y is the fieldbus flavour, and Z is the hardware revision level.

Code	Fieldbus Type	Fieldbus Flavour	Hardware Revision
520	5 (CAN)	2 (CAN)	0 (UD77A and UD77B Issue 1)

NOTE

System file V2.07.05 or later must be installed in the UD70 to indicate the full fieldbus code.

7.2 Firmware Version

Unidrive: #20.15

The version of firmware fitted to the CAN interface can be read from #20.15 on Unidrive.

	Code	Firmware Version	Hardware Revision
Unidrive	100	V1.00.00	0

The Hardware Revision column shows the hardware levels that can accept each version of firmware.

7.3 System File Version

Unidrive: #17.02

The system file installed in the UD70 must be the correct file for the communications option installed. The system file for the Unidrive CAN interface is “DPLCAN.SYS”.

The system file that must be installed can depend on the level of hardware and firmware in the module. In general, new system files are backward compatible with older versions of firmware and hardware, but there may be some limitations when upgrading older modules.

The system file version can be read from parameter #17.02 on the Unidrive.

Firmware	Hardware Revision	System File	Comments
V1.00.00	0	V2.07.05 or later	V2.07.05 was the first system file release for the Unidrive CAN interface.

7.4 CAN Enable

Unidrive: #20.01

The CAN interface must be enabled to communicate with the DPL program. If #20.01 is set to -1, the CAN controller is completely disabled.

Most communications systems require that each node on a network is assigned some sort of unique address. When DPL code is written to implement a protocol, #20.01 should be used to configure the node address.

7.5 Network Data Rate

Unidrive: #20.02

Every node on the system MUST be configured to run at the same data rate. Failure to ensure this can cause unpredictable results, as nodes attempting to communicate at different data rates can corrupt data frames on the network.

If a node has been configured to run at a data rate that is different to the rest of the network, bus-off errors on some or all nodes may result. In general, the data rate should be configured on a node BEFORE it is physically connected to the network.

7.6 Network Status

Unidrive: #20.50

#20.50 on Unidrive indicates the status of the CAN network. Under normal conditions, #20.50 should indicate that the network is running. The loss of network condition cannot be detected until the node attempts to transmit a data frame over the network.

#20.50 is updated immediately when a change of status is detected, otherwise the status indication is updated every second.

#20.50	Status	Description
1	Network running	CAN network OK, data frames can be transmitted or received.
0	Initialising hardware	UD70 is initialising the CAN interface.
-1	Loss of network	Data frames are not being acknowledged. Possible causes are faulty network connections, or the node is the only node present on the network.
-2	Bus Off	Bus Off condition requires a UD70 reset. Indicates that the node is having trouble transmitting messages, or faulty wiring somewhere on the network.

7.7 Trip Action On Network Loss

To enable the automatic trips when a network error occurs, the **ENABLECANTRIPS** function block must be called from the DPL program. (See section 6.6.)

#17.14	Mode	Action
0	Ride-through	The Unidrive will continue to operate with the previous values received from the network
1	Trip	Automatic CAN trips are enabled, provided the **ENABLECANTRIPS** function block has been executed in the DPL program.

If a DPL program is present in the UD70, this will continue to run as normal. (An ERROR task is not raised.) If #17.14 is changed, the UD70 must be reset by setting #MM.00 to 1070, and pressing RESET to make the trip mode change take effect.

Trip Code	Fault Description	Possible Causes
tr60	Network Loss	"tr60 occurs when a node sends a data frame, and no other node acknowledges receipt of the frame message. This can happen if there is only one node on the network, the network connection has been lost, or the node is transmitting corrupted data frames.
tr61	Bus Off Error	"tr61" indicates the Can "Bus Off" condition. A complete reset of the CAN controller is required to clear this condition.
tr62	Configuration Error	"tr62" indicates an error in the set-up of the CAN interface. Ensure parameters #20.01, #20.02 and #20.03 have been configured correctly.

Problems can occur during initialisation, particularly with only 2 nodes on a network. If the first node completes initialisation before the second node, it will trip on "tr60". As the second node has not been initialised yet, it will not acknowledge receipt of the data frame, causing the first node to think it is transmitting error frames and trip.

The **ENABLECANSTATUS** function block allows full error detection to be enabled when network operation has been fully established. This can be enabled during any task, e.g. when the first network message from a remote node is received.

7.8 Other UD70 Trip Codes

If certain errors occur, the Unidrive will trip and show the trip code in the upper window.

Trip Code	Error
tr56	The UD70 does not contain the correct operating system. Download the system file “DPLCAN.SYS”.
tr57	An illegal operating system call has been made, e.g. **WRNET** is a CTNet command, and is not available with CAN.

8 Example Application

The example application below implements the "Easy Mode" cyclic data available with CTNet. The node, menu and parameter mappings are defined in exactly the same way and use the same parameters as with CTNet, with one node (the "pseudo-master") being responsible for generating the data synchronisation interrupt.

This example addresses the allocation of CAN slots for transmitting and receiving data frames, allocation of CAN identifier codes, and use of the EVENT task trigger.

8.1 "Easy Mode" Cyclic Data using DPLCAN

The basic specification for data transfer is that each node has 3 IN and 3 OUT data channels. CTNet is a "producer-orientated" network, i.e. the target node data (target node address and data channel) is only configured in the transmitting node.

8.1.1 CAN Identifier for Transmitting Data Frames

The source (producing) node knows where the data is to be sent, but the target node has no idea where the data is coming from. This means that the target node must have fixed CAN identifiers, and the source node must assign the CAN identifier such that it matches the CAN identifier in the intended target node. This example will limit the number of nodes to 31, with 3 OUT data channels per node, so the CAN identifier will be assigned as shown below.

```
can_identifier% = (target_channel% * 32)  +  target_node%
```

Variable	Range	Description
`can_identifier%`	33 to 127	The CAN identifier that is assigned for each outgoing data frame.
`target_channel%`	1 to 3	Defines the target channel into which the data will be written in the target node. (0 is reserved.)
`target_node%`	1 to 31	Defines the target node for the data frame.

8.1.2 CAN Identifier for Receiving Data Frames

The target node does not know where the data is actually going to come from, so the input slots can be configured with fixed CAN identifiers. If a source node produces a message that matches one of the configured slot masks in target node, the data will be loaded into the appropriate CAN slot.

```
can_slot_mask% = (in_channel% * 32)  +  node_address%
```

Variable	Range	Description
`can_slot_mask%`	33 to 127	The CAN identifier that is assigned for each outgoing data frame.
`in_channel%`	1 to 3	Defines the target channel into which the data will be written in the target node. (0 is reserved.)
`node_address%`	1 to 31	Defines the target node for the data frame.

8.1.3 SYNC Message Generator

One node on the network must be responsible for generating the SYNC message. This is the message that informs the other nodes that it is time to send their configured messages again. This node is known as the "pseudo-master".

The TIMER unit is used in the pseudo-master to trigger the EVENT task on a regular defined time-base. Once the EVENT task has been triggered, the SYNC message is immediately loaded into the CAN controller for transmission. This message format is fixed, with CAN identifier set to 1 to give very high priority, and no data bytes.

The data frames for transmission out of channels 1, 2 and 3 are also configured and downloaded to the CAN controller in the EVENT task. If separate CAN slots are used for each OUT channel, all three data frames can be downloaded immediately, and the CAN controller left to handle transmission automatically.

8.1.4 EVENT Task Trigger

A signal is required from the "pseudo-master" to indicate that it is time to construct and transmit the cyclic OUT data frames. By allocating a slot as the EVENT Task Trigger slot, and setting the slot mask to 1, the EVENT task can be used to configure and load the data frames onto the CAN controller.

8.1.5 CAN Slot Allocation

CAN slots need to be allocated for each function. The allocation chosen for this example is shown in the table below.

Slot		Function	Description
0	IN	EVENT Trigger	Configured with slot mask = 1. This is the slot used to trigger the EVENT task in the "slave" drives.
1	IN	IN Channel 1	IN data channel 1. Data is automatically transferred to _S90%.
2	IN	IN Channel 2	IN data channel 2. Data is automatically transferred to _S91%.
3	IN	IN Channel 3	IN data channel 3. Data is automatically transferred to _S92%.
4		Not used	
5		Not used	
6		Not used	
7		Not used	
8		Not used	
9		Not used	
10	OUT	SYNC Telegram	Slot used to transmit the SYNC telegram data frame
11	OUT	OUT Channel 1	OUT data channel 1. Data is picked up from _R90%.
12	OUT	OUT Channel 2	OUT data channel 2. Data is picked up from _R91%.
13	OUT	OUT Channel 3	OUT data channel 3. Data is picked up from _R92%.
14		Not used	
15		Not used	

8.1.6 Receiving Data Frames

Receiving data frames cannot be done in the EVENT task, as the time delay before incoming data frames are received cannot be predicted. As the EVENT task has priority over all tasks except the INITIAL task. it cannot wait for all messages to be received, as this will prevent all other tasks from running.

Any of the time-base tasks (SPEED, ENCODER or CLOCK) can be used to check for messages received but the slower the task, the greater the potential delay between receiving a data frame and updating the destination parameter. For this reason, this example uses a continuous loop in the BACKGROUND task.

8.1.7 Node Configuration

The DPL code in this section implements the above specifications to produce regular data transfer, with virtually identical configuration to CTNet "Easy Mode" cyclic data.

Function	Unidrive	Setting
Node Address	#20.01	1 to 31
Network Data Rate	#20.02	0 to 7
Synchronisation Message	#20.13	1 to 130ms
EVENT Trigger Slot	#20.03	0

Slot	Source/Destination Parameter	Destination Node
IN Slot 1	#20.10 (MMPP)	
IN Slot 2	#20.11 (MMPP)	
IN Slot 3	#20.12 (MMPP)	
OUT Slot 1	#20.05 (MMPP)	#20.04 (NNNSS)
OUT Slot 2	#20.07 (MMPP)	#20.06 (NNNSS)
OUT Slot 3	#20.09 (MMPP)	#20.08 (NNNSS)

The source and destination parameters are entered in the form MMPP, where MM is the menu number and PP is the parameter number. The destination node and slot is entered in the form NNNSS, where NNN is the destination node address, and SS is the IN slot to write to.

8.1.8 DPL Code - INITIAL Task

```
INITIAL   {
node_address% = #20.01

; master mode
IF #20.13 > 0 THEN
       master% = 1
       #17.23 = 0    ; trigger EVENT task from timer

       ; configure TIMER unit
       #85.04 = LIMIT(#20.13, 100) * 500         ; re-load value
       #85.01 = 0x17

; slave mode
ELSE
       master% = 0
       #17.23 = 1    ; trigger EVENT task from slot 0

       ; slot 0 receives SYNC telegram, CAN-ID = 1
       init_status% = PUTCAN(0, 1, 1, 0, 0, 0)
ENDIF

REINIT        ; force UD&0 to re-read #17.23

; configure the masks for slots 1 to 3 to receive IN data
slot% = 1
DO
       ; calculate and configure masks for IN slots 1,2,3
       slot_mask% = (slot% * 32) + node_address%
       init_status% = PUTCAN(slot%, 1, slot_mask%, 0, 0, 0)
       slot% = slot% + 1
LOOP WHILE slot% <= 3

; parameter mapping configuration for input and output data
target1% = #20.04     ; target node and channel
out_source1% = #20.05 ; source parameter
target2% = #20.06     ; target node and channel
out_source2% = #20.07 ; source parameter
target3% = #20.08     ; target node and channel
out_source3% = #20.09 ; source parameter
in_dest1% = #20.10    ; destination parameter
in_dest2% = #20.11    ; destination parameter
in_dest3% = #20.12    ; destination parameter
}
```

8.1.9 DPL Code - EVENT Task

```
EVENT   {
; task is triggered by the timer on master, and by the
; SYNC telegram on slave

; if master, transmit SYNC telegram
IF master% = 1 THEN
       tx_status% = PUTCAN(10, 0, 1, 0, 0, 0)

; if slave, empty slot 0
ELSE
       (get_status%,len%,rx_w0%,rx_w1%,ts%) = GETCAN(0)
ENDIF

; channel 1, update value, calculate CAN-ID and send data frame
IF out_source1% <> 0 AND target1% <> 0 THEN
       _R90% = #out_source1%
       can_id1% = ((target1% % 100) * 32) + (target1% / 100)
       tx_status1% = PUTCAN(11, 0, can_id1%, 4, _R90%, 0)
ENDIF

; channel 2, update value, calculate CAN-ID and send data frame
IF out_source2% <> 0 AND target2% <> 0 THEN
       _R91% = #out_source2%
       can_id2% = ((target2% % 100) * 32) + (target2% / 100)
       tx_status2% = PUTCAN(12, 0, can_id2%, 4, _R91%, 0)
ENDIF

; channel 3, update value, calculate CAN-ID and send data frame
IF out_source3% <> 0 AND target3% <> 0 THEN
       _R92% = #out_source3%
       can_id3% = ((target3% % 100) * 32) + (target3% / 100)
       tx_status3% = PUTCAN(13, 0, can_id3%, 4, _R92%, 0)
ENDIF
}
```

8.1.10 DPL Code - BACKGROUND Task

```
BACKGROUND   {
TOP:

; check slots 1,2,3 for new received CAN frames
slot% = 1
DO
       rx_status% = CANSTATUS(slot%)

       ; if CAN slot has a new data frame
       IF rx_status% = 2 THEN
              (status%,len%,rx_w0%,rx_w1%,ts%) = GETCAN(slot%)

              ; slot 1 holds data for IN channel 1
              IF slot% = 1 THEN
                     _S90% = rx_word0%  ; update _S90%
                     #in_dest1% = _S90% ; update dest param

              ; slot 2 holds data for IN channel 2
              ELSEIF slot% = 2 THEN
                     _S91% = rx_word0%  ; update _S91%
                     #in_dest2% = _S91% ; update dest param

              ; slot 3 holds data for IN channel 3
              ELSEIF slot% = 3 THEN
                     _S92% = rx_word0%  ; update _S92%
                     #in_dest3% = _S92% ; update dest param
              ENDIF
       ENDIF
       slot% = slot% + 1
LOOP WHILE slot% <= 3

GOTO TOP:
}
```

9 CAN Overview

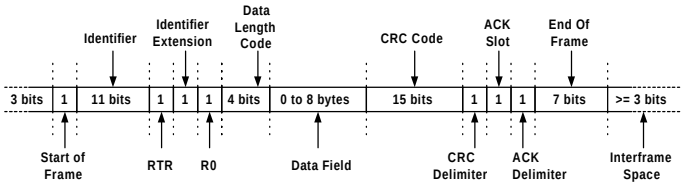
9.1 What is CAN?

CAN specifies a "physical layer" which defines the method used for transmitting individual bits of data over an electrical transmission line. It also specifies a "message protocol layer", which determines the structure of control bits, data bytes and error detection codes for each individual frame. This layer, known as the "CAN Data Frame", ensures the integrity of each group of data bytes as they are passed between nodes.

CAN does NOT specify a "system protocol layer". This layer actually uses the data bytes transmitted over the physical layer, and grouped and checked by the message protocol layer, to implement instructions defined by the system protocol. CANOpen and DeviceNet are examples of system protocols that use the CAN hardware and message layers.

The user must choose and implement the system protocol that is to use the CAN hardware. The Unidrive CAN allows the user to implement the protocol of their choice using DPL code, by providing access to the message protocol layer for transmission and receipt of data frames. Provided each node has the same protocol implemented, they will be able to communicate over a CAN network.

9.2 CAN Data Frame



9.2.1 CAN Identifier

The single most important part of the message is the CAN identifier. CAN uses this 11-bit number to determine which node will have control of he network to transmit a message. The lower the number in the identifier field, the higher the priority of the message. (See non-destructive bit arbitration.)

9.2.2 Remote Transmit Request

This bit indicates that the CAN frame is a remote frame, and is requesting another node on the network (with a matching identifier field) to transmit the pre-configured RTR response frame.

9.2.3 Identifier Extension

CAN 2.0 Part B can use 11 bits and/or 29 bit identifiers. Part B Passive nodes only use 11 bit identifiers, but can co-exist with nodes using 29-bit identifiers without generating errors. The Unidrive CAN interface is CAN V2.0 Part B Passive. Part B Active nodes use 29-bit identifiers.

9.2.4 Data Length Code

This is a 4-bit code that indicates the number of data bytes contained in the message. Data bytes can be from 0 to 8 bytes.

9.2.5 Data Field

The data field contains the actual data bytes for the message. Up to 8 data bytes can be included in a single CAN data frame.

9.2.6 CRC Code

The Cyclic Redundancy Check code is calculated using the bit pattern of the whole message. It is calculated automatically in hardware by the CAN controller when the relevant data is passed to it, and added to the final CAN frame to be transmitted.

When a CAN controller receives a data frame, it will re-calculate the CRC code on the received message, and check it against the CRC code included in the message. If the calculated CRC code matches the received CRC code, the message is deemed to have been transmitted error free.

If the CRC error check fails, the CAN controller discards the message.

9.2.7 ACK Slot

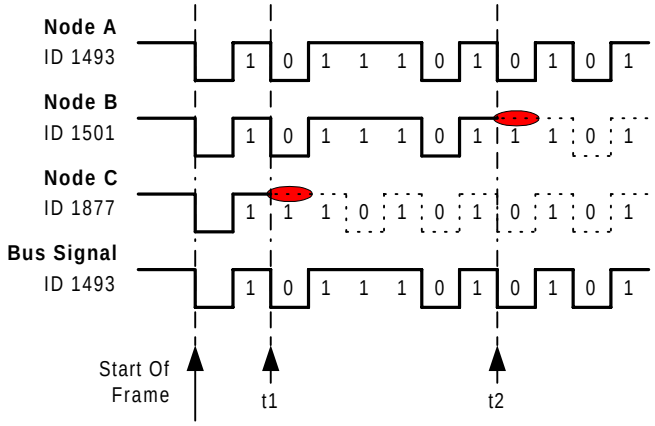
When transmitting a message, the node sets this bit passive (1). If ANY node on the network received the message error-free, it will set this bit dominant (0). If the transmitting node sees this bit dominant, it assumes that the message was transmitted OK.

9.2.8 End Of Frame

This is a 7 bit code that indicates the end of the CAN message. A delay of at least 3 bits will also occur before the cycle starts again.

9.2.9 Non-Destructive Bit Arbitration

All communications network must have a method of ensuring that only one node can transmit over the network at any point in time. This is necessary to prevent 2 or more nodes from attempting to transmit and corrupting each other's data message.



CAN use non-destructive bit arbitration to prevent data collisions.

9.2.10 Bit Stuffing

Non-Destructive Bit Arbitration relies on the timing between the bits on the message. This makes CAN intolerant of cable delays, and also requires that the various nodes on the network are kept synchronised.

A problem could occur if a message had a long sequence of consecutive 1's or 0's. If 20 consecutive bits have the same value, a 5% error in the clock frequency of a node (50ns at 1Mbit/sec) could cause a node to get out of synchronisation. To solve this problem, CAN uses a technique known as "Bit Stuffing".

"Bit-stuffing" is implemented in the hardware of the CAN controllers, and requires no action on the part of the programmer. If a message contains a sequence with more then 5 consecutive bits of the same value, a bit with the opposite value is inserted into the bitstream. Receiving nodes will have counted 5 consecutive bits of the same value, and will expect to see a transition on the sixth bit. When this occurs, each node discards the "stuff bit" so that the reconstituted message is the same as the transmitted message.

If a node receives a message containing 6 consecutive bits with the same value, it will flag an error, and discard the message.