WINTRANS

GRAPHICAL USER INTERFACE LIBRARY
FOR
TRANSPUTER SYSTEMS

VERSION 1.0

FEBRUARY 1995

Dear User,

the following document describes WinTrans library, which is a toolset for constructing graphical user interfaces for Transputer systems. To get an overview of the functionality two demonstrations demo.bah and demo2.bah are supplied in wintrans.zip. It was originally programmed as a graduation work by one of my students, Markus Schmidtner. WinTrans consists of two parts: A dynamic link library for use together with WIServer and a collection of routines for the Transputer to control the DLL. This reference mainly describes the Transputer procedures, which are coded in Occam. Furthermore an interested reader gets advice in how to extend the functionality of WinTrans Version 1.0.
Since I am currently in the last year of my post graduate studies, I do not have plenty of time. This is why the translation of this reference manual was done in quite a hurry. I apologise for the mistakes and lack of style.
Another point results from this: I am not able of providing any support for this product. The source code of any part of WinTrans is provided with this package. Feel free to recompile it for adapting to Your needs. Please contact me if You extended the functionality. I will try to keep track of the changes and collect them for a new release.
This software may be freely distributed including the source code, provided that the original authors are mentioned and no fee is asked for distribution.

> The author of this program accepts no responsibility for damages resulting from the use of this product and makes no warranty or representation, either express or implied, including but not limited to, any implied warranty of merchantability or fitness for a particular purpose. This software is provided "AS IS", and You, its user, assume all risks when using it.

Best regards

Robert Westendorp

Email: west@emt.e-technik.tu-muenchen.de
or: 74462,2731@compuserve.com

1 Introduction 4

1.1 Loading and removing the DLL 5

1.2 Programming the interface (Transputer) 6

2 Applying the module library 8

2.1 General call (from Transputer) 8

2.2 Inside WinTrans DLL 9

2.3 Base level procedures 10

2.3.1 create.control 10
2.3.2 set.status 12
2.3.3 send.data 12
2.3.4 update.control 13
2.3.5 poll.control 13
2.3.6 destroy.control 14

2.4 14

2.4 Additional functions and tools 15

2.4.1 Real32 floating point support 15
2.4.2 Loading and removing the DLL 15
2.4.3 Setting the object status 15
2.4.4 Sending display data 16
2.4.5 Polling user input 17
2.4.6 Getting specific object status 18
2.4.7 Message box (msg.box) 20
2.4.8 Removing all objects (close.all) 20
2.4.9 Termination 21

2.5 Single modules 23

2.5.1 Window 23
2.5.2 Lamp 25
2.5.3 Switch 25
2.5.4 Pushbutton 26
2.5.5 Checkbox 27
2.5.6 Radiobuttons 28
2.5.7 Scrollbar 30
2.5.8 Textinput 31
2.5.9 Textoutput 32
2.5.10 Textbox 33
2.5.11 Bargraph 34
2.5.12 Digital display 35
2.5.13 Knob 36

2.5.14 Graphic display 37
2.5.15 Switchable Bitmap 39
2.5.16 Oscilloscope 40
2.5.17 Digital multimeter 42
2.5.18 Filter control 43
3 Extending the library 44
3.1 Adding object modules 44
3.2 Improvements 45
4 Demonstration 47
4.1 Preparations 47
4.2 demo.bah 47
4.3 demo2.bah 49
References 50
Appendix A Procedure declarations in 'wintrans.occ' 51
Appendix B Contents of the parameter array 59

1 Introduction

WinTrans is the Implementation of a comfortable, clear and intuitively operable graphical user interface for Transputer systems. Microsoft Windows™ was chosen as a platform because of its powerful graphical features. User interaction under MS Windows is known to many users from other applications, so the usage of standard controls avoids the necessity of adapting to a new interface.
WinTrans is universally useful in the construction of measurement and automation applications. Some controls feature the displaying of data, others enable user input and process control.
Using WinTrans the Transputer is freed from the demands of graphical user interaction. Visualisation and handling of the controls is managed by the PC. Complicated tasks are kept away from the Transputer system, for user actions are registered inside the PC and announced in a standardised way. Refreshing of displays is fast and flicker free, because only the data is transferred, not graphical information.
The interface was programmed in a very structured way, so that extending it is not a big effort. On side of the Transputer, many procedures have been developed that enable easy usage of all controls. The programmer of a measurement application for a Transputer system has full access to all controls through calls to specialised functions for each module. He does not need any knowledge of Windows programming.
For easier reading and better differentiation a nomenclature is used in this manual. Names of **variables**, **procedures** and important **catchwords** are **printed fat**. Paragraphs containing Transputer programs use `Courier New`. C++ programs use the font `Fixedsys`.
The project was realised using WIServer, a Transputer server written by Mike Morgan. It supports MS Windows as a platform and supports the Transputer with commands for opening text windows and sending information to them. In addition there is a possibility of extending the functionality of WIServer by using Dynamic Link Libraries (DLL). The procedures for doing this are:

- **load.dll**
- **exec.dll**
- **unload.dll**

1.1 Loading and removing the DLL

WinTrans DLL is loaded by the procedure

```
PROC load.dll (CHAN OF SP          from.isv, to.isv,
               VAL []BYTE          dll.filename,
               [DLL.DATA.SIZE]BYTE dll.data,
               BYTE                isv.result)
```

The DLL is removed by sending a data packet using

```
PROC unload.dll (CHAN OF SP          from.isv, to.isv,
                 [DLL.DATA.SIZE]BYTE dll.data,
                 BYTE                isv.result)
  [MAX.PACKET.SIZE]BYTE packet:
  INT pos:
  INT16 dll.handle:
  INT32 dll.proc.addr:

PROC exec.dll (CHAN OF SP              from.isv, to.isv,
               VAL [DLL.DATA.SIZE]BYTE dll.data,
               VAL []BYTE              params,
               []BYTE                  data,
               INT                     data.len,
               BYTE                    isv.result)
[MAX.PACKET.SIZE]BYTE packet:
INT pos:
INT16 dll.handle:
INT32 dll.proc.addr:
```

This is the procedure to call the DLL procedure The first two parameters are the server protocol channels **from.isv** and **to.isv** The array **dll.data** was defined in `load.dll` It is used for identifying the DLL procedure The array **params** can be defined by the user and takes parameters for the DLL procedure The result of the Server call is written into **data** Its size is **data.len** Bytes The following sections of Occam code show a sample call to WinTrans Library

```
IF
    isv.result = SPR.OK
      SEQ
        pos := 0
        push.byte (packet, pos, SP.EXEC.DLL)
```

Here the filling of the arrays **packet** is started The procedure **push.byte** from „wiserver lib“ puts one Byte with the value **SP.EXEC.DLL** into the array at position **pos.** By doing this **pos** is incremented

```
        []BYTE b RETYPES dll.handle:
        b := [dll.data FROM 0 FOR 2]
        []BYTE b RETYPES dll.proc.addr:
        b := [dll.data FROM 2 FOR 4]
```

The seven Bytes of **dll.data** are divided into **dll.handle** and **dll.proc.addr** by retyping them

```
        push.int16 (packet, pos, dll.handle)
        push.int32 (packet, pos, dll.proc.addr)
        push.string (packet, pos, params)
        adjust.packet (packet, pos)
```

These values are also packed into the array and finally **params** is concatenated **push.string** automatically places a two byte length in front of the string The procedure **adjust.packet** ensures, that the packet size fits to protocol conventions The readily prepared array is shown in figure

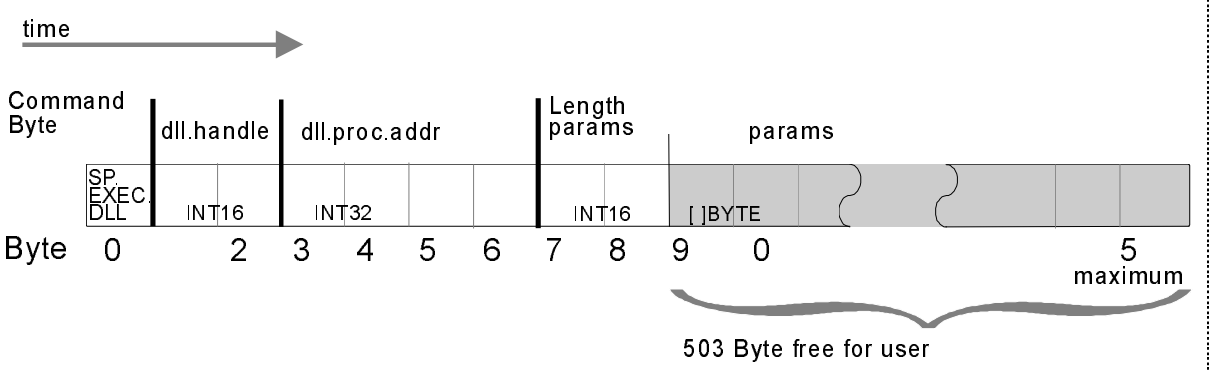


Figure 1.1: Data array to server

```
to.isv ! INT16 pos::packet
```

This line sends the packet to the server The first two bytes **pos** describe the length of the following **packet**

```
INT16 pos.int16:
        from.isv ? pos.int16::packet
```

Now the process is waiting for the answer to be sent The length is returned in **pos.int16**

```
        pos := 0
        pop.byte (packet, pos, isv.result)
        pop.string (packet, pos, data, data.len)
    TRUE
      SKIP
:
```

The first two bytes of the answer string take the result value **isv.result** The remaining 5 byte have to take a string with preceding length **data.len** bytes of that string are copied into **data** by the procedure **pop.string** So 509 bytes are free for taking the result of the DLL procedure call

1.2 Programming the interface (Transputer)

For user interaction the contents of the array **params** is interpreted by the DLL procedure

All communication procedures towards the DLL are based on **exec.dll** on the Transputer side Therefore every procedure needs the parameters **from.isv**, **to.isv**, **dll.data** and **isv.result.** Thus they are globally defined in the application program

On base level there are some routines that set a small part of the parameter field They are universally useful and not specific for some objects other, higher level routines enable the parameterizing of a whole object in one procedure call

The file 'wintrans occ' contains all procedures Constants are kept in 'wintrans inc' 'wintrans occ' was compiled to wintrans tah, so the following lines have to be included into source code:

```
#INCLUDE "wintrans.inc"
#USE     "wintrans.tah"
```

2 Applying the module library

2.1 General call (from Transputer)

Figure .1 shows the procedure hierarchy.

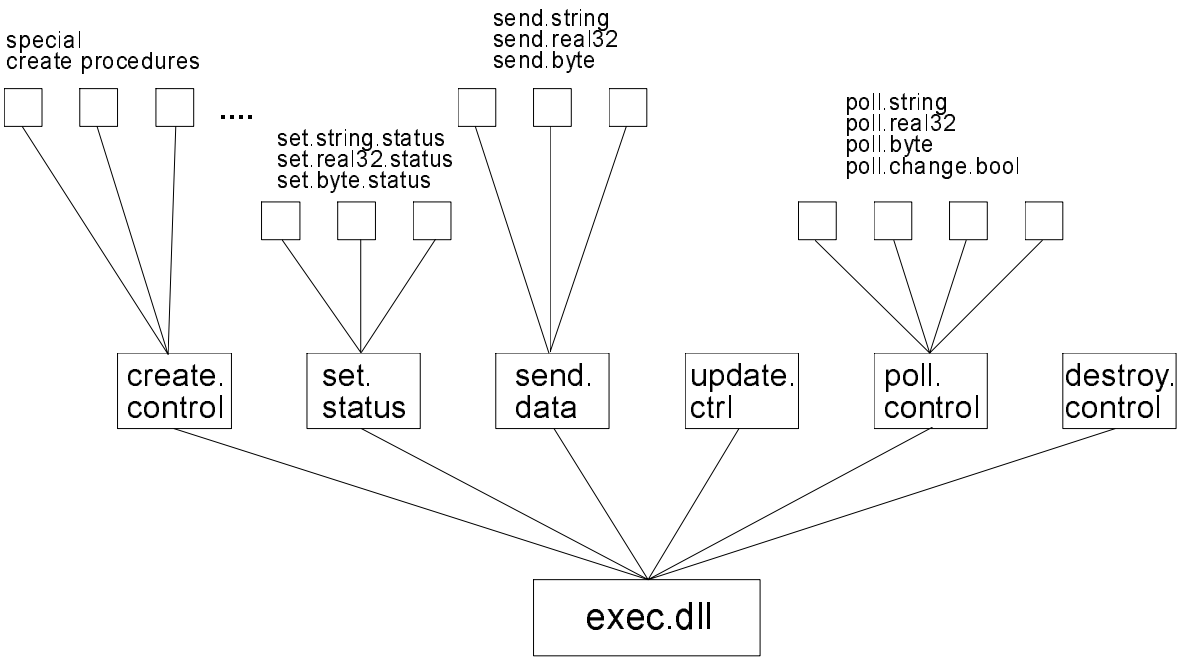


Figure 2.1: Procedure hierarchy

The usage of a module from WinTrans Library is very easy and is described below:

1. Creating an object:
   With object specific **Create** procedures an object can be displayed on screen. All necessary parameters are provided in the procedure call. For later identification of the object, WinTrans DLL sends a handle for each created object to the Transputer. This 4 byte Integer (declared as **control.handle**) has to be used in any subsequent call to WinTrans DLL.
. Setting the status
   Every object that can be changed by the user -like a switch for example- and is not only for displaying data, may be adjusted by **set.status**. If the call is omitted, the default settings are used.
3. Polling the status
   The routine **poll.control** gives the Transputer access to user input. The DLL sends an answer packet, that represents the status of the object.
4. Sending data
   With **send.data** display data is sent to an object. This is stored inside the object class but the display is not refreshed. It is possible to send multiple

packets to an object that is able to display large data arrays without redrawing it every time.

5. Display refresh
   Data sent to an object is redrawn with the procedure **update.ctrl**. The object is refreshed by a Windows Redraw message.
6. Deleting an object
   An object is deleted by **destroy.control**. it is removed from the screen and allocated memory is freed. This has to happen before program termination on the Transputer. It may be used to change the layout of the user interface. For closing all objects at once the procedure **close.all** is useful.

2.2 Inside WinTrans DLL

After receiving a data packet with the command tag SP.EXEC.DLL from the Transputer, WIServer calls the procedure defined by dll.handle and dll.proc.addr. It is the procedure **Link**, that takes a **pointer** to the data packet as a parameter.

```
void FAR PASCAL _export Link(HINSTANCE hInstance, HWND hWnd,
                             PLINKSTRING pointer)
```

TLINKSTRING is a structure type declared in 'global.h' that takes the data packet.

```
struct TLINKSTRING
{
   int  size;
   unsigned char data[512];
};
```

Data is a copy of the array **params** sent by the Transputer. The number of bytes is stored in **size.** Data items may be accessed by dereferenciating with the * and -> operators.

```
BYTE befehl;
BYTE tag;
befehl= *(pointer->data);
tag   =*((pointer->data)+1);
```

The first two bytes determine the action that has to be executed. This is what the two variables **befehl** and **tag** are used for. The following combinations exist:

befehl	tag	Operation	Chapter
0	0	msg.box	.4.7
0	96	allow.no.exit	.4.9
0	97	close.all	.4.8
0	98	changed.something	.4.5
0	99	poll.exit	.4.5
1	TYPE	create (Object of type TYPE)	.3.1
		set.status	.3.
3		poll.control	.3.5
4		send.data	.3.3
5		update.ctrl	.3.4
6		destroy.control	.3.6

2.3 Base level procedures

All objects may completely be controlled using the base level procedures. For doing that the parameter array has to be constructed 'manually'. The specialised functions are much easier to use and based upon these procedures.
The server channels, dll.data and isv.result are no longer mentioned in the text, because they are present in all procedure calls and have been explained above.

2.3.1 create.control

PROCEDURE CALL

```
PROC create.control(    VAL BYTE art,
                        VAL INT x.pos, y.pos,
                        VAL INT breite, hoehe,
                        VAL INT32 parent,
                        VAL []BYTE parameter,
                        INT32 ctrl.hdl,
                        CHAN OF SP fs, ts,
                        VAL [DLL.DATA.SIZE]BYTE dll.data,
                        BYTE isv.result)
```

DESCRIPTION

An object of type **art** is created and displayed on the screen. Position and size are declared according to the window referred by **parent**. The **parameter** array contains object specific data. This is put into **params** before it is sent to the server. A handle to the object is returned in **ctrl.hdl**.

PARAMETERS

params (to.server) = data (DLL)

```
VAL BYTE art
```

art may be one of the constants described in „wintrans inc“

```
VAL BYTE SCHALTER           IS 1 (BYTE) :
VAL BYTE LAMPE              IS 2 (BYTE) :
VAL BYTE MULTIMETER         IS 3 (BYTE) :
VAL BYTE OSZI               IS 4 (BYTE) :
VAL BYTE FILTER             IS 5 (BYTE) :
VAL BYTE FENSTER            IS 6 (BYTE) :
VAL BYTE BUTTON             IS 7 (BYTE) :
VAL BYTE CHECKBOX           IS 8 (BYTE) :
VAL BYTE RADIOBUTTON        IS 9 (BYTE) :
VAL BYTE SCROLLBAR          IS 10(BYTE) :
VAL BYTE TEXTIN             IS 11(BYTE) :
VAL BYTE TEXTOUT            IS 12(BYTE) :
VAL BYTE BAR                IS 13(BYTE) :
VAL BYTE DIGITAL            IS 14(BYTE) :
VAL BYTE DREHKNOPF          IS 15(BYTE) :
VAL BYTE ZWEIBIT            IS 16(BYTE) :
VAL BYTE PIXWIN             IS 17(BYTE) :
VAL BYTE TEXTBOX            IS 18(BYTE) :
```

```
VAL INT x.pos, y.pos
VAL INT breite, hoehe
```

Coordinates relative to the parent window All values are given in screen pixels (A common user screen has 6 0x 80 or 10 x768 pixels) Values increase from the upper left to the lower right corner The origin (upper left corner) of an object has to be determined relative to the upper left corner of the working field of the parent window The working field is smaller than its window because of the border and title bar
Objects with fixed size ignore **breite** and **hoehe** as width and height

```
VAL INT32 parent
```

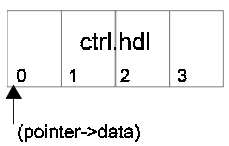
The handle of the parent window into which the object will be placed If its value is `INT32 0` (Null), the parent window is the main window of the DLL

```
VAL []BYTE parameter
```

Objectspecific data The array has to be filled by the user

RESULTS

data (DLL) = data (from.server)




```
INT32 ctrl.hdl
```

The DLL returns a handle to the newly generated object Later an object may only be referred by this handle

2.3.2 set.status

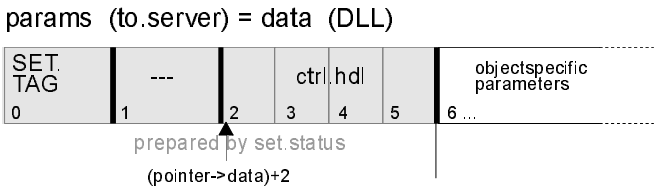
PROCEDURE CALL

```
PROC set.status(    VAL INT32 ctrl.hdl,
                    VAL []BYTE parameter,
                    CHAN OF SP fs, ts,
                    VAL [DLL.DATA.SIZE]BYTE dll.data,
                    BYTE isv.result)
```

DESCRIPTION

The status of the object identified by **ctrl.hdl** is set with data contained in **parameter**

PARAMETERS



`VAL INT32 ctrl.hdl`

The object handle returned by **create.control.**

`VAL []BYTE parameter`

Objectspecific data individually prepared for each object type

RESULTS

None

2.3.3 send.data

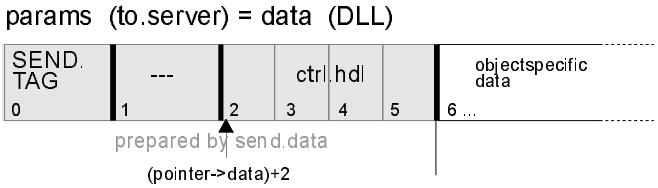
PROCEDURE CALL

```
PROC send.data   (VAL INT32 ctrl.hdl,
                  VAL []BYTE parameter,
                  CHAN OF SP fs, ts,
                  VAL [DLL.DATA.SIZE]BYTE dll.data,
                  BYTE isv.result)
```

DESCRIPTION

In **parameter** display data is sent to the object identified by **ctrl.hdl.**

PARAMETERS



```
VAL INT32 ctrl.hdl,
```

The object handle returned by **send.data.**

```
VAL []BYTE parameter
```

Objectspecific data individually prepared for each object type

RESULTS
None

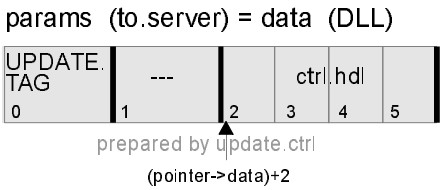
2.3.4 update.control

PROCEDURE CALL

```
PROC update.ctrl  ( VAL INT32 ctrl.hdl,
                    CHAN OF SP fs, ts,
                    VAL [DLL.DATA.SIZE]BYTE dll.data,
                    BYTE isv.result)
```

DESCRIPTION
The object identified by **ctrl.hdl** is redrawn

PARAMETERS




```
VAL INT32 ctrl.hdl
```

The object handle returned by **update.control.**

RESULTS
None

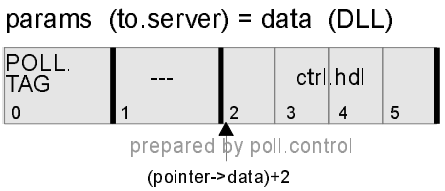
2.3.5 poll.control

PROCEDURE CALL

```
PROC poll.control (  VAL INT32 ctrl.hdl,
                     BOOL changed,
                     []BYTE status,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)
```

DESCRIPTION
The object identified by **ctrl.hdl** is checked for changes by the user Actual data is returned by **status Changed** is TRUE, if the object was changed since the last poll

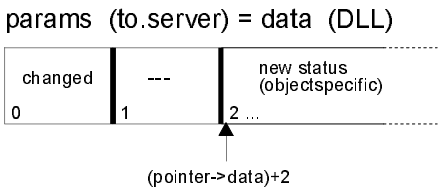
PARAMETERS




```
VAL INT32 ctrl.hdl
```

The object handle returned **poll.control**

RESULTS




```
BOOL changed
```

If the polled object was changed by the user, the first byte contains 1, if not 0 is transferred The procedure sets changed to TRUE or FALSE accordingly

```
[]BYTE status
```

This array contains the value of the user input

2.3.6 destroy.control

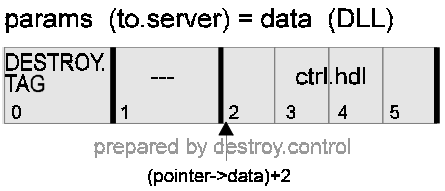
PROCEDURE CALL

```
PROC destroy.control ( VAL INT32 ctrl.hdl,
                       CHAN OF SP fs, ts,
                       VAL [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result )
```

DESCRIPTION

The object **ctrl.hdl** is deleted

PARAMETERS




```
VAL INT32 ctrl.hdl
```

The object handle returned **destroy.control**

RESULTS

None

2.4 Additional functions and tools

2.4.1 Real32 floating point support

PROC push.real32

```
PROC push.real32([]BYTE buffer,
                 INT pos,
                 VAL REAL32 value)
```

The procedure **push.real32** writes a Byte floating point value at position **pos** into the array **buffer pos** is incremented by

PROC pop.real32

```
PROC pop.real32([]BYTE buffer,
                INT pos,
                REAL32 value)
```

The procedure **pop.real32** reads a Byte floating point value at position **pos** from the array **buffer** into the parameter **value pos** is incremented by

2.4.2 Loading and removing the DLL

PROC DLL.laden

```
PROC DLL.laden(VAL []BYTE filename,
               CHAN OF SP fs, ts,
               [DLL.DATA.SIZE]BYTE dll.data,
               BYTE isv.result)
```

The DLL called **filename** is loaded into PC memory The identification of the DLL is returned in **dll.data.** This routine is based on the WIServer procedure load dll

PROC DLL.entfernen

```
PROC DLL.entfernen(CHAN OF SP fs, ts,
                   [DLL.DATA.SIZE]BYTE dll.data,
                   BYTE isv.result)
```

The DLL identified by **dll.data** is removed and memory released This routine is based on the WIServer procedure unload dll

2.4.3 Setting the object status

The following procedures are used for setting common status parameters in an object

PROC set.byte.status

```
PROC set.byte.status (VAL INT32 control.handle,
                      VAL BYTE  byte,
                      CHAN OF SP fs, ts,
                      VAL [DLL.DATA.SIZE]BYTE dll.data,
                      BYTE isv.result)
```

Sends the status **byte** to an object referenced with **control.handle** The following object may get their status in one Byte:
S (5 3), Checkbox (5 5) , Radiobutton (5 6) and switchable Bitmap (5 15)
Constants: ON=1, OFF=0

PROC set.real32.status

```
PROC set.real32.status (VAL INT32  control.handle,
                        VAL REAL32 real,
                        CHAN OF SP fs, ts,
                        VAL [DLL.DATA.SIZE]BYTE dll.data,
                        BYTE isv.result)
```

Sets the object defined by **control.handle** to the floating point value **real** The following objects may be parametrized by that:
Scrollbar (5 7), Knob (5 13)

PROC set.string.status

```
PROC set.string.status(VAL INT32   control.handle,
                       VAL []BYTE  string,
                       CHAN OF SP  fs, ts,
                       VAL [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result)
```

Sends the string with preceding length to **control.handle** Only one object can take a string status: Textin (5 8)

2.4.4 Sending display data

The following procedures transfer commonly used data for displaying to objects

PROC send.byte

```
PROC send.byte( VAL INT32  control.handle,
                VAL BYTE   byte,
                CHAN OF SP fs, ts,
                VAL [DLL.DATA.SIZE]BYTE dll.data,
                BYTE isv.result)
```

A Byte **byte** is sent to the object identified by **control.handle**
This Byte may switch the Lamp (5)

PROC send.real32

```
PROC send.real32( VAL INT32  control.handle,
                  VAL REAL32 real,
                  CHAN OF SP fs, ts,
                  VAL [DLL.DATA.SIZE]BYTE dll.data,
                  BYTE isv.result)
```

A floating point value **real** is sent to the object identified by **control.handle**
The following objects display a byte floating point value:
Ba (5 11), Digital (5 1) and Digital multimeter (5 17)

PROC send.string

```
PROC send.string( VAL INT32  control.handle,
                  VAL []BYTE string,
                  CHAN OF SP fs, ts,
                  VAL [DLL.DATA.SIZE]BYTE dll.data,
                  BYTE isv.result)
```

A string **string** is sent to the object identified by **control.handle**
A string may be displayed in Text (5 9) or added as a selection in Textbox (5 10)

2.4.5 Polling user input

The Transputer system has to poll the DLL for any user input The procedure changed something was developed to avoid the necessity of polling every single input object that might be manipulated by the user

PROC changed.something

```
PROC changed.something(BOOL som.ch,
                       CHAN OF SP fs, ts,
                       [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result)
```

DESCRIPTION
The global flag **something_changed** inside the DLL is tested by the procedure The result is returned in **som.ch**

PARAMETERS
None

RESULTS

```
BOOL  som.ch
```

TRUE, if the user changed any input object on the screen, FALSE if not
If TRUE is returned, every relevant object has to be polled to get source and data of the input

2.4.6 Getting specific object status

The following procedures are implemented to get the object status in commonly used data formats

PROC poll.byte

```
PROC poll.byte  (VAL INT32  control.handle,
                 BOOL       changed,
                 BYTE       byte,
                 CHAN OF SP fs, ts,
                 VAL [DLL.DATA.SIZE]BYTE dll.data,
                 BYTE isv.result)
```

DESCRIPTION

A status that can be represented using a byte **byte** is returned The object is identified by **control.handle.** If the object was changed since the last call **changed** is set to TRUE

PARAMETERS

```
VAL INT32  control.handle
```

The handle defined for that object by **create.control.**

RESULTS

```
BOOL       changed
```

Is TRUE if the object has been changed since the last call

```
BYTE       byte
```

The status represented as a Byte
The procedure may be used with S (5 3), Checkbox (5 5), Radiobutton (5 6) and switchable Bitmaps (5 15)

PROC poll.real32

```
PROC poll.real32 (VAL INT32  control.handle,
                  BOOL       changed,
                  REAL32     real,
                  CHAN OF SP fs, ts,
                  VAL [DLL.DATA.SIZE]BYTE dll.data,
                  BYTE isv.result)
```

DESCRIPTION

A status that can be represented using a real value **real** is returned The object is identified by **control.handle.** If the object was changed since the last call **changed** is set to TRUE

PARAMETERS

```
VAL INT32  control.handle
```

The handle defined for that object by **create.control.**

RESULTS

```
BOOL        changed
```

Is TRUE if the object has been changed since the last call

```
REAL32      real
```

The status represented as a REAL3
May be used for Scrollbar (5 7) and Knob (5 13)

PROC poll.string

```
PROC poll.string (VAL INT32  control.handle,
                  BOOL       changed,
                  []BYTE     string,
                  CHAN OF SP fs, ts,
                  VAL [DLL.DATA.SIZE]BYTE dll.data,
                  BYTE isv.result)
```

DESCRIPTION

A status that can be represented using a string **string** is returned The object is identified by **control.handle.** If the object was changed since the last call **changed** is set to TRUE

PARAMETERS

```
VAL INT32  control.handle
```

The handle defined for that object by **create.control.**

RESULTS

```
BOOL        changed
```

Is TRUE if the object has been changed since the last call

```
 []BYTE     string
```

The status represented as a Byte
May be used for Textin (5 8) and Textbox (5 10)

PROC poll.change.bool

```
PROC poll.change.bool (VAL INT32  control.handle,
                       BOOL       changed,
                       CHAN OF SP fs, ts,
                       VAL [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result)
```

DESCRIPTION

A status that can be represented using a bool value **bool** is returned The object is identified by **control.handle.**
The return value **changed** is TRUE if the object was manipulated after the last call No special status is returned This is useful for objects without any special status like the Pushbutton (5)

PARAMETERS

```
VAL INT32  control.handle
```

The handle defined for that object by **create.control.**

RESULTS

```
BOOL       changed
```

Is TRUE if the object has been changed since the last call

2.4.7 Message box (msg.box)

PROC msg.box

```
PROC msg.box(         VAL []BYTE string1,
                      VAL []BYTE string2,
                      VAL []BYTE string3,
                      CHAN OF SP fs, ts,
                      [DLL.DATA.SIZE]BYTE dll.data,
                      BYTE isv.result)
```

DESCRIPTION

A small window is displayed on screen, containing a message to the user It is closed if ENTER or a OK button is pressed The message box is a modal dialogue As an effect of that no other object can be manipulated while this window is on screen

PARAMETERS

```
VAL []BYTE string1
VAL []BYTE string2
VAL []BYTE string3
```

These three strings will be displayed in three lines There is a maximum of 395 characters for all lines together

RESULTS

None

2.4.8 Removing all objects (close.all)

PROC close.all

```
PROC close.all(CHAN OF SP fs, ts,
               [DLL.DATA.SIZE]BYTE dll.data,
               BYTE isv.result)
```

DESCRIPTION

In advance of a new screen layout or the termination of an application all objects have to be removed from screen an deleted inside the DLL

A comfortable way to do this is provided by **close.all.** Thus it is not necessary to close every single object and no mistake may occur by forgetting an object or closing a parent window prior to its child

PARAMETERS
None

RESULTS
None

2.4.9 Termination

There are two ways of terminating an application
1 The application is terminated by the Transputer when the program has finished The user is not allowed to exit the application
The user closes the main window
The WinTrans user interface for a Transputer application is running as a usual task under MS Windows Therefore it can be terminated by using the control field in the window title bar, by pressing Alt-F or by using the task manager Sometimes however the user is not intended to exit the application - especially in automation applications Moreover an error occurs, if the Transputer tries to send data to an object, that does no longer exist, because the main window was closed
For realising method 1 the procedure **allow.no.exit** was created

PROC allow.no.exit

```
PROC allow.no.exit(CHAN OF SP fs, ts,
                   [DLL.DATA.SIZE]BYTE dll.data,
                   BYTE isv.result)
```

DESCRIPTION
The procedure disables the user from closing the main window
The menu item File is greyed out and is not selectable The entry 'Exit' in the System menu of the main window is replaced by 'No Exit' and greyed out

INPUT
None

OUTPUT
None
If method is implemented, the Transputer has to poll the existence of the main window before sending output to the DLL For doing that **poll.exit** was written

PROC poll.exit

```
PROC poll.exit(BOOL buttonvalue,
               CHAN OF SP fs, ts,
               [DLL.DATA.SIZE]BYTE dll.data,
               BYTE isv.result)
```

DESCRIPTION

The procedure delivers information about the state of the main window in **buttonvalue.** TRUE means, the window has been closed by the user
In addition no more procedures are executed inside the DLL if this value is TRUE This prevents errors that may occur when the Transputer system sends data to an object while the user closes the main window on the PC
Status information gathered by **poll.control** is invalid after that point
Because of the disadvantages of the second method mentioned above, the following procedure is recommended if user exit should be possible:
According to method 1 closing of the main window is prohibited On the user screen a additional push-button is created, that the user should use for exiting the application By polling this button the Transputer system will take the necessary steps to terminate the application

GENERAL REMARK

WIServer is not able of removing a DLL from memory when it is terminated So before exiting the server, the DLL has to be removed from memory with **DLL.entfernen.** If this is not done a second attempt to load the DLL results in a segmentation error that can only be fixed by rebooting the PC

2.5 Single modules

This section describes each single object Every object is realised as a class in C++ These classes are build from two files whose names sound similar to the module name The file with the h ending contains the class declaration, the one with the cpp ending takes the implementation Every object is displayed during construction After the execution of the create method the module is on screen
The following parameters are identical to every object:

```
VAL INT x, y, b, h
```

x and **y** tell the position of the upper left corner of the object relative to the upper left corner of the working area of the enclosing window in pixel units
Width and height of an object are defined in **b** and **h** respectively Some modules do not need these values for they are of fixed size

```
VAL INT32  parent.handle
```

This is the handle of a window that acts as a parent window for the created object If it is set to 0, the main window is referred to Every handle can be used many times to group objects in one window together

```
INT32 control.handle
```

Every **Create** procedure returns the handle to the module just created

```
CHAN OF SP fs, ts,
VAL [DLL.DATA.SIZE]BYTE dll.data,
BYTE isv.result
```

The channels to and from the server, the DLL identification and the rolling error flag are explained above They are no longer mentioned in the following descriptions

2.5.1 Window

DESCRIPTION
A window is used for grouping and separating the childobjects placed into it The colour can be freely chosen Different types of windows feature various handling and displaying options

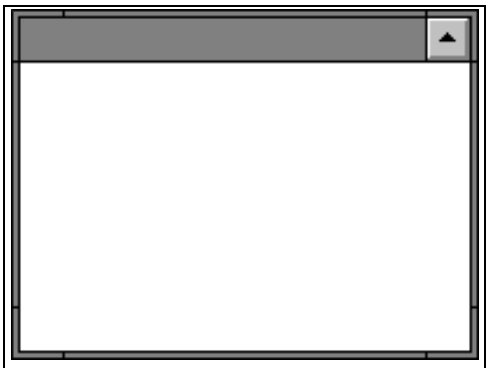


Abb. 2.2: Window

CREATION

```
PROC create.fenster (VAL INT    x, y, b, h,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     VAL INT16  art,
                     VAL BYTE   rot, gruen, blau,
                     VAL []BYTE fenstertext,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)
```

```
VAL INT16 art
```

The following window types are implemented:

art=0 not resizeable, not movable

This window has no title bar and the frame is only one pixel thick It keeps its size even when it is a child object of a resizeable window and this one is resized

art=1 not resizeable, movable

The window can be moved with its title bar The size is fixed

art=2 resizeable, not movable, automatic size

The window has no title bar and cannot be moved, but it may be resized Furthermore it adapts its size to the parent window

art=3 resizeable, movable, automatic size

The window has a title bar It may be moved and resized It adapts its size to the parent window A maximise button enlarges it to the size of the parent window

art=4 not resizable, movable, automatic size

Equal to type 0 with the difference of automatic sizing by the parent window The user can not directly resize it, but it may be resized by changing the parent window

art=5 not resizable, movable, automatic size

Equal to type 1 with the difference of automatic resizing by the parent window

```
VAL BYTE  rot, gruen, blau
```

Here the colour of the working area of the window is defined in parts of red, green and blue The principle of additive colour mixing is used (black 0,0,0; white , ,)

```
VAL []BYTE   fenstertext
```

This line of text is displayed in the title bar For windows without title bar it is of no relevance Its maximum length is 0 characters

DATA HANDLING

For this object only the procedures **create** and **destroy** exist

IMPLEMENTATION
Class: Cfenster (fenster h, fenster cpp) derived from CFrameWnd
Defaults: b=100, h=100

2.5.2 Lamp

DESCRIPTION
A lamp can display two states It may be switched on and off by sending byte values

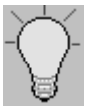


Abb. 2.3: LAMP

CREATION

```
PROC create.lampe   (VAL INT    x, y, b, h,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)
```

The appearance of the lamp is fixed No further parameters have to be supplied

DATA HANDLING
The lamp is used for displaying data It is able to receive one byte, so it is controlled using **send.byte.** Valid values of **byte** are:
0 = OFF: Lamp off,
1 = ON: Lamp on
The new value is displayed after a call to the routine **update.ctrl.**

IMPLEMENTATION
Class: CLampe (lampe h, lampe cpp) derived from CFrameWnd
Defaults: b=40, h= 0, off

2.5.3 Switch

DESCRIPTION
The switch is an object with two states that may be changed by the user It can be switched by a mouse click The user can receive the state in a byte value

Abb. 2.4: SWITCH

CREATION

```
PROC create.schalter  (VAL INT x, y, b, h,
                       VAL INT32  parent.handle,
                       INT32 control.handle,
                       CHAN OF SP fs,ts,
                       VAL [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result)
```

The appearance of the switch is fixed No further parameters have to be supplied

DATA HANDLING

A switch is used to control a state A state may be set using **set.status** A user input is polled using **poll.status** The routines **set.byte.status** and **poll.byte** can be used, because the state of the switch is stored in a byte

IMPLEMENTATION

Class:CSchalter (bitswitch h, bitswitch cpp) derived from CWnd
Defaults: b=40, h= 0
Default status: OFF

2.5.4 Pushbutton

DESCRIPTION

A pushbutton only registers, if it was pressed The button field can be labelled

Abb. 2.5: PUSHBUTTON

CREATION

```
PROC create.pushbutton (VAL INT x, y, b, h,
                        VAL INT32  parent.handle,
                        INT32 control.handle,
                        VAL []BYTE buttontext,
                        CHAN OF SP fs,ts,
                        VAL [DLL.DATA.SIZE]BYTE dll.data,
                        BYTE isv.result)
```

A standard Windows pushbutton is created

```
VAL []BYTE buttontext
```

In **buttontext** a label with a maximum length of 30 characters can be supplied

DATA HANDLING

Internally the pushbutton only registers a user action It only delivers the information that anything has happened No special status is stored or can be set

For recognising a user input the procedure **poll.change.bool** can be used

IMPLEMENTATION
Class:CPush (push h, push cpp) derived from CWnd
Defaults: b=7 , h=

2.5.5 Checkbox

DESCRIPTION
The checkbox is a Windows standard control with two states A rectangle can be selected by a mouse click The status of the Checkbox may be transferred in a Byte

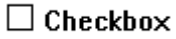


Abb. 2.6: CHECKBOX

CREATION

```
PROC create.checkbox (VAL INT x, y, b, h,
                      VAL INT32  parent.handle,
                      INT32 control.handle,
                      VAL BYTE   textfarbe,
                      VAL []BYTE buttontext,
                      CHAN OF SP fs,ts,
                      VAL [DLL.DATA.SIZE]BYTE dll.data,
                      BYTE isv.result)
```

`VAL BYTE textfarbe`

This parameter chooses the text colour of the box Where `textfarbe` means:

textfarbe=0	black,
textfarbe=1	white,
textfarbe=	red,
textfarbe=3	green,
textfarbe=4	blue,
textfarbe=	yellow

`VAL []BYTE buttontext`

As with Pushbutton a maximum of 30 characters for labelling may be transferred in **buttontext.**
The control automatically chooses the colour of the parent window as a background colour

DATA HANDLING
A Checkbox is used to control a state A state may be set using **set.status** A user input is polled using **poll.status** The routines **set.byte.status** and **poll.byte** can be used, because the state of the Checkbox is stored in a Byte

IMPLEMENTATION
Class:CCheck (check h, check cpp) derived from CWnd
Defaults: b=100, h=
Defaultstatus: not selected (no cross)

2.5.6 Radiobuttons

DESCRIPTION
A group of Radiobuttons can take different states Only one or none of the buttons can be pressed Radiobuttons are Windows standard controls The status is stored in a Byte Implementations from two up to six buttons are implemented It is not recommended to use Radiobuttons for selecting among more than six choices

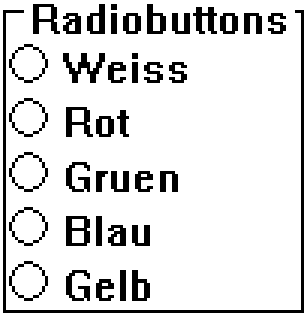


Abb. 2.7: RADIOBUTTON

CREATION

```
PROC create.2.radiobutton (VAL INT    x, y, b, h,
                           VAL INT32  parent.handle,
                           INT32      control.handle,
                           VAL BYTE   text.color,
                           VAL []BYTE header,
                           VAL []BYTE string1,
                           VAL []BYTE string2,
                           CHAN OF SP fs,ts,
                           VAL [DLL.DATA.SIZE]BYTE dll.data,
                           BYTE isv.result)
```

For three up to six buttons a further set of procedures exists They differ in the number in the procedure name and the number of strings that have to be supplied

```
PROC create.6.radiobutton (VAL INT    x, y, b, h,
                           VAL INT32  parent.handle,
                           INT32      control.handle,
                           VAL BYTE   text.color,
                           VAL []BYTE header,
                           VAL []BYTE string1,
                           VAL []BYTE string2,
                           VAL []BYTE string3,
                           VAL []BYTE string4,
                           VAL []BYTE string5,
                           VAL []BYTE string6,
                           CHAN OF SP fs,ts,
                           VAL [DLL.DATA.SIZE]BYTE dll.data,
                           BYTE isv.result)
```

`VAL BYTE text.color`

This parameter takes the colour of the buttons The scheme is the same as for the Checkbox
(0=black, 1=white, =red, 3=green, 4=blue, =yellow)

`VAL []BYTE header`

This string is displayed as a headline in the surrounding rectangle (30 characters maximum)

`VAL []BYTE string1` to `string6`

According to the number of buttons up to six strings for labelling have to be provided (30 characters maximum)

DATA HANDLING

A group of Radiobuttons is used to control a state A state may be set using **set.status** A user input is polled using **poll.status** The routines **set.byte.status** and **poll.byte** can be used, because the state of the Radiobuttons is stored in a byte The value represents the number of the button pressed (0=no button, 1=upper button, = second button)

IMPLEMENTATION

Class:CRadio (radio h, radio cpp) derived from CWnd
Defaults: b=100, h=(button count+1)*18
Defaultstatus: Statusbyte =0, no button selected

2.5.7 Scrollbar

DESCRIPTION

With a scrollbar the user can enter a floating point value Upper and lower limits can freely be defined A small separate window shows the actual value It may also be used to edit the value with the keyboard The position of this field is definable The user can choose between vertical and horizontal types The status is stored in a 4-byte floating point value

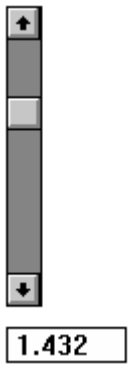


Abb. 2.8: SCROLLBAR

CREATION

```
PROC create.scrollbar.h (VAL INT    x, y, b, h,
                         VAL INT32  parent.handle,
                         INT32      control.handle,
                         VAL INT    dx.zahl,dy.zahl,
                         VAL REAL32 minimum,maximum,
                         VAL INT    teilung,
                         CHAN OF SP fs,ts,
                         VAL [DLL.DATA.SIZE]BYTE dll.data,
                         BYTE isv.result)
```

The creating procedure for a horizontal bar ends with a **.h**, the one for the vertical type with **.v**

```
PROC create.scrollbar.v (VAL INT    x, y, b, h,
                         VAL INT32  parent.handle,
                         INT32      control.handle,
                         VAL INT    dx.zahl,dy.zahl,
                         VAL REAL32 minimum,maximum,
                         VAL INT    teilung,
                         CHAN OF SP fs,ts,
                         VAL [DLL.DATA.SIZE]BYTE dll.data,
                         BYTE isv.result)
```

```
VAL INT    x, y, b, h
```

As **b** is always the width of the object from left to right, it is the width of a vertical Scrollbar and the length of a horizontal one The true width of a Scrollbar is limited to a minimum of 7 and a maximum of 0 pixels to ensure a good appearance The length is unlimited

```
VAL INT    dx.zahl,dy.zahl
```

These two integers describe the position of the entry field in pixels relative to the upper left corner of the bar

```
VAL REAL32 minimum,maximum
```

These two floating point values take the upper and lower limit of the values that can be entered by the Scrollbar **Minimum** has to be mathematically smaller than **maximum**

```
VAL INT    teilung
```

Teilung is the setting into how many steps the bar is divided, thus controlling the granularity of user input (minimum 1) By pressing the arrow buttons the value is increased or decreased by one step By clicking into the dark greyed area the value is changed by 10 steps It is always possible to enter precise values by using the numerical entering field

DATA HANDLING

A Scrollbar is used to control a state A state may be set using **set.status** A user input is polled using **poll.status** The routines **set.real32.status** and **poll.real32** can be used, because the state of the Scrollbar is stored in a 4 byte floating point value

IMPLEMENTATION

Class:CScroll (scroll h, scroll cpp) derived from CWnd
Default size Scrollbar: None
Fixed size numerical field: b=60, h=18
Default status: Arithmetic middle value

2.5.8 Textinput

DESCRIPTION

With Textinput one line of text can be entered by the user The entry is terminated with ENTER The left button cancels the input The text is stored in a string

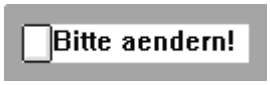


Abb. 2.9: TEXTINPUT

CREATION

```
PROC create.textin     (VAL INT    x, y, b,
                        VAL INT32  parent.handle,
                        INT32      control.handle,
                        CHAN OF SP fs,ts,
                        VAL [DLL.DATA.SIZE]BYTE dll.data,
                        BYTE isv.result)
```

```
VAL INT   x, y, b
```

The height and appearance of the object is fixed Only position and width have to be supplied

DATA HANDLING

The state of a Textinput is determined by user input A state may be set using **set.status** A user input is polled using **poll.status** The routines **set.string.status** and **poll.string** can be used, because the state of the Textinput is stored in a string
The maximum amount of characters is 100

IMPLEMENTATION

Class:CTextin (textin h, textin cpp) derived from CWnd
Defaults: b=1 0, h=19 (height is fixed)
Default status: Empty

2.5.9 Textoutput

Abb. 2.10: TEXTOUTPUT

DESCRIPTION

The user interface is object oriented and the user is not allowed to write text directly into a window A special object is supplied for displaying one line of text It offers different colours and background colour is adapted to the parent window

CREATION

```
PROC create.textout  (VAL INT    x, y, b, h,
                      VAL INT32  parent.handle,
                      INT32      control.handle,
                      VAL BYTE   textfarbe,
                      CHAN OF SP fs, ts,
                      VAL [DLL.DATA.SIZE]BYTE dll.data,
                      BYTE isv.result)
```

```
VAL INT    x, y, b, h,
```

The text size is determined with **h** The object automatically selects a font of appropriate size If the width **b** is too small the string is truncated

```
VAL BYTE   textfarbe
```

Text colour is the same as for the Checkbox
(0=black, 1=white, =red, 3=green, 4=blue, =yellow)

DATA HANDLING

Textout is an object for displaying data These are supplied with **send.data** The string is immediately displayed without a call to **update.ctrl** The user can send the string with **send.string** One line is limited to 6 characters

IMPLEMENTATION

Class:CTextout (textout h, textout cpp) derived from CWnd
Defaults: None

Default: Empty

2.5.10 Textbox

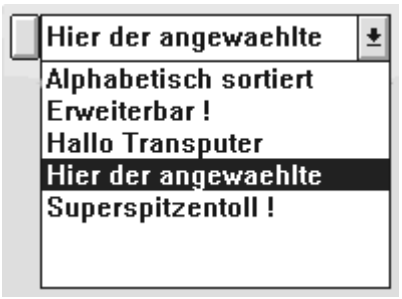


Abb. 2.11: TEXTBOX

DESCRIPTION

The Textbox offers the user a list of strings from which he can select one The list may be edited or extended by him A click on the arrow button opens a box with the list A selection may then be done using the mouse or cursor keys The selection process may be cancelled using the left button This object is comparable with the Windows Combobox

CREATION

```
PROC create.textbox (VAL INT    x, y, b,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)
```

```
VAL INT    x, y, b
```

Only the position an width are recognised during creation Width b tells the horizontal extension including the two buttons The selection line is always 3 pixel high, the Listbox is 1 0 pixels high

DATA HANDLING

After creation of the box, every entry has to be sent as a single string The entries are sorted alphabetically By preceding a string with a @ it is displayed as the first selection If no such string is sent, the selection box is left empty The strings may be delivered using **send.string** based on **send.data.** For fetching the selected string **poll.string** based on **poll.control** can be used

IMPLEMENTATION

Class:CTextbox (textbox h, textbox cpp) derived from CWnd
Defaults: b=100, fixed height

2.5.11 Bargraph

The different Bargraphs offer the visualisation of floating point values

Four different types are implemented:

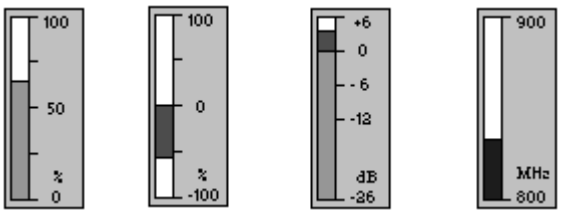


Abb. 2.12: Bargraphs

- Linear positive: A value between zero and a maximum is displayed as a percentage in green
- Linear symmetric: The scale reaches from a negative to a positive value The value is displayed in percentages of a reference value between -100% and +100% Negative values are displayed red
- Logarithmic: The logarithmic proportion $10 * \log \frac{\text{data value}}{\text{reference}}$ in decibel is displayed This scale is often used in electrotechnical or audio applications The scale reaches from - 6dB to +6dB Values greater than 0dB are painted red
- Linear userdefined: Minimum, maximum and labelling are entered by the user The bar is blue

CREATION:

```
PROC create.bar  ( VAL INT    x, y, b, h,
                   VAL INT32  parent.handle,
                   INT32      control.handle,
                   VAL BYTE   art,
                   VAL REAL32 referenzwert,
                   CHAN OF SP fs,ts,
                   VAL [DLL.DATA.SIZE]BYTE dll.data,
                   BYTE isv.result)
```

This procedure is implemented for the first three versions

```
VAL BYTE   art
```

With **art** one of the following types is selected:

art = 0: Linear positive

art = 1: Linear symmetric

art = : Logarithmic - 6dB to +6dB

```
VAL REAL32 referenzwert
```

This value, which has to be positive, contains the ±100% values for type 0 and 1 For type it takes the reference value A Proportion of 1:1 is 0dB

```
PROC create.bar.scale( VAL INT    x, y, b, h,
                       VAL INT32  parent.handle,
                       INT32      control.handle,
                       VAL REAL32 minimum, maximum,
                       VAL []BYTE textmin, textmax, einheit,
                       CHAN OF SP fs,ts,
                       VAL [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result)
```

This procedure creates the user definable bar graph

`VAL REAL32 minimum, maximum`

Here the `minimum` for the bottom and the `maximum` for the top is defined Minimum has to be smaller than maximum

`VAL []BYTE textmin, textmax, einheit`

These three strings label the graph Only the first three letters are important **Textmin** is displayed at the bottom, **textmax** at the top of the bar **Einheit** gives the units of the displayed data

DATA HANDLING

The Bargraph is used for displaying a floating point value For sending data **send.real32** may be used A value is displayed using **update.ctrl**

IMPLEMENTATION

Class:CBar (bar h, bar cpp) derived from CWnd
Breite: b=40 fixed
Default height: h=100

2.5.12 Digital display

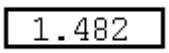


Abb. 2.13: DIGITAL DISPLAY

DESCRIPTION

The digital display shows a floating point value in letters

CREATION

```
PROC create.digital  (VAL INT    x, y, b, h,
                      VAL INT32  parent.handle,
                      INT32      control.handle,
                      CHAN OF SP fs,ts,
                      VAL [DLL.DATA.SIZE]BYTE dll.data,
                      BYTE isv.result)
```

The appearance of the digital display is fixed No further parameters have to be supplied

DATA HANDLING

A single floating point value is sent using **send.data** or **send.real32.** For updating the display **update.ctrl** has to be used

IMPLEMENTATION
Class:CDigital (digital h, digital cpp) derived from CWnd
Defaults: b=80, h= 0
Default value: 0 000

2.5.13 Knob

The knob is a user control for entering floating point values between two limits The knob may be manipulated with the mouse Exact values can be entered by keyboard in a numerical field

Abb. 2.14: KNOB

CREATION

```
PROC create.drehknopf    (VAL INT    x, y,
                          VAL INT32  parent.handle,
                          INT32      control.handle,
                          VAL REAL32 minimum,maximum,
                          CHAN OF SP fs,ts,
                          VAL [DLL.DATA.SIZE]BYTE dll.data,
                          BYTE isv.result)
```

```
VAL INT    x, y
```

Only the position can be defined

```
VAL REAL32 minimum, maximum
```

The limits between which the value can change

DATA HANDLING
The knob has to be set with **send.data** or better **set.real32.status.** For receiving the entered value **poll.control** or **poll.real32** may be used

IMPLEMENTATION
Class:CDreh (dreh h, dreh cpp) derived from CWnd
Fixed size: b= 0, h= 7 including numerical field
Default status: Knob in middle position

2.5.14 Graphic display

DESCRIPTION

The graphic display is able of displaying simple black and white graphics With some special commands it is possible to draw pixels, lines, circles and rectangles in black colour on white background

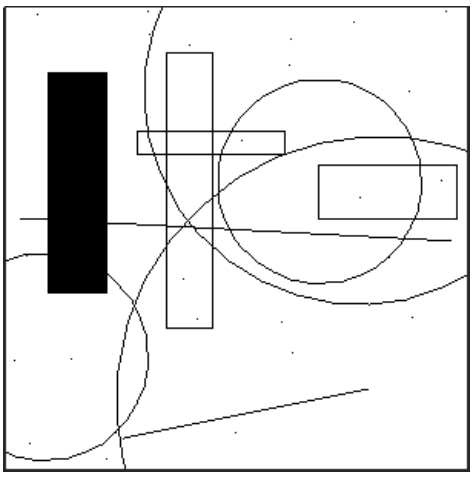


Abb. 2.15: PIXWIN

CREATION

```
PROC create.pixwin  (VAL INT    x, y, b, h,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)
```

Height and width of the object define the number of pixel rows and columns in the window The origin of the coordinate system is the upper left corner of the parent window Some procedures have been implemented for drawing:

```
PROC pixwin.clear(VAL INT32  control.handle,
                  CHAN OF SP fs, ts,
                  VAL [DLL.DATA.SIZE]BYTE dll.data,
                  BYTE isv.result)
```

A call to this procedure clears the graphic window

```
PROC pixwin.plot(VAL INT32  control.handle,
                 VAL INT    x1, y1,
                 CHAN OF SP fs, ts,
                 VAL [DLL.DATA.SIZE]BYTE dll.data,
                 BYTE isv.result)
```

A single pixel is set at position **x1**,**y1.**

```
PROC pixwin.line(VAL INT32  control.handle,
                 VAL INT    x1, y1, x2, y2,
                 CHAN OF SP fs, ts,
                 VAL [DLL.DATA.SIZE]BYTE dll.data,
                 BYTE isv.result)
```

A line is drawn from **x1**, **y1** to **x2**, **y2.**

```
PROC pixwin.rect(VAL INT32  control.handle,
                 VAL INT    x1, y1, x2, y2,
                 CHAN OF SP fs, ts,
                 VAL [DLL.DATA.SIZE]BYTE dll.data,
                 BYTE isv.result)
```

A rectangle with the corner coordinates **x1**, **y1** and **x2**, **y2** is painted

```
PROC pixwin.rect.fill(VAL INT32  control.handle,
                      VAL INT    x1, y1, x2, y2,
                      CHAN OF SP fs, ts,
                      VAL [DLL.DATA.SIZE]BYTE dll.data,
                      BYTE isv.result)
```

As above, but now the rectangle is filled

```
PROC pixwin.circle(VAL INT32  control.handle,
                   VAL INT    x1, y1, r,
                   CHAN OF SP fs, ts,
                   VAL [DLL.DATA.SIZE]BYTE dll.data,
                   BYTE isv.result)
```

A circle with radius **r** is drawn around the point **x1**, **y1.**

DATA HANDLING

Data may be sent to the graphic window using **send.data** after the correct preparing of the parameter array However it is more convenient to use the specialised procedures **pixwin.clear**, **pixwin.plot**, **pixwin.line**, **pixwin.rect.fill**, **pixwin.rect**, and **pixwin.circle**

The display is updated only after a call to **update.ctrl**

For drawing multiple graphic elements in one command, **send.data** has to be used as demonstrated in the following example:

```
[14]BYTE parameter:        *only the size really needed
  INT p:
  SEQ
    p:=0
    push.int16(parameter,p, 3(INT16)) *pixel count
    push.int16(parameter,p,INT16 x1)  *coordinates 1
    push.int16(parameter,p,INT16 y1)
    push.int16(parameter,p,INT16 x2)  *coordinates 2
    push.int16(parameter,p,INT16 y2)
    push.int16(parameter,p,INT16 x3)  *coordinates 3
    push.int16(parameter,p,INT16 y3)
    send.data(control.handle,parameter,fs,ts,
                             dll.data,isv.result)
```

If the first parameter is not a positive value, one of the special functions for drawing a graphic figure is invoked

IMPLEMENTATION

Class:CPixwin (pixwin h, pixwin cpp) derived from CWnd

Defaults: None

2.5.15 Switchable Bitmap

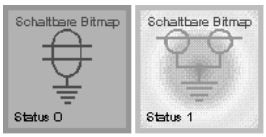


Abb. 2.16: ZWEIBIT

DESCRIPTION

This object is very useful for many applications It is a user input with two states that can be changed by clicking on it The special thing about it is, that the two states are represented by two bitmaps that can be loaded from disc Moreover it is possible to prohibit user input, so that only the Transputer is able to control the state

CREATION

```
PROC create.zweibit( VAL INT    x, y, b, h,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     VAL BOOL   schaltbar,
                     VAL []BYTE fname0, fname1,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)
```

`VAL BOOL schaltbar`

If this parameter is set to TRUE, The status may be changed by the user with a mouse click For indication of this possibility the Bitmap will get a small frame If this parameter is set to FALSE, only the Transputer is allowed to manipulate the object and no frame is drawn

`VAL []BYTE fname0, fname1`

Here the filenames of the bitmaps have to be entered The following formats with extension bmp are recognised:

- Black and white
- 16 colours (greyscale also)
- 6 colours (greyscale also)
- True Colour (16 Mio)

Both compressed and uncompressed RGB Format is supported The Bitmap can have any size for it is adjusted to the surrounding frame The number of colours supported depends on the graphic card of the PC

DATA HANDLING

Switchable Bitmaps are elements to control a status This may be set **set.status** and received using **poll.status.** The state of the Bitmap is stored in one (0=Status 0, 1=Status 1),so that the routines **set.byte.status** and **poll.byte** are appropriate

IMPLEMENTATION
Class:CZweibit (zweibit h, zweibit cpp) derived from CWnd
Defaults: None
Default status: Status 0, Bitmap fname0 is displayed

2.5.16 Oscilloscope

DESCRIPTION
The oscilloscope is able of displaying whole sets of data This data may be divided into two channels The number of values displayed on a channel can be chosen during construction The user can adjust vertical gain and the screen layout during runtime The scope supports a grid and a scale for orientation There is space for a label

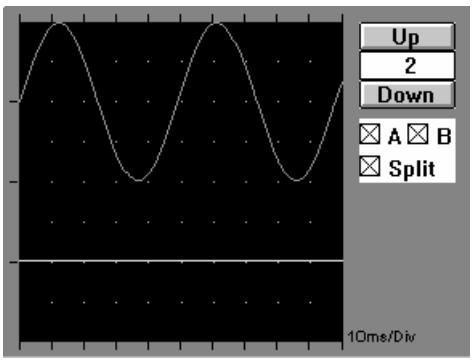


Abb. 2.17: SCOPE

CREATION

```
PROC create.oszi    ( VAL INT    x, y, b, h,
                      VAL INT32  parent.handle,
                      INT32      control.handle,
                      VAL INT    pixel,
                      VAL []BYTE beschriftung,
                      CHAN OF SP fs, ts,
                      VAL [DLL.DATA.SIZE]BYTE dll.data,
                      BYTE isv.result)
```

`VAL INT    pixel`

This parameter describes the number of pixels displayed for each channel If that number of values is transferred to the oscilloscope, drawing starts again at the left border of the display

`VAL []BYTE beschriftung`

The label at the lower right side of the display

DATA HANDLING
The values transferred are stored in 4 Byte floating point format The transfer is done using **send.data.** Refresh may be done with the command **update.ctrl** The setting of vertical gain, splitting and displaying of the channels is supported by **set.status.**
For the oscilloscope no special functions were created, because of their variety

```
create.oszi(5,5,160,150,fenster2.hdl,oszi1.hdl,
    200,"10ms/Div",fs,ts,dll.data,isv.result)
    p:=0
    push.byte(parameter,p,BYTE 7)  --Range 2.0
    push.byte(parameter,p,BYTE 1   --Channel A on
    push.byte(parameter,p,BYTE 1)  --Channel B on
    push.byte(parameter,p,BYTE 1)  --Split on
    set.status(oszi1.hdl,parameter,fs,ts,dll.data,isv.result)
```

Set.status needs 4 Byte for setting the status The first byte describes the maximum amplitude Where the values are listed below:

Byte value 0:	Amplitude	0,01
Byte value 1:	Amplitude	0,0
Byte value :	Amplitude	0,0
Byte value 3:	Amplitude	0,1
Byte value 4:	Amplitude	0,
Byte value :	Amplitude	0,
Byte value 6:	Amplitude	1,0
Byte value 7:	Amplitude	,0
Byte value 8:	Amplitude	,0
Byte value 9:	Amplitude	10,0
Byte value10:	Amplitude	0,0
Byte value11:	Amplitude	0,0
Byte value1 :	Amplitude	100,0
Byte value13:	Amplitude	00,0
Byte value14:	Amplitude	00,0
Byte value1 :	Amplitude	1000,0
Byte value 16:	Amplitude	000,0
Byte value 17:	Amplitude	000,0

Here the amplitude is ment to be the range of one half of the screen (4 grid units)
The next two bytes switch the displaying of the two channels A and B
The last Byte selects splitting of the channels

```
p:=0
push.int16(parameter,p,INT16 100) --count of pixels
push.int16(parameter,p,INT16  0)  --channel A
SEQ i=0 FOR 100
  SEQ
    push.real32(parameter,p,SIN(..for green sinuskurve..))
    ...
send.data(oszi1.hdl,parameter,fs,ts,dll.data,isv.result)
```

The first value in an array of data indicates the number of valid data values following in this packet The next byte integer describes the channel for which the data is sent (A=0, B=1) The rest of the packet is used for data values in real format The parameter field is restricted to 03 byte length, so it is possible to transfer 1 3 values in one packet If the scope should display

more data, subsequent calls to **send.data** have to be done before refreshing the display with **update.ctrl**

IMPLEMENTATION

Class:COszi (oszi h, oszi cpp) derived from CWnd
Defaults: b= 00, h=1 0
Minimal size: b=1 0, h=70
Horizontal count: min , max 00 values
Defaults: Channel A on, Amplitude 0 0

2.5.17 Digital multimeter

DESCRIPTION

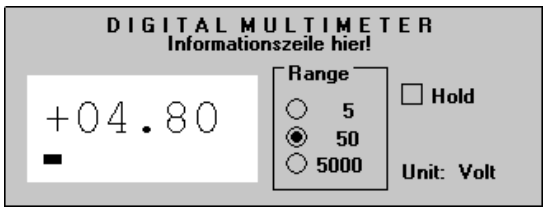


Abb. 2.18: MULTIMETER

The digital multimeter is the first of two specialised objects that have been created for a demonstration They are supplied here as an example and maybe they are somehow useful
The multimeter is able of displaying floating point values These may be 'freezed' by a hold button Three ranges are supported
This object was created using the 'Application Studio' which is the resource editor of MS Visual C++ As a result it is not resizeable

CREATION

```
PROC create.multimeter(VAL INT x, y,
                       VAL INT32  parent.handle,
                       INT32 control.handle,
                       VAL []BYTE einheit,
                       VAL []BYTE infozeile,
                       CHAN OF SP fs,ts,
                       VAL [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result)
```

`VAL INT x, y`

Width and height of the object are determined by the resource and are thus not selectable

`VAL []BYTE einheit`

This string appears after the Word 'Unit: ' Four characters may be used for showing the unit

`VAL []BYTE infozeile`

This string is displayed as the second line of the header It should take some information about the data displayed Maximum string length is 40 characters

DATA HANDLING

A floating point value may be sent to the multimeter for displaying by **send.real32.** Display refresh is initialised with **update.ctrl**
Range and hold button can be set by the Transputer with **set.status** according to the following scheme:

```
p:=0
push.byte(parameter,p,BYTE 1) -- 0->Range 5, 1->50, 2->5000
push.byte(parameter,p,BYTE 0) -- Hold 0-> off, 1-> on
set.status(multi1.hdl,parameter,fs,ts,dll.data,isv.result)
```

IMPLEMENTATION

Class:CMulti (multi h, multi cpp) derived from CDialog
Fixed size: b=181, h=69 from Application Studio
Default status: Range 0, Hold off
Default display: - - - -

2.5.18 Filter control

DESCRIPTION

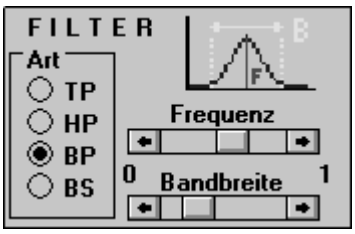


Abb. 2.19: FILTER

Filter control is also a resource based module It is quite special, but may be useful as an example It is used for parametrizing a FIR filter By selecting type, frequency and bandwidth 3 floating point values are generated that control the filter algorithm

CREATION

There is no special create procedure, so the base procedure has to be used

```
create.control(FILTER,x,y,b,h,parent.hdl,"",control.hdl,
                      fs,ts,dll.data,isv.result)
```

Since there are no further parameters necessary for display, an empty parameter string may be supplied

DATA HANDLING

This object can only be manipulated by the user The Transputer may only poll the control with **poll.control.** Four floating point values are supplied that control a special filter algorithm

IMPLEMENTATION

Class:CFilter (filter h, filter cpp) derived from CDialog
Fixed size: b=98, h=67
Default status: Lowpass, frequency = 0 (relative to ½ f_A)

3 Extending the library

3.1 Adding object modules

It is easy to add an object to the user interface library In the DLL a class must be added, that contains the required methods In the Transputer library in 'wintrans occ' at least a special create procedure for the new object should be written The DLL was compiled using MS Visual C++ 1 The Transputer library was created with Inmos Occam Toolset D730

The following steps have to be taken:

1 After the visual appearance of the object is defined, the format of the parameters should be described It has to be decided which methods are needed for getting and setting the status or displaying data
The two files „leer h" and „leer cpp" contain the frame of a module class They have to be copied and renamed into a descriptive name for the module The class name in the files is Cxxx This expression has to be globally replaced with a special one for the new module

3 Using the **Class Wizard,** these two files have to be imported into the project, so that they can be administered by the development system

4 Now the constructor has to be coded After construction the object should be visible on screen The parameters up to byte #13 are predefined and must not be changed Starting with **(pointer->data)+14** the parameter array can deliberately be used by the programmer
According to the function of the object the **SendData**, **SetStatus**, **PollControl** and **UpdateControl** also have to be implemented Some bytes of the parameter array are fixed even in these routines See the source code for this information

6 For binding the object module with the DLL main procedure some modifications are needed for 'link cpp':

- Importing the header „xxx h" with `#include`
- ```
  switch (tag)     // Art des Objekts?
          { case 18:  ptr = new CTextbox(pMainFrame,pointer); break;
  ```

  Copying and renaming one line of the **case** statement The selection value chosen for that entry is the new object number Instead of **CTextbox** the new class name has to be inserted In this way the new constructor can be called
  ```

- ```
  switch(art)
     {
     case 18:
       switch (befehl)
         {
         case 2: ((CTextbox*)ptr)-> SetStatus(pointer); break;
         case 3: ((CTextbox*)ptr)->PollStatus(pointer); break;
         case 4: ((CTextbox*)ptr)->  SendData(pointer); break;
         case 5: ((CTextbox*)ptr)->UpdateCtrl();        break;
         }
       break;
  ```

  Using the same object number, this **case** block has to be copied or added The name of the class has to be renamed again Thus it is possible to call the four standard procedures for the object

After this step it should be possible to create the object on screen using **create.control** and a correctly prepared parameter array

7 The Transputer library should be extended with at least one special create procedure that prepares the parameter string

```
PROC create.objektname(VAL INT    x, y, b, h,
                       VAL INT32  parent.handle,
                       INT32      control.handle,
                       VAL BOOL   parameter1,
                       VAL REAL32 parameter2,
                       VAL []BYTE parameter3,
                       CHAN OF SP fs, ts,
                       VAL [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result)
  [503]BYTE parameter:
  INT p:
  SEQ
    p:=0
    push.byte  (parameter,p,BYTE parameter1)
    push.real32(parameter,p,     parameter2)
    push.string(parameter,p,     parameter3)
    create.control(OBJ.NAME, x, y, b, h, parent.handle,
                   [parameter FROM 0 FOR p],control.handle,
                   fs,ts,dll.data,isv.result)
:
```

This is just a rough example of a create procedure In **OBJ.NAME** the object number defined in the DLL has to be inserted It should be included in „wintrans inc“ The easiest way of creating a new module is copying an old one and modifying it

3.2 Improvements

Here are some suggestions for improving WinTrans or the WIServer:

- As mentioned above, the WIServer does not remove wintrans dll from memory when it is closed As a result subsequent calls to WIServer cause
```

Windows to crash For avoiding this error WIServer has to be modified and recompiled

- If a recompilation of WIServer is done, wintrans dll might be linked to it conventionally The main switch statement of WinTrans has to be included into the main server loop for that purpose This would avoid a 6 byte overhead in each call to WinTrans
- It would be useful if VBX controls are supported by WinTrans This is not possible in DLLs, because Visual C++ does not support the use of VBX controls in DLLs The only possibility is a static linking as explained above
- The graphic display window may be extended with colours and more complex drawing routines The output of text into this window should be considered

4 Demonstration

The package 'wintrans zip' contains this documentation, the WinTrans libraries, complete source code and two demonstration programs
The example **demo** shows five screens with samples of all object modules
This application is terminated by the Transputer and can not be terminated by closing the window
The **demo2** example only shows one screen It may be terminated by closing the window or pressing an exit button

4.1 Preparations

The bootable files **demo.bah** and **demo2.bah** have to be in the same directory as the DLL file **link.dll** WIServer has to be installed properly
WIServer is called with an initialisation file as a parameter This file **demo.ini** is also provided with **wintrans.zip** It might have to be edited for the correct path of the driver DLLs of WIServer

```
Link_name ....\WISERVER\B008DRV.DLL X
```

4.2 demo.bah

The source code of this demonstration is provided with WinTrans and may be freely used as a basis for own applications
After declaring the object handles and some global variables, the DLL is called with:

```
DLL.laden ("link.dll",fs,ts,dll.data,isv.result)
allow.no.exit(fs,ts,dll.data,isv.result)
```

The procedure **allow.no.exit** prevents the user from closing the main window of the DLL After confirming the message box with the Enter key, the first screen appears

SCREEN 1 - WINDOWS

A selection of windows in different colours, frames and titlebars is displayed The blue and the white ones are independent from each other The smaller ones are child windows of the blue one By moving and resizing some of the windows the relations between parent and child windows can be explored It shows, that the yellow and the green one do not change their dimensions These are of type 0 and Every text line inside the windows is a textoutput It is resized with the surrounding window
This step is left using the next button

SCREEN 2 - OBJECTS

During this stage a green window contains different objects These are: Switch, Lamp, Pushbutton, Switchable Bitmap, Checkbox, Radiobuttons and Textinput using these modules the interaction of user input and output is demonstrated Each input is registered by the Transputer and causes another object to change If the switch is operated, the lamp is turned on and off The Pushbutton changes the Textoutput below The strings entered into the Textinput are displayed at the bottom left side of the window
The upper of the two Bitmaps is switchable Its status is sent to the black Checkbox and vice versa The lower Bitmap is not switchable, but may be changed using the yellow Checkbox

SCREEN 3 - GRAPHIC & RESOURCES

A Digital Multimeter and a Filter control are displayed on screen as examples of resource based objects They do not change their size when the window is resized The multimeter as well as a Graphic window display random data

SCREEN 4 - INSTRUMENTATION

This window shows the elements that process floating point data The digital display shows a random value (0 0) that is used as a multiplier for all displayed data The other display shows the product of this value and the value provided by the upper knob for example

All input objects have different scales, ranges or divisions The four types of Bargraphs are demonstrated

SCREEN 5 - PROCESS CONTROL

The left picture is a demonstration of the usage of switchable Bitmaps for interactive process visualisation A big Bitmap is used as a background The valve, the vessel and the switch are further Bitmaps By clicking on them the picture can be animated

At the right side a selection box is used for text input that is displayed at the bottom left side The selection list is extended by one entry each time it is selected

The difference between the two oscilloscopes is the amount of data sent The sine wave is sent with full 200 pixels each time before it is refreshed The second scope only gets one floating point value each refresh interval

The program is terminated with the End button

4.3 demo2.bah

This sample demonstrates, how an application is created, that allows direct termination by the user Only the second screen of 'demo bah' is displayed The window may be close using Alt-F4 or the command button of the window For correct program termination, the existence of the main window has to be steadily polled with **poll.exit** If this results in TRUE, the Transputer has to remove the DLL and terminate the Transputer application A call to **close.all** is not necessary, because the objects were deleted by the DLL when e g Alt-F4 was entered

/ / Pountain, Dick und May, David: A tutorial Introduction to occam Programming; INMOS / BSP Professional Books, Oxford (UK) 988

/2/ Morgan, Mike: WIServer user manual; Publication over Internet, Bristol (UK) 99

/ / INMOS: occam 2 Toolset, Language and Libraries Reference Manual, 72 TDS 68 0 ; INMOS, Bristol (UK) 994

/4/ Kruglinski, David J : Inside Visual C++ (dt Übersetzung); Microsoft Press Deutschland, Unterschleißheim 994

/5/ Microsoft Corporation: Windows™-Guide to Programming; Microsoft Press, Washington (USA) 990

/6/ Klein, Mike: Windows Programmer's Guide To DLLs And Memory Management; Sams Publishing, New Baskerville (USA) 992

TOOLS

```
PROC push.real32([]BYTE buffer, INT pos,VAL REAL32 value)
PROC pop.real32 ([]BYTE buffer, INT pos,REAL32 value)
```

BASE PROCEDURES

```
PROC create.control( VAL BYTE   art,
                     VAL INT    x.pos, y.pos,
                     VAL INT    breite, hoehe,
                     VAL INT32  parent,
                     VAL []BYTE parameter,
                     INT32      ctrl.hdl,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC set.status    ( VAL INT32  ctrl.hdl,
                     VAL []BYTE parameter,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC send.data     (  VAL INT32  ctrl.hdl,
                     VAL []BYTE parameter,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC update.ctrl   (  VAL INT32  ctrl.hdl,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC poll.control  (  VAL INT32 ctrl.hdl,
                     BOOL changed,
                     []BYTE status,
                     CHAN OF SP fs,ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC destroy.control(VAL INT32  ctrl.hdl,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result )
```

ADDITIONAL PROCEDURES

```
PROC DLL.laden (     VAL []BYTE filename,
                     CHAN OF SP fs, ts,
                     [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)
```

```
PROC DLL.entfernen ( CHAN OF SP fs, ts,
                     [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC msg.box    (    VAL []BYTE string1,
                     VAL []BYTE string2,
                     VAL []BYTE string3,
                     CHAN OF SP fs, ts,
                     [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC close.all  (    CHAN OF SP fs, ts,
                     [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC allow.no.exit ( CHAN OF SP fs, ts,
                     [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC poll.exit    (  BOOL buttonvalue,
                     CHAN OF SP fs, ts,
                     [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC changed.something(BOOL       som.ch,
                       CHAN OF SP fs, ts,
                       [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result)
```

CREATE PROCEDURES

```
PROC create.schalter(VAL INT    x, y, b, h,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC create.lampe  ( VAL INT    x, y, b, h,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC create.multimeter(VAL INT    x, y,
                       VAL INT32  parent.handle,
                       INT32      control.handle,
                       VAL []BYTE einheit,
                       VAL []BYTE infozeile,
                       CHAN OF SP fs,ts,
                       VAL [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result)
```

```
PROC create.oszi    ( VAL INT    x, y, b, h,
                      VAL INT32  parent.handle,
                      INT32      control.handle,
                      VAL INT    pixel,
                      VAL []BYTE beschriftung,
                      CHAN OF SP fs, ts,
                      VAL [DLL.DATA.SIZE]BYTE dll.data,
                      BYTE isv.result)

PROC create.fenster( VAL INT      x, y, b, h,
                     VAL INT32    parent.handle,
                     INT32        control.handle,
                     VAL INT16    art,
                     VAL BYTE     rot, gruen, blau,
                     VAL []BYTE   fenstertext,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC create.pushbutton(VAL INT    x, y, b, h,
                       VAL INT32  parent.handle,
                       INT32      control.handle,
                       VAL []BYTE buttontext,
                       CHAN OF SP fs, ts,
                       VAL [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result)

PROC create.checkbox(VAL INT    x, y, b, h,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     VAL BYTE   textfarbe,
                     VAL []BYTE buttontext,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC create.2.radiobutton(VAL INT    x, y, b, h,
                          VAL INT32  parent.handle,
                          INT32      control.handle,
                          VAL BYTE   text.color,
                          VAL []BYTE header,
                          VAL []BYTE string1,
                          VAL []BYTE string2,
                          CHAN OF SP fs, ts,
                          VAL [DLL.DATA.SIZE]BYTE dll.data,
                          BYTE isv.result)
```

```
PROC create.3.radiobutton(VAL INT    x, y, b, h,
                          VAL INT32  parent.handle,
                          INT32      control.handle,
                          VAL BYTE   text.color,
                          VAL []BYTE header,
                          VAL []BYTE string1,
                          VAL []BYTE string2,
                          VAL []BYTE string3,
                          CHAN OF SP fs, ts,
                          VAL [DLL.DATA.SIZE]BYTE dll.data,
                          BYTE isv.result)

PROC create.4.radiobutton(VAL INT    x, y, b, h,
                          VAL INT32  parent.handle,
                          INT32      control.handle,
                          VAL BYTE   text.color,
                          VAL []BYTE header,
                          VAL []BYTE string1,
                          VAL []BYTE string2,
                          VAL []BYTE string3,
                          VAL []BYTE string4,
                          CHAN OF SP fs, ts,
                          VAL [DLL.DATA.SIZE]BYTE dll.data,
                          BYTE isv.result)

PROC create.5.radiobutton(VAL INT    x, y, b, h,
                          VAL INT32  parent.handle,
                          INT32      control.handle,
                          VAL BYTE   text.color,
                          VAL []BYTE header,
                          VAL []BYTE string1,
                          VAL []BYTE string2,
                          VAL []BYTE string3,
                          VAL []BYTE string4,
                          VAL []BYTE string5,
                          CHAN OF SP fs, ts,
                          VAL [DLL.DATA.SIZE]BYTE dll.data,
                          BYTE isv.result)

PROC create.6.radiobutton(VAL INT    x, y, b, h,
                          VAL INT32  parent.handle,
                          INT32      control.handle,
                          VAL BYTE   text.color,
                          VAL []BYTE header,
                          VAL []BYTE string1,
                          VAL []BYTE string2,
                          VAL []BYTE string3,
                          VAL []BYTE string4,
                          VAL []BYTE string5,
                          VAL []BYTE string6,
                          CHAN OF SP fs, ts,
                          VAL [DLL.DATA.SIZE]BYTE dll.data,
                          BYTE isv.result)
```

```
PROC create.scrollbar.h ( VAL INT    x, y, b, h,
                          VAL INT32  parent.handle,
                          INT32      control.handle,
                          VAL INT    dx.zahl, dy.zahl,
                          VAL REAL32 minimum,maximum,
                          VAL INT    teilung,
                          CHAN OF SP fs, ts,
                          VAL [DLL.DATA.SIZE]BYTE dll.data,
                          BYTE isv.result)

PROC create.scrollbar.v ( VAL INT    x, y, b, h,
                          VAL INT32  parent.handle,
                          INT32      control.handle,
                          VAL INT    dx.zahl, dy.zahl,
                          VAL REAL32 minimum,maximum,
                          VAL INT    teilung,
                          CHAN OF SP fs, ts,
                          VAL [DLL.DATA.SIZE]BYTE dll.data,
                          BYTE isv.result)

PROC create.textin ( VAL INT    x, y, b,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC create.textout( VAL INT    x, y, b, h,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     VAL BYTE   textfarbe,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC create.bar   (  VAL INT    x, y, b, h,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     VAL BYTE   art,
                     VAL REAL32 referenzwert,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC create.bar.scale(VAL INT    x, y, b, h,
                      VAL INT32  parent.handle,
                      INT32      control.handle,
                      VAL REAL32 minimum, maximum,
                      VAL []BYTE textmin, textmax, einheit,
                      CHAN OF SP fs, ts,
                      VAL [DLL.DATA.SIZE]BYTE dll.data,
                      BYTE isv.result)
```

```
PROC create.digital( VAL INT    x, y, b, h,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC create.drehknopf( VAL INT    x, y,
                       VAL INT32  parent.handle,
                       INT32      control.handle,
                       VAL REAL32 minimum, maximum,
                       CHAN OF SP fs, ts,
                       VAL [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result)

PROC create.zweibit( VAL INT    x, y, b, h,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     VAL BOOL   schaltbar,
                     VAL []BYTE fname0, fname1,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC create.pixwin ( VAL INT    x, y, b, h,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC create.textbox( VAL INT    x, y, b,
                     VAL INT32  parent.handle,
                     INT32      control.handle,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)
```

SEND PROCEDURES

```
PROC send.string  (  VAL INT32  control.handle,
                     VAL []BYTE string,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC send.real32  (  VAL INT32  control.handle,
                     VAL REAL32 real,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)
```

```
PROC send.byte  (    VAL INT32  control.handle,
                     VAL BYTE   byte,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)
```

SET PROCEDURES

```
PROC set.byte.status(VAL INT32  control.handle,
                     VAL BYTE   byte,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC set.string.status(VAL INT32  control.handle,
                       VAL []BYTE string,
                       CHAN OF SP fs, ts,
                       VAL [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result)

PROC set.real32.status(VAL INT32  control.handle,
                       VAL REAL32 real,
                       CHAN OF SP fs, ts,
                       VAL [DLL.DATA.SIZE]BYTE dll.data,
                       BYTE isv.result)
```

POLL PROCEDURES

```
PROC poll.byte   (   VAL INT32  control.handle,
                     BOOL       changed,
                     BYTE       byte,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC poll.string  (  VAL INT32  control.handle,
                     BOOL       changed,
                     []BYTE     string,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC poll.real32  (  VAL INT32  control.handle,
                     BOOL       changed,
                     REAL32     real,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC poll.change.bool(VAL INT32  control.handle,
                      BOOL       changed,
                      CHAN OF SP fs, ts,
                      VAL [DLL.DATA.SIZE]BYTE dll.data,
                      BYTE isv.result)
```

PIXWIN PROCEDURES

```
PROC pixwin.clear  ( VAL INT32 control.handle,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC pixwin.plot   ( VAL INT32  control.handle,
                     VAL INT    x1, y1,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC pixwin.line   ( VAL INT32  control.handle,
                     VAL INT    x1, y1, x2, y2,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC pixwin.rect   ( VAL INT32  control.handle,
                     VAL INT    x1, y1, x2, y2,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)

PROC pixwin.rect.fill(VAL INT32  control.handle,
                      VAL INT    x1, y1, x2, y2,
                      CHAN OF SP fs, ts,
                      VAL [DLL.DATA.SIZE]BYTE dll.data,
                      BYTE isv.result)

PROC pixwin.circle ( VAL INT32  control.handle,
                     VAL INT    x1, y1, r,
                     CHAN OF SP fs, ts,
                     VAL [DLL.DATA.SIZE]BYTE dll.data,
                     BYTE isv.result)
```

CONTENTS OF THE PARAMETER ARRAY FOR CREATE

Th gr y f l s ar f ll by th bas roc ur **create.control.** Th r st of th array has to b s t us g ush comma s or s c al s cr at roc - ur s.

Str g aram t rs g t a r c g two byt s l gth. Th s s automat cally o by **push.string.** Th l gth of a str g s ot l m t , so th os t o of subs qu t aram t rs s ot f .

Objektart	CREATE TAG	art																													
FENSTER	1	6	x	y	b	h	parent.hdl	art		color r	g	b	-	fenstertext..... strlen	text																
LAMPE	1	2	x	y	b	h	parent.hdl																								
SCHALTER	1	1	x	y	b	h	parent.hdl																								
BUTTON	1	7	x	y	b	h	parent.hdl	buttontext ..... strlen	text																						
CHECKBOX	1	8	x	y	b	h	parent.hdl	bool	buttontext ..... strlen	text																					
RADIOBUTTON	1	9	x	y	b	h	parent.hdl	anz.	bool	string1.... strlen	text							string2.... strlen	text			je nach Anzahl und Textlängen									
SCROLLBAR	1	10	x	y	b	h	parent.hdl	richt.	dx		dy		minimum				maximum				teilung									
TEXTIN	1	11	x	y	b	--	parent.hdl																								
TEXTOUT	1	12	x	y	b	h	parent.hdl	bool																							
TEXTBOX	1	18	x	y	b	--	parent.hdl																								
BAR (Typ 0-2)	1	13	x	y	b	h	parent.hdl	art	referenzw.																						
BAR (Typ 3)	1	13	x	y	b	h	parent.hdl	art	maximum				minimum				textmax strlen	text			textmin strlen	text								
	Position der Strings im Array jeweils abhängig von ihrer Länge.																einheit strlen	text												
DIGITAL	1	14	x	y	b	h	parent.hdl																								
DREHKNOPF	1	15	x	y	--	--	parent.hdl	minimum					maximum																		
Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

CONTENTS OF THE PARAMETER ARRAY FOR CREATE

Object type	CREATE TAG	art																														
DIGITAL	1	14	x	y	b	h	parent.hdl																									
KNOB	1	15	x	y	b	h	parent.hdl	minimum				maximum																				
PIXEL WINDOW	1	17	x	y	b	h	parent.hdl																									
BITMAP	1	16	x	y	b	h	parent.hdl	switch	fname0.....			strlen	text											fname1.....			strlen	text				
SCOPE	1	4	x	y	b	h	parent.hdl	pixel		labelling....			strlen	text																	
MULTIMETER	1	3	x	y	--	--	parent.hdl	units.....			strlen	text				infoline.....			strlen	text											
FILTER	1	5	x	y	--	--	parent.hdl																									
Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

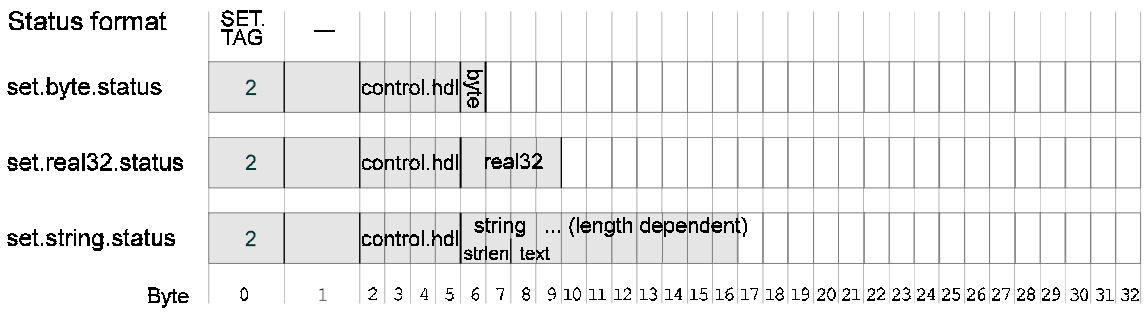
RESULTS OF CREATE

Th obj ct ha l s wr tt to th array by th DLL roc ur mm at ly aft r cr at o . Th s ha l has to b us for t f cat o of th obj ct lat r calls.

Object handle from CREATE	control.handle			
	m_hWnd		art	0
Byte	0	1	2	3

CONTENTS OF THE PARAMETER ARRAY FOR SET

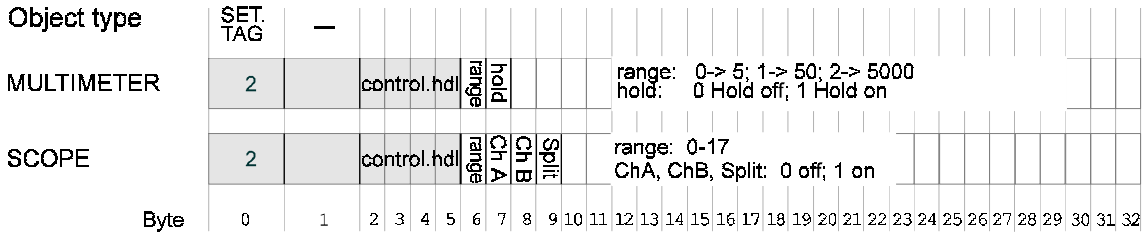
Most of th obj cts ca g t th r status w th o of thr s c al s t roc - ur s.



Byt : 0=Off, status 0, ot s l ct - 1=O , status 1, s l ct
for Sw tch, Ch ckbo , sw tchabl B tma .
0-6 = s l ct Ra obutto (0= o).
R al: Pr s t float g o t stat of Scrollbar or K ob.
Str g: E tabl fault t t for T t ut.

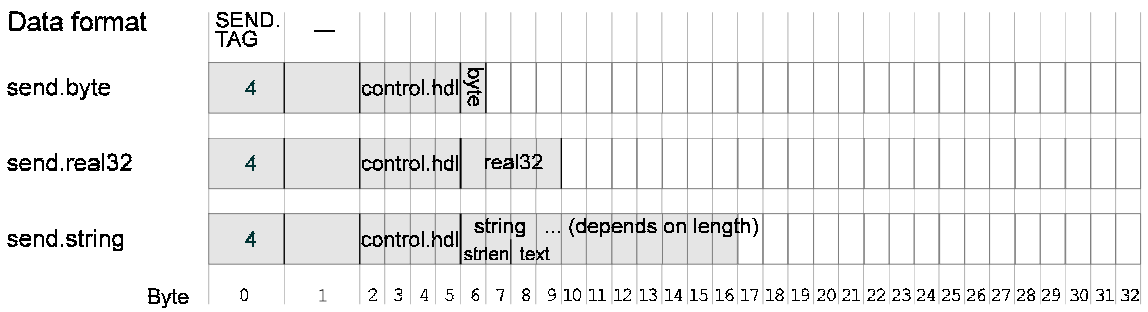
STATUS OF MULTIMETER AND OSCILLOSCOPE

Th r ar o s c al roc ur s for s tt g th status of Mult m t r a Os- c llosco . Th bas roc ur s t.status has to b us for aram tr z g start g w th byt 6.



CONTENTS OF THE PARAMETER ARRAY FOR SEND

Most of th obj cts ca b rov w th s lay ata us g th thr s c al s roc ur s.

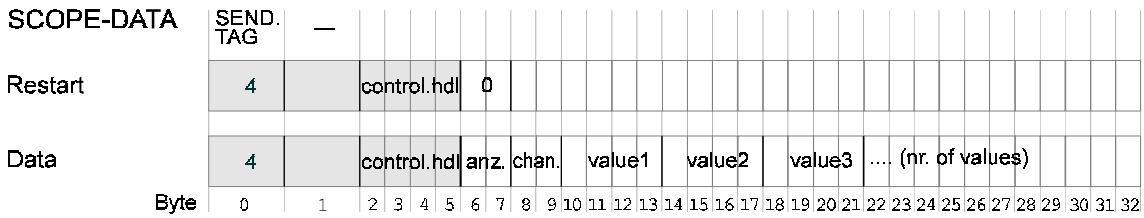


Byt : 0=Off, 1=O for Lam .
R al: Float g o t ata for
Mult m t r, Bargra h, D g tal s lay.

Str g: t o al l m t for S l ct o bo ,
co t t of T tout ut.

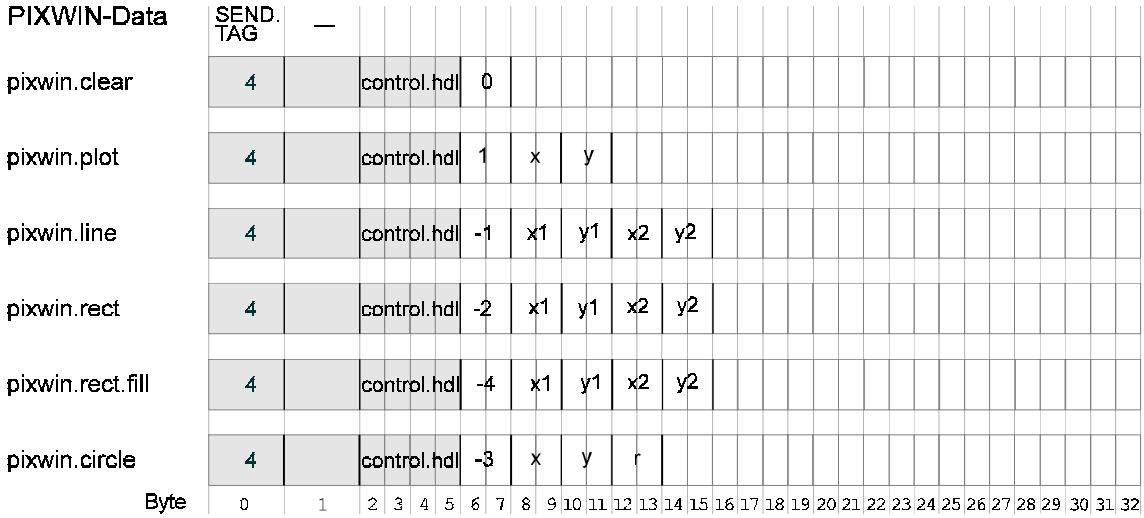
SENDING DATA TO THE OSCILLOSCOPE

Th gr y f l s ar f ll by th roc ur s . ata. Th wh t o s must b rov ma ually. Th ata to b s lay has to b s aft r cou t a cha l s l ct o .

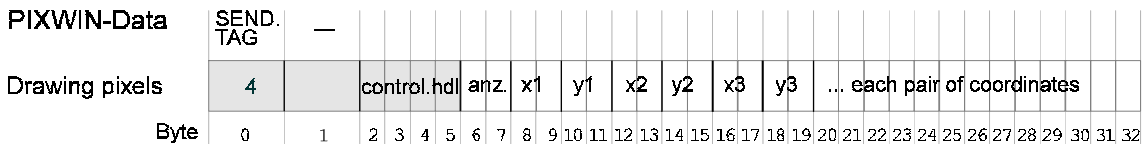


SENDING DATA TO THE GRAPHIC WINDOW

Th aram t r f l s com l t ly a m st r by th cat roc ur s for raw g to th Gra h c w ow.

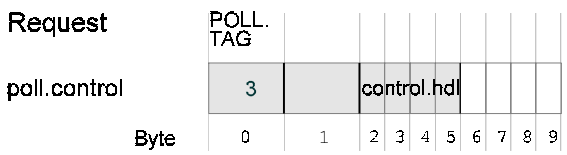


Tra sm tt g of mor tha o l o ack t r qu r s ma ual s tt g of th aram t r f l .

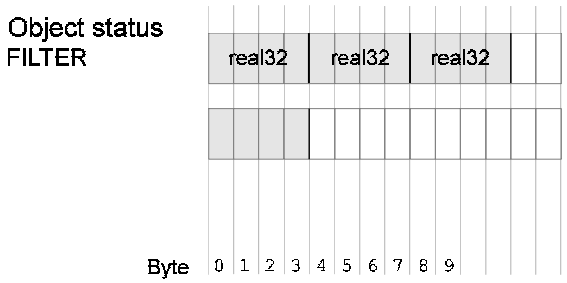


CONTENTS OF THE PARAMETER ARRAY FOR POLL

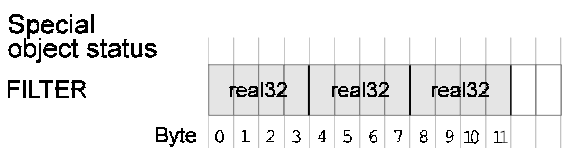
Th r qu st ack t looks l k th s:



Most of th obj cts r tur a aram t r f l l k th follow g. It ca b r a us g s c al s oll fu ct o s.

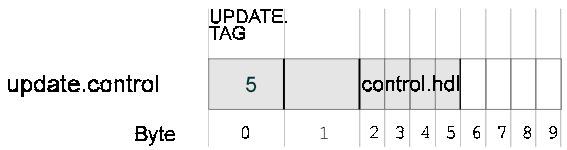


Byt : 0=Off, status 0, ot s l ct - 1=O status 1, s l ct
for Sw tch, Ch ckbo , sw tchabl B tma .
0-6 = s l ct Ra obutto (0= o).
R al: Pr s t float g o t valu for
Scrollbar a K ob.
Str g: S l ct or t r t t
S l ct o bo or T t ut.



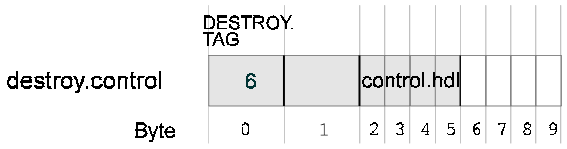
CONTENTS OF THE PARAMETER ARRAY FOR UPDATE

Th s s th array for **update.ctrl**.

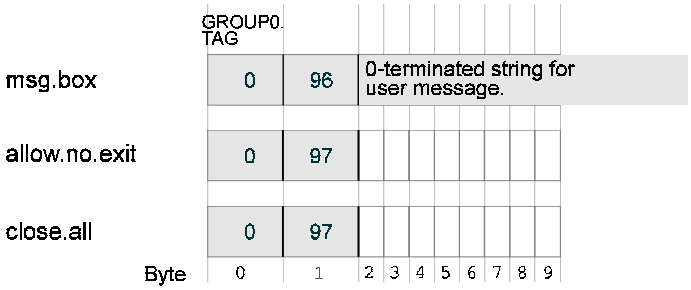


CONTENTS OF THE PARAMETER ARRAY FOR DESTROY

Th follow g array s s t by **destroy.control**.

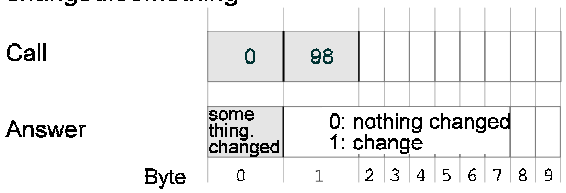
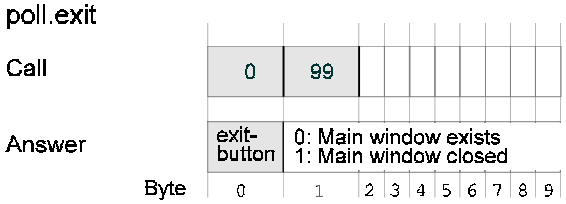


CONTENTS OF THE PARAMETER ARRAY FOR ADDITIONAL FUNCTIONS



ADDITIONAL FUNCTIONS WITH ANSWER

changed.something



poll.exit