AR-010-355

DSTO

Software Instrument Control Suite

David Clarke

DSTO-GD-0156

APPROVED FOR PUBLIC RELEASE

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Software Instrument Control Suite

David Clarke

Maritime Operations Division
Aeronautical and Maritime Research Laboratory

DSTO-GD-0156

ABSTRACT

The use of computers to control instrumentation can provide improvements in quality, quantity and turn around time of work carried out by a laboratory. These improvements must be balanced against the time taken to write the programs that control the instruments. This work documents a library of instrument control routines used to facilitate the task of programming and to enable the full advantage of computer controlled instrumentation to be realised.

RELEASE LIMITATION

Approved for public release

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

DSTO Aeronautical and Maritime Research Laboratory
PO Box 4331
Melbourne Victoria 3001

Telephone: (03) 9626 7000
Fax: (03) 9626 7999

AR-010-355
October 1997

APPROVED FOR PUBLIC RELEASE

Software Instrument Control Suite

Executive Summary

This document details the software library and control system written for the instruments used in controlling and recording data at the magnetic volume (a region in which the magnetic field may be controlled) at the Maritime Operations Division Maribyrnong.

Computer controlled experiments can have a very positive effect on the quality, quantity and turn around time of work carried out by a laboratory. The increase in quality is achieved as the computer controlled experiments, once set up correctly, will help eliminate errors due to carelessness and fatigue. An increase in efficiency can be achieved by reducing the staff effort required to run and produce primary data from the experiment. There will also be a reduction in the time taken to set up parameters in an existing experimental arrangement. This reduction in time required to run the experiments must be balanced against the time taken to write the software to run an experiment.

The process of setting up a computer controlled experiment can often be time consuming and frustrating as many instruments have large programming manuals and strange undocumented idiosyncrasies that are generally only discovered through trial and error.

This report provides a library of routines for a number of instruments. This means that the programmer no longer has to "know" the instrument inside out, just what he wants it to do. As an example, if he wants to check that an instrument is functioning correctly he just runs a procedure poll and it will notify him if there is a problem. He does not need to now the GPIB address of the instrument or what each bit in the status byte means. That work is already done. These modules will open the programming of experiments up to a wider group of people as the person writing a test procedure will no longer need to be familiar with the programming of the instrument.

The Software Instrument Control Suite currently allows for the use of five different instruments, additional instruments may be added if required.

A user interface has also been provided so basic settings of the instrument can be changed without altering the program. This facility allows non programmers to run an experiment and to make minor changes to tailor the experiment to individual tests.

Contents

1. Introduction

This library of routines is written to facilitate the creation of computer controlled experiments. To complement this library a series of dialog boxes have been created. These dialog boxes allow the change of device settings from the keyboard without altering the program. The dialog boxes provide the facility to set up to four sets of commands for each device. The commands are tested for range and syntax errors before a routine converts the information into a string format. Passage of information from the dialog boxes to the instruments is via an array of strings called a CommandSet. The dialog boxes write to the CommandSet and the software controlling the devices reads the settings from the CommandSet. Currently the dialog boxes and routines are available for the Tektronix TDS 420 CRO, Hewlett Packard 33120 AFG, Hewlett Packard 34401 DMM, Hewlett Packard 3478 DMM and Fluke 8840 DMM. A series of routines are also available for the Tektronix 8150 Test Interface System. The Borland Pascal language Version 7 is used for these routines. The following chapters provide reference material for the instrument control suite.

2. Basic Instrument Objects

The software control of the devices is organised in an object orientated framework. This enables elements of the device control that are similar to be inherited from common ancestors. As an example all the devices obtain their settings from a CommandSet so all the devices have a common ancestor *TCommandSet.* The instruments controlled via the IEEE 488 Bus have many common features so they are descendants of *TBus* which in turn is a descendant of *TCommandSet.* Using an object orientated framework an efficient controller can be built with common features being inherited from already existing objects (Figure 1). The software for the objects is compiled into a number of units. A listing for the units (Section 9) follows the objects outlined below. The field types used with the objects are declared in the unit listing.
Sections 2 to 9 provide information on the software in this instrument control suite. After examining Figure 1 it is probably most informative to look at the example in Section 10. Examine what the methods in this example are doing by looking up the methods in Sections 2 and 3.

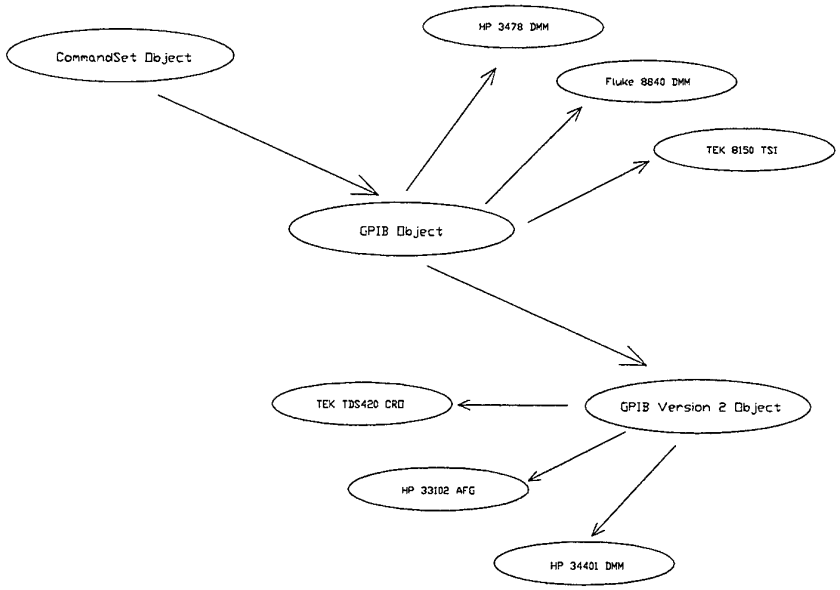

Figure 1 - Object Inheritance

2.1 CommandSet Object

Unit :- Ins_obje

TCommandSet provides the fields and methods that identify each device and its instruction set. Each instruction set can handle four different setups for each instrument.

2.1.1 **Fields**

dev_name : nstring; :-Device Identifier.

commandset : array[1..4, 1..4] of io_string; :- Instruction set, usually set through dialog boxes for each device.

2.1.2 **Methods**

constructor Init(init_dev_name: nstring); :-Initialises object. *init_dev_name* is the device identified.

procedure Zero_CommandSet; :- sets each *commandset* variable to empty.

procedure Read_CommandSet ; :-Reads *commandset* variables from file.

destructor Done; :-Disposes of object.

2.2 GPIB Bus Object

Unit :- Ins_obje

TBus provides fields and methods required by instruments that use the IEEE 488 Bus. *TBus* is a descendent of the *TCommandSet* Object.

2.2.1 **Fields**

pri : integer; :- Primary Address of instrument used by the IEEE 488 Bus.

Statusbyte : byte; :- Status byte as read from instrument.

PollResult : PollArray; :- Status byte in a more user friendly form, boolean array[0..7].

ins_reset : command; :- GPIB Instruction used to reset instrument to default settings.

2.2.2 **Methods**

constructor Init(init_dev_name : nstring; init_ins_reset : command); :-Initialises object. *init_dev_name* is the device identifier, *init_ins_reset* is the instruction used to reset the instrument.

procedure Initialise; :-Determines primary address of instrument on IEEE 488 Bus, *pri.* Resets instrument to default settings using *ins_reset.*

procedure Write_command(temp_command : io_string); :- Writes *temp_command* to instrument at primary address.

procedure Write_CommandSet(i : integer); :-Writes one CommandSet to instrument. *i* selects which of the four CommandSets to write.

procedure Serial_poll; :-Reads status byte from instrument at primary address and places information in *PollResult.*

procedure trig; :- Uses the IEEE 488 GET command to trigger device at primary address.

procedure error; :- Halts program execution (private).

procedure local; :-Returns instrument to local state using IEEE 488 GTL command.

destructor Done; :-Disposes of object.

2.3 GPIB Bus Version 2 Object

Unit :-GPIB2INS

TGPIB2 provides fields and methods suitable for instruments with the IEEE 488.2 Bus. *TGPIB2* uses features from the SCPI Instrument language. *TGPIB2* is a descendent of the *TBus* Object.

2.3.1 **Fields**

Event_Enable : string[2]; :-Stores Enable mask for Standard Event Register.

Status_Enable : string[2]; :- Stores Status mask for Status Register.

2.3.2 **Methods**

procedure Initialise; :-Runs *TBus.Initialise* and sets Status and Standard Event Register masks.

procedure serial_poll; :-Conducts serial poll using SCPI command, stores result in *PollResult.*

procedure QuestionableData; :-Reports data from Questionable Data Register.

procedure StandardEvent; :-Reports data from Standard Event Register.

procedure poll; :-Uses *Serial_poll, QuestionableData* and *StandardEvent* to do a full check on device.

procedure Trigger; :-Triggers device using CPI trigger command.

function Read_Check; :-The SCPI Language ends its messages with a line feed character. If this character is not read an error may occur. This procedure strips the linefeed and subsequent characters. This function is used by other functions when returning results from an instrument.

3. Tektronix TDS420 CRO

Unit :-TEKCRO

TTek_CRO provides fields and methods for use with the Tektronix TDS 420 CRO. This instrument has four channels. *TTek_CRO* is a descendent of the *TGPIB2* Object

Table 1 - Tektronix CRO Object Summary

TCommandSet	**TBus**	**TGPIB2**	**TTekCRO**
Init	**Init**	**Initialise**	**Init**
Zero_CommandSet	Initialise	**Serial_poll**	Sub_Get_Vertical_Scale
Read_CommandSet	Write_Command	QuestionableData	Get_Vert_Scale
Done	Write_CommandSet	StandardEvent	Get_Vert_Offset
	Serial_poll	Poll	Get_Hori_Scale
	trig	Trigger	Get_Trace_Start_Stop
	error		Get_Acquire_Mode
	local		Set_Data_Format
	Done		Start_Acquire
			Acquire_Finished
			Repeat_Until_Acquire_ Finished
			Stop_Acquire
			Sub_Read_Trace
			Read_Trace
			Store_Trace
			NewVertScale
			NewTimeBase
			Return_Trace_Element
			Return_Trace_Start
			Return_Trace_Stop
			Set_Display_Intensity
			Read_CommandSet
			Write_CommandSet
			Done

Note: Bold indicates this method overwrites an inherited method.

3.1 Fields

CH_AllSelect : array[1..4] of SubSelect; :-Records the channels selected by each of the four CommandSets.

CHSelect : SubSelect; :-Records the currently selected channels. Set by *Write_CommandSet.*

TraceFilename : array[0..4] of string[80]; :-Records the filename to store the trace data in for each of the four CommandSets. *TraceFilename[0]* contains the current CommandSets TraceFilename, this is set by *Write_CommandSet.*

Trace_ptr : ^Trace_Record; :-Pointer to variable used for storing the traces from the CRO.

Vert_Scale : array[1..4] of Real; :-Stores vertical scaling factor (volts/point).

Vert_Offset : array[1..4] of Real; :-Stores vertical Offset (volts).

Hori_Scale : real; :-Stores horizontal scaling factor (seconds/point).

Bits_16 : boolean; :-Used to record if CRO output is in 8 or 16 bit mode.

Trace_Start : word; :-Records Trace start position.

Trace_Stop : word; :-Records Trace stop position.

counter : integer; :-Records number of points.

3.2 Methods

Constructor Init(init_dev_name : nstring); :-Initialize object, initialise *Trace_ptr.*

procedure Sub_Get_Vertical_Scale(i : integer); :-Places value of one vertical scale in *Vert_Scale. i* selects which channel to obtain the scale from (Private).

procedure Get_Vert_Scale; :-Uses *Sub_Get_Vertical-Scale* to read the vertical scale from all selected channels.

procedure Get_Vert_Offset; :-Reads the vertical offset from all selected channels. Result stored in *Vert_Offset.*

procedure Get_Hori_Scale; :-Reads the horizontal scale of CRO. Result stored in *Hori_Scale.*

procedure Get_Trace_Start_Stop; :-Reads the start and stop position of the trace. Result placed in *Trace_Start* and *Trace_Stop* respectively.

procedure Get_Acquire_Mode; :-Reads Acquire mode from CRO and determines if the output data needs to be in an 8 or 16 bit format. Sets *Bits_16* to true if output data requires 16 bit format, false if 8 bit format required (private).

procedure Set_Data_Format; :-Determines required data format using *Get_Acquire_Mode.* Sets CRO Data output format as required.

procedure Start_Acquire; :-Triggers CRO to start recording.

procedure Stop_Acquire; :-Stops CRO recording.

function Acquire_Finished; :-Uses the 'Busy?' query to determine if the signal acquisition has completed.

procedure Repeat_Until_Acquire_Finished(max_time : real); :-Repeats until signal acquisition has completed or max_time has expired. *max_time* is the time out period in seconds.

procedure Sub_Read_Trace(i : integer); :-Reads one Trace from the CRO. *i* selects which of the traces to read (Private).

procedure Read_Trace; :-This procedure uses many of the preceding methods to read all the selected traces from the CRO. This procedure will generally be used to read data from the CRO. This procedure performs the following tasks:

- Determines the start and stop positions of the trace using *Get_Trace_Start_Stop.*
- Sets the data output format using *Set_Data_Format.*
-Determines the vertical offset using *Get_Vert_Offset.*
-Determines the vertical scale using *Get_Vert_Scate.*
-Determines the horizontal scale using *Get_Hori_Scale.*
-Reads the trace from the four channels into the memory pointed to by *Trace_ptr* using *Sub_Read_Trace.*

procedure Store_trace; :-This procedure saves the data in the memory pointed to by *Trace_ptr* into the file specified by *TraceFilename.*

procedure NewTimeBase(seconds_per_division: real); :- Sets the time base of the CRO to *seconds_per_division.*

procedure NewVertScale(channel: byte; volts_per_division); :-The vertical scale of one channel of the CRO is set to *volts_per_division.* The channel to be altered is selected by *channel. channel* to have any affect should have a value between 1 an 4 inclusive.

function Return_Trace_Element(channel: byte; i: word) : integer; :-Returns the value measured from the CRO and stored in the memory pointed to by Trace_ptr. *channel* designates the channel of the CRO and *i* the position of the reading.

function Return_Trace_Start : word; :- Returns the position of the first element in a trace.

function Return_Trace_Stop : word; :- Returns the position of the last element in a trace.

procedure Set_Display_Intensity(Percent : integer); :-Used to reduce the brightness of the screen on long running tests when no operator is present.

procedure Read_CommandSet; :- Reads CommandSet using inherited *Read_CommandSet*. Using information in the CommandSet it initialises *CHAllSelect* and *TraceFilename[1..4]*.

procedure Write_CommandSet(i : integer); :- Writes the selected CommandSet (1-4) to the instrument, sets *CHSelect* and *TraceFilename[0]*.

Destructor Done; :-Disposes of *Trace_ptr*, disposes of object.

4. Hewlett Packard 33102 AFG

Unit :-HPAFG

THP_AFG provides fields and methods for use with the Hewlett Packard 33120 AFG. This instrument produces one signal at a time. Either predefined, eg. sin, square etc, or arbitrary, user defined. *THP_AFG* is a descendent of the *TGPIB2* Object.

Table 2 Hewlett Packard AFG Object Summary

TCommandSet	**TBus**	**TGPIB2**	**THP_AFG**
Init	**Init**	**Initialise**	**Init**
Zero_CommandSet	Initialise	**Serial_poll**	NewFreq
Read_CommandSet	Write_Command	QuestionableData	NewAmp
Done	Write_CommandSet	StandardEvent	readARB
	Serial_poll	Poll	loadARB
	trig	Trigger	loadARB_BIN
	error		DetermineTriggerSource
	local		AutoTrigger
	Done		**Write_CommandSet**
			Done

Note: Bold indicates this method overwrites an inherited method.

4.1 Fields

WaveForm_ptr : ^Arb_File; :-Points to structure used to hold points for arbitrary waveform.

WaveFileName : string[80]; :-Stores name of file holding arbitrary waveform information.

ArbFile : text; :-File that holds arbitrary waveform information.

counter : integer; :-Number of points in arbitrary waveform.

trigger_source : string[5]; :-Stores trigger source;

burst_mode : string[3]; :- Stores burst mode;

file_found : boolean; :-If procedure *ReadARB* is unable to find *ArbFile* this field is set to False.

4.2 Methods

Constructor Init(init_dev_name : nstring); :- Initialise object, initialise *WaveForm_ptr.*

procedure Write_CommandSet(i: integer); :-Writes the selected CommandSet (1-4) to the instrument, sets Wave*Filename.*

procedure NewFreq(rate: real); :- Writes a new frequency to AFG. No range checking implemented.

procedure NewAmp(Amplitude: real); :- Writes a new amplitude to AFG. No range checking implemented.

procedure ReadARB; :-Reads arbitrary waveform from *ArbFile* to array pointed to by *WaveFile_ptr.* If unable to find designated file sets *file_found* to false.

procedure loadARB; :- If *file_found* is True loads information in array pointed to by *WaveFile_ptr* into AFG in ASCII format (slow).

procedure loadARB_BIN; :- If *file_found* is True loads information in array pointed to by *WaveFile_ptr* into AFG in binary format (fast).

procedure DetermineTriggerSource; :-Determines if a bus trigger is required. Stores trigger source in *trigger_source* and burst mode in *burst_mode.* A bus trigger is required when the AFG is bus triggered and in burst mode.

procedure AutoTrigger; :-Uses bus trigger if required. *DetermineTriggerSource* must be used before this command so trigger type is known. Use this triggering procedure if the trigger mode is unknown or not yet determined.

Destructor Done; :-Disposes of *WaveForm_ptr,* disposes of object.

5. Hewlett Packard 34401 DMM

Unit :-HP34401

THP_34401 provides fields and methods for use with the Hewlett Packard 34401 DMM. This instrument measures one signal at a time but can be used to record minimums, maximums and averages. *THP_34401* is a descendent of the *TGPIB2* Object.

Table 3 Hewlett Packard 34401 DMM Object Summary

TCommandSet	**TBus**	**TGPIB2**	**THP_34401**
Init	**Init**	**Initialise**	**Init**
Zero_CommandSet	Initialise	**Serial_poll**	**Initialise**
Read_CommandSet	Write_Command	QuestionableData	Read
Done	Write_CommandSet	StandardEvent	OnCalculate
	Serial_poll	Poll	OffCalculate
	trig	Trigger	SetMinMax
	error		MinRead
	local		MaxRead
	Done		AveRead
			CountRead
			trigger
			DetermineTrigger
			Source
			AutoTrigger
			Done

Note: Bold indicates this method overwrites an inherited method.

5.1 Fields

Quest_Enable : string[4]; :-Mask for Questionable Data Register.

trigger_source : string[5]; :- Stores trigger source.

5.2 Methods

Constructor Init(init_dev_name : nstring) :-Initialise object, initialise *Quest_Enable.*

procedure Initialise; :-Runs *TGPIB.Initialise* and sets Questionable Data Register mask.

function Read: io_string; :-Returns a reading from DMM. Reading must have been triggered. Polls device for errors.

procedure SetMinMax; :-Sets DMM so it is able to do Minimum, Maximum and Average functions.

procedure OnCalculate; :-Enables calculation function. Use procedure *SetMinMax* first.

procedure OffCalculate; :-Disables calculation function.

function MaxRead: io_string; :-Returns Maximum reading from DMM. Readings must have been triggered. Polls device for errors.

function MinRead: io_string; :-Returns Minimum reading from DMM. Readings must have been triggered. Polls device for errors.

function AveRead: io_string; :-Returns Average reading from DMM. Readings must have been triggered. Polls device for errors.

function CountRead: io_string; :- Returns number of readings in Maximum, Minimum or Average reading from DMM. Polls device for errors.

procedure Trigger(bus_trigger : integer); :- Triggers DMM reading. If *bus_trigger* $\neq 0$ uses GPIB GET to provide trigger from bus.

procedure DetermineTriggerSource; :-Determines if a bus trigger is required. Stores trigger source in *trigger_source*. A bus trigger is required when the DMM is in bus triggered mode.

procedure AutoTrigger; :- Uses bus trigger if required. *DetermineTriggerSource* must be used before this command so that the trigger type is known. Use this triggering procedure if the trigger mode is unknown.

6. Hewlet Packard 3478 DMM

unit :- HP3478

THP_3478 provides fields and methods for use with the Hewlett Packard 3478 DMM. This instrument measures one value at a time. *THP_3478* is a descendent of the *TBus* Object

Table 4 Hewlett Packard 3478 DMM Object Summary

TCommandSet	**TBus**	**THP_3478**
Init	**Init**	**Init**
Zero_CommandSet	Initialise	Read
Read_CommandSet	Write_Command	trigger
Done	Write_CommandSet	poll
	Serial_poll	**error**
	trig	
	error	
	local	
	Done	

Note: Bold indicates this method overwrites an inherited method.

6.1 Methods

Constructor Init(init_dev_name : nstring); :-Initialise object.

function Read: io_string; :-Returns a reading from DMM. Reading must have been triggered. Polls device for errors. Checks for over value error.

procedure Trigger; :-Triggers DMM reading.

procedure poll; :-Uses result from serial poll to report on errors

procedure error; :-Reports over value errors (Private).

7. Fluke 8840 DMM

unit :- F8840

TF_8840 provides fields and methods for use with the Fluke 8840 DMM. This instrument measures one value at a time. *TF_8840* is a descendent of the *TBus* Object.

Table 5 Fluke 8840 DMM Object Summary

TCommandSet	**TBus**	**TF_8840**
Init	**Init**	**Init**
Zero_CommandSet	Initialise	Read
Read_CommandSet	Write_Command	trigger
Done	Write_CommandSet	poll
	Serial_poll	**error**
	trig	
	error	
	local	
	Done	

Note: Bold indicates this method overwrites an inherited method.

7.1 Methods

Constructor Init(init_dev_name : nstring); :-Initialise object.

function Read: io_string; :-Returns a reading from DMM. Reading must have been triggered. Polls device for errors. Checks for over value error.

procedure Trigger; :-Triggers DMM reading.

procedure poll; :-Uses result from serial poll to report on errors

procedure error; :-Reports over value errors (Private).

8. Tektronix 8150 TSI

unit :- TEKTSI

TTek_TSI provides fields and methods for use with the Tektronix 8150 Test System Interface with the low level scanner card. The test system provides the capability to multiplex up to 60 channels. No dialog boxes are provided for this instrument as in normal usage it is easier to cycle through the switches using a simple loop with the methods provided in this object. These routines only use the TSI in the immediate mode. *TTek_TSI* is a descendent of the *TBus* Object.

Table 6 Tektronix 8150 TSI Object Summary

TCommandSet	**TBus**	**TTek_TSI**
Init	**Init**	**Init**

Zero_CommandSet	Initialise	**Initialise**
Read_CommandSet	Write_Command	poll
Done	Write_CommandSet	CloseSwitch
	Serial_poll	OpenSwitch
	trig	CloseAll
	error	OpenAll
	local	**error**
	Done	

Note: Bold indicates this method overwrites an inherited method.

8.1 Methods

Constructor Init(init_dev_name : nstring) :-Initialise object.

procedure Initialise; :-Initialises instrument, sets trigger mode.

procedure poll; :-Calls error procedure on abnormal bit.

procedure CloseSwitch(card: string3; switch: string3); :-Closes the selected switch on the designated card. *Card* is set to the value *Card[i]* where i has a value of 1-3. *Switch* is set to a value *SwitchA[i]* or *SwitchB[i]* where i has a value of 1-10. The constant arrays *card, SwitchA* and *SwitchB* are defined in this unit.

procedure OpenSwitch(card: string3; switch: string3); :-Opens the selected switch on the designated card. See *CloseSwitch* for details on variables.

procedure CloseAll(card: string3); :-Closes all the switches on the designated card. See *CloseSwitch* for details on variables.

procedure OpenAll(card: string3); :-Opens all the switches on the designated card. See *CloseSwitch* for details on variables.

9. Units

The software for controlling the instruments is in a series of units. Listed below are the contents of these units including the other units they use, variable types, objects and some of the procedures and functions (non object). Listed under the uses heading are the non standard Pascal units, see the source code for a full listing.

9.1 Variable.pas

This unit provides commonly used variable types to the other units.

const :-maxibuf = $FF;

```
    minibuf = $20;
    Extra_String_Length = 20;

Type :- nstring = string[7];
    io_buf = array[1..maxibuf] of char;
    command = minibuf;
    long_command = string[maxibuf];
```

9.2 Message.pas

This unit provides dialog boxes for information to be written to the screen.

uses :-Variable;

procedure Write_Message(Line_One, Line_Two : io_string); :-Writes *Line_One* and *Line_Two* to screen. Dialog box closed by operator.

procedure Write_Message_Three(Line_One, Line_Two, Line_Three : io_string); :-Writes *Line_One, Line_Two,* and *Line _Three* to screen. Dialog box closed by operator.

function Get_Reply(Line_One : io_string) : boolean; :-Writes *Line_One* to screen, returns operators response to a True / False query.

procedure Open_Message(Line_One, Line_two : io_string); :-Writes *Line_One* and *Line_Two* to screen. Dialog box closed by *Close_Message.*

procedure Update_Message(Line_One, Line_Two : io_string); :-Writes *Line_One* and *Line_Two* to dialog box created by *Open_Message,* overwrites previous *Line_One* and *Line_Two.*

procedure Close_Message; :-Closes dialog box created by *Open_Message.*

9.3 tpdecl.pas

This unit contains software routines provided with the National Instruments GPIB card. See Manual for further information[1].

9.4 GPIBwrt.pas

This unit provides basic GPIB read and write procedures. These procedures are developed from the National Instrument procedures provided in tpdecl.pas.

```
uses :- Variable;
        Message;
        tpdecl;
```

function find_pri(name : nstring) : integer; :-Determines primary address of instrument on GPIB bus and returns address as integer. *name* is the device identifier.

procedure ibwrite(addr1 : integer; incomm : command); :-Writes instruction in *incomm* to instrument on the GPIB address *addr1* after removing any spaces from *incomm.*

procedure ibwrites(addr1 : integer; incomm : command); :-Writes instruction in *incomm* to instrument on the GPIB address *addr1.*

procedure ibwrite_long(addr1 : integer; long_incomm : long_command); :-Writes instruction in *long_incomm* to instrument on the GPIB address *addr1.*

procedure ibread(addr1 : integer; var data : io_string; count : integer); :-Reads *count* characters from instrument at GPIB address *addr1* into *data.*

9.5 Ins_Obje.pas

This unit contains a data structure used by the instrument control objects and two instrument control objects. A procedure for determining the presence of a file is also included.

```
uses :- Variable;
    GPIBwrt;
    tydecl;
    message;

type :- PollArray = array[0..7] of boolean;

Objects :- TCommandSet;
          TBus(TCommandSet);
```

function FileExists(FileName : io_string) : boolean; :-Checks existence of file of string as designated by *FileName.*

9.6 GPIB2INS.pas

This unit provides software for use with instruments with a IEEE 488.2 Bus.

```
const :- Operation_Complete : string[4] = '*OPC';
        Clear_Serial_Poll : string[4] = '*CLS';

uses :- Variable;
       tpdecl;
       GPIBwrt;
       ins_obje;
```

object :- TGIPB2(TBus);

9.7 TekCRO.pas

This unit provides software directly concerned with operation of the Tektronix CRO.

```
uses :- Variable;
    tpdecl;
    GPIBwrt;
    ins_obj;
    GPIB2INS;

type :- Trace_Record; :-See source code for details;
        SubSelect : array[1..4] of Boolean;

objects :- TTek_CRO(TGPIB2);
```

9.8 HPAFG.pas

This unit provides software directly concerned with operation of the Hewlett Packard AFG.

```
uses :- Variable;
        tpdecl;
        GPIBwrt;
        ins_obj;
        GPIB2INS;

const :-  Arb_array_size = 8191;

type :-  Arb_array = array[0..Arb_array_size] of integer;

objects :- HP_AFG(TGPIB2);
```

9.9 HP34401.pas

This unit provides software directly concerned with operation of the Hewlett Packard 34401 DMM.

```
uses :- Variable;
        tpdecl;
        GPIBwrt;
        ins_obj;
        GPIB2INS;

const :-  array_size = 100;
```

objects :- HP_34401(TGPIB2);

9.10 HP3478.pas

This unit provides software directly concerned with operation of the Hewlett Packard 3478 DMM.

uses :- Variable;
tpdecl;
GPIBwrt;
ins_obj;

objects :- HP_3478(TBus);

9.11 F8840.pas

This unit provides software directly concerned with operation of the Fluke 8840 DMM.

uses :- Variable;
GPIBwrt;
ins_obj;

objects :- TF_8840(TBus);

9.12 TekTSI.pas

This unit provides software directly concerned with operation of the Tektronix 8150 Test Interface System.

uses :- Variable;
tpdecl;
GPIBwrt;
ins_obj;

const :- card : array[1..3] of string3 = ('F1 ', 'F2 ', 'F3 ');
switchA[1..10] of string3 = ('A1', 'A2',.....,'A9', A10');
switchB[1..10] of string3 = ('B1', 'B2',.....,'B9', B10');

objects :- TTek_TSI(TBus);

10. Example

The following is an example of a program that uses the object *TTek_CRO* to set up a Tektronix CRO to acquire the selected traces and record them to file. The settings of the CRO may be changed using the dialog boxes without changing the program.

```
unit CROTest;

interface

uses crt, gpibwrt, variable, message, TekCRO;

procedure Tek_CRO_Test(init_dev_name : nstring);

implementation
{————————————————————————}

procedure Tek_CRO_Test(init_dev_name : nstring);
    {**** TESTS COMMANDS WRITTEN TO THE Tek CRO****}

var
 i, j : integer;
 Tek_CRO_ptr : PTek_CRO;                { POINTER TO INSTANCE OF  OBJECT }
 temp_string_one : io_string;
 temp_string_two : io_string;

begin
    {CREATES INSTANCE OF OBJECT TTek_CRO}
 Tek_CRO_ptr :=New(PTek_CRO, init(init_dev_name)):
 Open_Message('Initilising CRO','');
{INSTRUMENT RETURNED TO DEFAULT SETTINGS, GPIB ADDRESS FOUND}
 Tek_CRO_ptr^.Initialise;
 Tek_CRO_ptr^.Read_CommandSet;          {READ ALL COMMANDSETS}
   {CommandSet[1] WRITTEN TO INSTRUMENT}
 Tek_CRO_ptr^.Write_CommandSet(1);
 Tek_CRO_ptr^.poll;                     {CHECK INSTRUMENT}
 Update_Message('Acquiring Data','');
 Tek_CRO_ptr^.Start_Acquire;            {START ACQUIRING DATA}
 delay(2000);                           {WAIT WHILE DATA OBTAINED}
 Tek_CRO_ptr^.Stop_Acquire;             {STOP ACQUIRING DATA}
 Tek_CRO_ptr^.Read_Trace;               {READ TRACE FROM CRO}
 Tek_CRO_ptr^.poll;                     {CHECK INSTRUMENT}
 Update_Message('Storing Data','');
 Tek_CRO_ptr^.StoreTrace;               {SAVE DATA TO TRACEFILENAME[0]}
```

```
 Tek_CRO_ptr^.Local;                    {RETURN CRO TO FRONT PANEL
CONTROL}
 Dispose(Tek_CRO_ptr, Done);            {DISPOSE OF OBJECT}
 Close_Message;
end;
{-------------------------------------------}
end.
```

Examples of programming other instruments are available in the procedures used to test the commands sent to the instruments from the dialog boxes. (AFGTest.pas and DMMTest.pas)

11. Dialog Boxes

The dialog boxes provide the means to enter device settings. Each device has from one to four dialog boxes, depending on the number of parameters required by the device. After checking the settings derived from the dialog boxes for errors, a conversion of the data into a string format occurs. This string contains the instructions for the instrument. The CommandSet file for the specified device is then modified to allow for the new commands. Instances and descendants of *TCommandSet* may read the CommandSet file using the *Read_CommandSet* method. A facility to test the commands in the CommandSet file for each individual instrument independent of any major test routine exists on the main menu.

The following section is an outline of the dialog boxes used in the instrument controller. An understanding of this section is not necessary to use the instrument controller only to enter new instruments and their dialog boxes. All the dialog boxes in this controller are descendants of the Turbo Vision object *TDialog*. The object *TManager* is used as the base on which to construct the objects that handle the Dialog boxes and the checking and saving of the data obtained from the dialog boxes. Objects from the Turbo Vision package are the basis for this menu system. Below is an outline of *TManager*.

11.1 TManager

Unit :-Dia_Obje

This object provides the basic managers used by the Instrument Screens to obtain, check and save Instrument Instructions.

11.1.1 Fields

dev_name : nstring; :-Device identifier.

screen_no : byte; :-Indicates if this is the 1..4 dialog box for the device. Used in determining where the settings will be stored in the CommandSet.

save_position; :-Determines if the Instrument Instructions will be saved in CommandSet 1..4.

Command_Line; :-Holds the Instrument Instructions to be saved by *Save_CommandSet*. *Command_Line* is obtained from *Record_to_String*.

11.1.2 Methods

constructor init(xscreen : byte; init_dev_name : nstring); :-Initialises object, *screen_no* and dev_name. *xscreen* is used to set *screen_no* and *init_dev_name* to set *dev_name*.

function record_to_string : io_string; **virtual**; :-An empty routine that is over written by each descendent. In the descendent objects this function converts record information returned by the dialog boxes into string information to be saved on file by *save_CommandSet*.

procedure save_CommandSet; **virtual**; :-Writes commands to CommandSet file.

destructor Done; **virtual**; :-Disposes of object.

The software for each dialog box is in a separate unit. Below is a list of the instrument type and the corresponding unit.

Table 7 - Dialog Box Software Units

Instrument	Object	Unit
Tek TDS 420 CRO	TCRODialog_One(Tmanager)	CRODial1
Tek TDS 420 CRO	TCRODialog_Two(Tmanager)	CRODial2
Tek TDS 420 CRO	TCRODialog_Three(Tmanager)	CRODial3
Tek TDS 420 CRO	TCRODialog_Four(Tmanager)	CRODial4
HP 33120A AFG	TFunction_Manager(TManager)	HP_AFGD1
HP 33120A AFG	TParameter_Manager(TManager)	HP_AFGD2
HP 33120A AFG	TArbitrary_Manager(TManager)	HP_AFGD3
HP 34401 DMM	THP34401_Manager(TManager)	HP34401D
HP 3478 DMM	THP3478_Manager(TManager)	HP3478D
Fluke 8840 DMM	TF8840_Manager(TManager)	F8840D

Each of the objects listed above has a pointer to the object of the same name except a "P" replaces the first letter "T".

In addition to the object, the units contain code to write the input boxes (ie. radio buttons, check boxes and input lines) onto the dialog boxes. It is necessary to read the

Turbo Vision Programming Guide[2] to understand the code that creates the input boxes.

In the interface section of each unit there is one procedure, this procedure when called creates the dialog box for the designated device type with the specified identifier. Listed below are the procedures that create dialog boxes and the instrument types the dialog box is for.

Table 8 - Dialog Box Procedures

Instrument	**Procedure**	**Unit**
Tek TDS 420 CRO	CRODialog_screenOne(init_dev_name: nstring)	CRODial1
	CRODialog_screenTwo(init_dev_name: nstring)	CRODial2
	CRODialog_screenThree(init_dev_name: nstring)	CRODial3
	CRODialog_screenFour(init_dev_name: nstring)	CRODial4
HP 33120A AFG	HP_AFGDialog_Function(init_dev_name: nstring)	HP_AFGD1
	HP_AFGDialog_Parameter(init_dev_name: nstring)	HP_AFGD2
	HP_AFGDialog_Arbitrary(init_dev_name: nstring)	HP_AFGD3
HP 34401 DMM	HP34401Dialog(init_dev_name: nstring);	HP34401D
HP 3478 DMM	HP3478Dialog(init_dev_name: nstring);	HP3478D
Fluke 8840	F8840Dialog(init_dev_name: nstring);	F8840D

In order to create the second Tek TDS 420 CRO Dialog box, for a Tek TDS 420 CRO with a device identifier "DEV1" the following line of code would be written

```
CRODialog_ScreenTwo("DEV1");
```

After calling this code, provided the user did not cancel the box or abandon the data, the command file for "DEV1" will be updated to the new settings.

12. Main Menu

The file Insmenue.pas contains the software that provides the menus and desktop that the other units use. The menu controls the creation of the dialog boxes that enter information into the CommandSets and the execution of test procedures. The desktop

provides the surface where the dialog boxes and messages are displayed. The Insmenue program is a product created from Turbo Vision package. As such it is necessary to refer to the Turbo Vision programming guide [2] to follow the code. Help is available for most commands by using the F1 key.

13. Reference

1. NI-488 Functions for Turbo Pascal. July 1990 Edition. National Instruments Corporation. Part Number 320014-12.

2. Turbo Vision Version 2.0. Programming Guide. Borland International, 1992

Appendix A Software Design Information

This appendix shows a data flow diagram that was used in the prototype of the control sweet for the first instrument , the Tektronix TDS420 CRO.
Although variable names and object names have changed the data flow and structure remain the same.

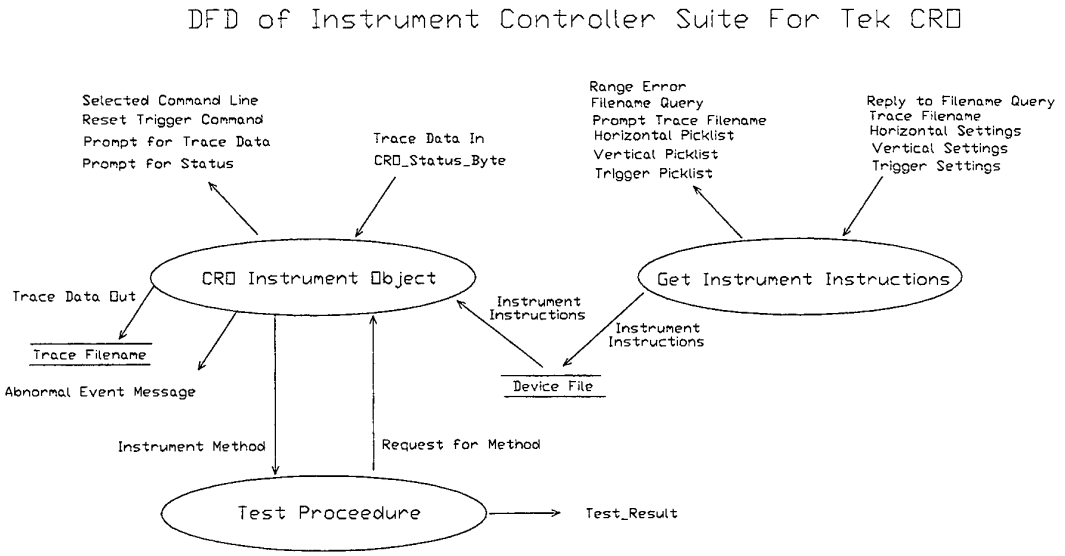

Figure A1 - Data flow diagram for Tek TDS 420 CRO

DISTRIBUTION LIST

Software Instrument Control Suite

David Clarke

AUSTRALIA

DEFENCE ORGANISATION

S&T Program
Chief Defence Scientist
FAS Science Policy
AS Science Corporate Management
} shared copy
Director General Science Policy Development
Counsellor Defence Science, London (Doc Data Sheet)
Counsellor Defence Science, Washington (Doc Data Sheet)
Scientific Adviser to MRDC Thailand (Doc Data Sheet)
Director General Scientific Advisers and Trials/Scientific Adviser Policy and Command (shared copy)
Navy Scientific Adviser
Scientific Adviser - Army (Doc Data Sheet and distribution list only)
Air Force Scientific Adviser
Director Trials

Aeronautical and Maritime Research Laboratory
Director
Chief of Maritime Operations Division
D. Richardson
F. May
A. Theobald
B. Jessup
David Clarke

DSTO Library
Library Fishermens Bend
Library Maribyrnong
Library Salisbury (2 copies)
Australian Archives
Library, MOD, Pyrmont
Library, MOD, HMAS Stirling

Capability Development Division
Director General Maritime Development : Att DD Mine Warfare
Director General Land Development (Doc Data Sheet only)
Director General C3I Development (Doc Data Sheet only)

Navy

SO Science, Maritime HQ, MHQ (Doc Data Sheet only)
MWSCPD
CO HMAS Waterhen

Army

ABCA Office, G-1-34, Russell Offices, Canberra (4 copies)
SO (Science), DJFHQ(L), MILPO Enoggera, Queensland 4051 (Doc Data Sheet only)
NAPOC QWG Engineer NBCD c/- DENGRS-A, HQ Engineer Centre Liverpool Military Area, NSW 2174 (Doc Data Sheet only)

Intelligence Program

DGSTA Defence Intelligence Organisation

Corporate Support Program (libraries)

OIC TRS, Defence Regional Library, Canberra
Officer in Charge, Document Exchange Centre (DEC), 1 copy
*US Defence Technical Information Center, 2 copies
*UK Defence Research Information Centre, 2 copies
*Canada Defence Scientific Information Service, 1 copy
*NZ Defence Information Centre, 1 copy
National Library of Australia, 1 copy

UNIVERSITIES AND COLLEGES

Australian Defence Force Academy
Library
Head of Aerospace and Mechanical Engineering
Senior Librarian, Hargrave Library, Monash University
Librarian, Flinders University

OTHER ORGANISATIONS

NASA (Canberra)
AGPS

OUTSIDE AUSTRALIA

ABSTRACTING AND INFORMATION ORGANISATIONS

INSPEC: Acquisitions Section Institution of Electrical Engineers
Library, Chemical Abstracts Reference Service
Engineering Societies Library, US
Materials Information, Cambridge Scientific Abstracts, US
Documents Librarian, The Center for Research Libraries, US

INFORMATION EXCHANGE AGREEMENT PARTNERS

Acquisitions Unit, Science Reference and Information Service, UK
Library - Exchange Desk, National Institute of Standards and Technology, US

SPARES (10 copies)

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

Software Instrument Control Suite

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR(S)

David Clarke

5. CORPORATE AUTHOR

Aeronautical and Maritime Research Laboratory
PO Box 4331
Melbourne Vic 3001

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-GD-0156	AR-010-355	General Document	October 1997

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
510/207/0788		DGMD	26	2

13. DOWNGRADING/DELIMITING INSTRUCTIONS

14. RELEASE AUTHORITY

Chief, Maritime Operations Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

computer software, instrumentation, programming

19. ABSTRACT

The use of computers to control instrumentation can provide improvements in quality, quantity and turn around time of work carried out by a laboratory. These improvements must be balanced against the time taken to write the programs that control the instruments. This work documents a library of instrument control routines used to facilitate the task of programming and to enable the full advantage of computer controlled instrumentation to be realised.