A.S.O.S.

COLLABORATORS			
	TITLE : A.S.O.S.		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		April 15, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

A.S.O.S.

1.1 A.S.O.S. v1.35 docs.

```
                Power Programs documentation,                           A.S.O.S  ↩
                   v1.35

Preface:

    As allways, writing a piece of software takes time. This product,
    A.S.O.S. v1.35, has been in development for over FOUR YEARS now,
    we at Power Programs hope that this effort is not lost on you. We
    hope that this product is one of high standards and will aid you
    in your quest for great AMOS programs!
                                                Jeroen Knoester,
                                                Coder Power Programs.

Contents:
```

1.2 Our Adress

```
Power Programs documentation,                           A.S.O.S v1.35

Power Programs,
Madelieventuin 40,
2724PL Zoetermeer,
Holland,
Europe.

Our web-pages are at http://www.infothuis.nl/jeroen.

You can e-mail us with questions at jeroen@infothuis.nl.
Flames are ignored!

Power Programs documentation,                                  Page 2
```

1.3 Power Programs Disclaimer and Copyright

```
Power Programs documentation,                           A.S.O.S v1.35

Please note:

Terms of usage / liscense:

In the following disclaimer "The author" represents the main
programmer on this product, in this case: Jeroen Knoester.

In the following disclaimer "THE SOFTWARE" represents the
software package A.S.O.S., including, but not limited to,
the executable code, the documentation and any datafiles.

    The author makes no warranties, conditions or representations express
    or implied, with respect to THE SOFTWARE, its quality, merchantability
    or fitness for any particular purpose. THE SOFTWARE is provided "AS IS".
    In no event shall the author be liable for any special, indirect or
    conseqeuntial damages.

A.S.O.S. is free software, wich means:

A.S.O.S. can be used, copied, edited, remade, etc. As long as the
original authors, package name and these terms are clearly mentioned
in the documentation of the so-changed software.

It also means that all source and datafiles are available for
free and should have been included in this package.

Remember, A.S.O.S. is (C) 1995-1998 Jeroen Knoester / Power Programs.

Power Programs documentation,                                  Page 3
```

1.4 Introduction to A.S.O.S. v1.35

```
                Power Programs documentation,                            A.S.O.S  ←↩
                   v1.35

A.S.O.S. is an utility for all the AMOS users out there who wish
to be able to add a GUI ( Graphical User Interface ) to their
program, without all the long-winded hassle of programminging
one from scratch. A.S.O.S. is an abreviation, standing for:
Amos mouSe Operated System. I originally named it AMOS, but that
name was in use by someone else allready...

Although A.S.O.S. cannot perform miracles, such as true intuition
support from AMOS, it does provide you with a set of ready to use
procedures for the creation of a GUI and examples on how to use
these routines.

A.S.O.S. is a PROGRAMMING utility and can be thought of as a runtime
library. If you cannot program in AMOS, then A.S.O.S. will not
teach you to. A.S.O.S. is free software.

Next: The
                requirements
                 of A.S.O.S.

Power Programs documentation,                                   Page 4
```

1.5 Requirements to use A.S.O.S. v1.35

```
                Power Programs documentation,                            A.S.O.S  ←↩
                   v1.35

Brief:

Minimum: 1mb of ram, a 68000 processor and Kickstart 1.3.
Recommended: 2mb+ ram, a 68020+ processor, Kickstart 2.04+ and a harddrive.

In both cases, you obviously also need AMOS 1.3+ or AMOS PRO. I also
recommend having acces to the AMOS/PRO compiler.

Detailed:

A.S.O.S. is a complex piece of AMOS code and, as a result, it is quite
large. It’s source currently is 58834 bytes big. I recommend having
a variable space of at least 64kb and an editor space of at least 64kb.

A.S.O.S. is NOT an extension and therefore has a few extra things worth
noting:

  - Since it is AMOS compatible code, it won’t interfere with other
    extensions.

  - A.S.O.S. uses quite a bit of global variables and editor buffer space,
    so a ’power’ Amiga is recommended.
  - AMOS code is not lightning fast assembly, so don’t expect any speed
```

```
    miracles. A.S.O.S. is as fast as any efficient AMOS progam.

  - A.S.O.S. works in any screenmode, but 640x256x8 or 640x256x16 is
    recommended. Default is 640x256x8.
  - A.S.O.S. uses standard AMOS functions for drawing / redrawing.
    Note: A.S.O.S. is not meant for usage on double buffered screens.
    It will probably mess up your screen looks.

  - Last, but certainly not least: A.S.O.S. is a programmers utility.
    It won't write your programs for you. It will, however, hopefully
    make writing your code easier.
```

Next: The
installation
of A.S.O.S.

Power Programs documentation, Page 5

1.6 Installing of A.S.O.S. v1.35

Power Programs documentation, A.S.O.S ↩
v1.35

Installing A.S.O.S. is simple, simply drag the ASOS drawer to
the location where you wish to use it. Note that if you wish to use
A.S.O.S. in your programs, you will need the asos icon file in the local
directory.

For example, you wish to use A.S.O.S. in a program located in
'work:amos/prog1/' the A.S.O.S. icons should then be located:
'work:amos/prog1/asicons.abk'.

For your finished releases, you will also need to include the asos icon
file on the distribution disk, in a directory wich is local to your path.

A.S.O.S. requires about 100KB to install.

Next: A description of the
usage
of A.S.O.S.

Power Programs documentation, Page 6

1.7 Usage of this program

Power Programs documentation, A.S.O.S ↩
v1.35

4.1, supplied executables.

```
The supplied executables are all AMOS compiled ones should be able
to run without any extra work.

4.2, supplied demos/tutorials.

The supplied demos and tutorials are AMOS source code examples,
wich do NOT have the A.S.O.S. procedures includes, as to save
disk space. Including the A.S.O.S. procedures can be done reasonably
simply:

If you have AMOSPRO, simply use the 'include' command to include
them from their path ( example: 'include work:ASOS_Sys/asos-procedures.amos' )

If you have AMOS, go to the top of your program and use the MERGE
Command, from the 'right mousebutton menu'.

You can now test if all works by running the program. If it did not
work, try again from scratch. If you cannot get it to work, by all
means
                send
                 me mail. I will try to answer it to the
best of my ability. Note: e-mail is probably answered quicker than
snail-mail. Also note: snail-mail is only answered if I got the time
to do so.

4.3, Your own programs.

The best way to usage A.S.O.S. from your own program is to include
the A.S.O.S. procedures in your program from the very beginning.
It's easier to remove A.S.O.S. code later than to add it to a
finished product!

For instructions on how to include A.S.O.S. procedures in your program,
see chapter 4.2, supplied demos/tutorials, wich is found on this very
page.

Next: A
                tutorial
                 on A.S.O.S.
Power Programs documentation,                                 Page 7
```

1.8 Tutorial on A.S.O.S. v1.35

```
                Power Programs documentation,                         A.S.O.S ↩
                   v1.35


5.1 Introduction.

   In order to begin making ASOS driven programs, you should first get
   to know ASOS. This tutorial aims to be a helping hand in getting to
   know the ASOS gui 'language'.
```

To help you get started, all programs in this tutorial are also on disk, so you can easily load them into AMOS and experiment with them.

Before we can begin coding ASOS programs however, you need to know how ASOS programs work.

To begin, you need to add the ASOS procedures to your program (see section 4.2 to know how to do that).

After that, you need to 'tell' ASOS where some things are, so it can get started. This is because all ASOS programs use the same set of variables and procedures to get them to work. And like a computer program needs to get started up, so does ASOS.

Starting up ASOS is easy. In order to get any ASOS program 'started' just type the following lines into your ASOS program (before any ASOS calls).

```
ABOUT
```

About displays a small bit of information about ASOS and 'starts' ASOS by initializing important variables, as well as the screen.

If you do not want to have the about window, you can also use ADEFAULT instead. ADEFAULT just initializes the variables and the screen, but displays no welcome message. For know, we assume to use ABOUT.

One last thing: If you wish to know more about a command, check the

Quick Reference guide
for more information!

5.2 Opening a window.

Since ASOS's functions all have to do with windows, the first thing we want to have is an open window, to experiment with. Opening a window in ASOS is really simple. You just use WINREQ to get a window number (stored in the variable ANS) and then make a call to WINOP. See for yourself:

```
ABOUT
WINREQ : OURWINDOW=ANS
WINOP [ 40, 40, 600, 160, 0, OURWINDOW, "Hello world window!" ]
```

That's all. Now for what it all means:

```
ABOUT About is used to 'start' ASOS.
WINREQ : OURWINDOW=ANS Winreq is a procedure which gets the first free window
                       number for you. This is stored in the global variable
     ANS.
WINOP [ 40, 40, 600, 160, 0, OURWINDOW, "Hello world window!" ]
Winop opens a window for you.
```

The first four variables state it's size (Top left X, Top left Y, Bottom right X, Bottom right Y).

The next variable sets the type of window. For now, just use type 0 (default) ↩ .

The next variable is the window number. ASOS allows you to have more than one ↩
open
window, and this number identifies each one.

The last variable is the title of the window.

Opening a window is easy. Now lets add some text to it:

Add the following line to your program:

```
WINTX [ OURWINDOW, "Hello, World!", -1, 2, 0 ]
```

This command puts text into window OURWINDOW. The text is Hello, World!.

The last three variables give a place and a mode to the text.

The first (-1) is the x coordinate. Normally, you'd use something like 1 or ↩
10 here.
-1 Is a special form. If the X coordinate is -1, the text will be centered in ↩
the window.

The second (2) is the y coordinate, use it to set the vertical position of ↩
the text.

The last is the text-mode. Multiple modes (underline, shade and inverse) are ↩
available.
For now, use type 0, normal text.

Now we wish to wait for a key and close the window (and end the first tutorial ↩
!).

To do that, add the following lines of code:

```
WATKEY
WINCL[OURWINDOW]
```

Watkey waits for a key, or the left mouse button.
Wincl closes a window, in this case, our window.

5.3 Using gadgets.

In this tutorial, we will create a window and add an OK gadget to it. An ASOS
gadget is a small button with some text in it. You can click on a gadget to
do something (Like the OK and CANCEL buttons in most Amiga requesters !).

We start the program in a similar way as before:

```
ABOUT
WINREQ : OURWINDOW = ANS
WINOP [ 40, 40, 600, 160, 0, OURWINDOW, "Gadget" ]
WINTX [ OURWINDOW, "Click on the OK button to exit!", -1, 2, 0 ]
```

Now we have a window with some text in it. Let's add the gadget to the window.

Adding gadgets to windows is simple. You just call ZONREQ to get a free 'zone'
(A zone is where ASOS keeps it's information about gadgets and windows). You

can then use the answer in ANS to create the gadget using WINGAD.

The code to do that is:

```
ZONREQ : OKGADGET = ANS
WINGAD [ OURWINDOW, OKGADGET, "OK", 33, 12, 2 ]
```

Let’s analyse this bit of code:

```
ZONREQ : OKGADGET = ANS This line of code requests a free zone and then names
                        it OKGADGET.
WINGAD [ OURWINDOW, OKGADGET, "OK", 33, 12, 2 ] This line creates the gadget in
                                                the window.
```

Ourwindow is the window number, Okgadget is the zone number. "OK" is the text ↩ to put into
the gadget.

The next three parameters describe where the gadget will be placed and how it ↩ will look.

The first is the X position, The second is the Y position, both using ↩ coordinates as wintx.
The last defines how the gadget will look. For now use type 2, standard gadget.

Now we wish to wait until the gadget is clicked and then end the program. To do ↩ that, we add
the line:

```
WTCLK[OURWINDOW]
```

This line calls upon ASOS’s event handler. It will wait until a gadget in the ↩ window OURWINDOW
is clicked and then continue. In later tutorials, we will show how to use the ↩ event handler
more effectivly.

Now all that’s left is to end the program like we did in the first tutorial:

```
WINCL[OURWINDOW]
```

Wincl also removes the gadget and it’s zone from the screen.

5.4 Using buttons.

In this tutorial, we will create a window, with an OK button and a button on it ↩ . Buttons
are more like switches: they can be either on or off.

First, we start ASOS, open the window and add the OK button:

```
WINREQ : OURWINDOW = ANS
WINOP [ 40, 40, 600, 160, 0, OURWINDOW, "Button" ]
WINTX [ OURWINDOW, "Click on the OK button to exit!", -1, 2, 0 ]
ZONREQ : OKGADGET = ANS
WINGAD [ OURWINDOW, OKGADGET, "OK", 33, 14, 2 ]
```

We now have a window with an OK-button on it. We will now add the button, using ↩
    ZONREQ to get
a free zone and BUTTON to add the button to the window:

```
ZONREQ : OURBUTTON = ANS
BUTTON [ OURWINDOW, OURBUTTON, 3, 200, 32, 0 ]
```

Let’s see how this works:

ZONREQ : OURBUTTON = ANS This line of code requests the zone for our button.
BUTTON [OURWINDOW, OURBUTTON, 3, 275, 50, 0] This line of code adds the ↩
  button.

The last four parameters of the procedure have the following meaning:

The first (3) is for the type of button. For now leave it on 3 (Radiobutton ↩
).
The second (275) is the X coordinate of the button (in pixels from the top- ↩
  left).
The third (50) is the Y coordinate of the button (in pixels from the top- ↩
  left).
The fourth (0) is the number of the ’radiorange’. It can be used to create a ↩
  number
       of buttons, from which only one may be active at any time. ↩
         Leave it
    for now.

Now we will wait for the user to click on the OK button before exiting the ↩
  program
(as we did in earlier tutorials), but this time, we also, before exiting, ↩
  show
the user in what state the button is (On or Off). (We wait a small time ↩
  before
actually stopping the program).

```
WTCLK[OURWINDOW]

If BT(OURBUTTON,5) = 1
   WINTX[ OURWINDOW, "The button was ON", -1, 8, 0 ]
Else
   WINTX[ OURWINDOW, "The button was OFF", -1, 8, 0 ]
End If
Wait 200

WINCL[OURWINDOW]
```

The BT() array contains information about buttons. In index 5 of the array, the ↩
    state of
any button is stored. You can use this to see wether a button is on or off.
(So BT (1, 5) will tell you if button 1 is on (1) or off (0)).

Note that WTCLK will not stop if you press the button, only if you press the OK
gadget.

Wincl also removes buttons from screen and memory.

5.5 Event loop.

In this tutorial, we will create a window with three gadgets: An OK gadget to exit
the program, a Click me gadget and a Click me instead gadget. If a gadget other than
OK is pressed, a text will appear, stating which one.

Let’s first create a window like before:

```
WINREQ : OURWINDOW = ANS
WINOP [ 40, 40, 600, 160, 0, OURWINDOW, "Button" ]
WINTX [ OURWINDOW, "Click on the OK button to exit!", -1, 2, 0 ]
ZONREQ : OKGADGET = ANS
WINGAD [ OURWINDOW, OKGADGET, "OK", 33, 14, 2 ]
```

Now, we add the other two gadgets like we did with the OK gadget:

```
ZONREQ : CLICKME = ANS
WINGAD [ OURWINDOW, CLICKME, "Click me!", 5, 5, 2 ]
ZONREQ : METOO = ANS
WINGAD [ OURWINDOW, METOO, "Click me instead!", 50, 5, 2 ]
```

At this point, it is worth noting that the ZONREQ and WINREQ commands need to be acted on
immediatly, if you use the following code, you will get an error.

Faulty code:

```
ZONREQ : CLICKME = ANS
ZONREQ : METOO = ANS
WINGAD [ OURWINDOW, CLICKME, "Click me!", 2, 5, 2 ]
WINGAD [ OURWINDOW, METOO, "Click me instead!", 50, 5, 2 ]
```

The reason this will not work is that ZONREQ and WINREQ request a free zone-number, but they
do not reserve the zone, so you have to ask for a zone and immidiatly after that create the
zone.

Now, we have to build an event loop. An event loop will check for each gadget if it has been
clicked and act accordingly. In this case, our event loop will display which gadget has been
clicked, or exit if OK is clicked.

The code to do this:

```
Do
   WTCLK[OURWINDOW]
   If FL2 = CLICKME
      Ink 6,6 : Text 446,105,"Thank you!"
      Ink 3,6 : Text 62,105,"Thank you!"
   End If
   If FL2 = METOO
      Ink 3,6 : Text 446,105,"Thank you!"
      Ink 6,6 : Text 62,105,"Thank you!"
   End If
```

```
      Exit If FL2 = OKGADGET
   Loop

   Let's analyse this bit of code:

   Do This begins the loop.
      WTCLK[OURWINDOW] Here, we wait for a click in our window. If a gadget in the
                       window is clicked, WTCLK will exit and put the result in ←
                          FL2.
      If FL2 = CLICKME Here we check if CLICKME was the gadget the user clicked.
         Ink 6,6 : Text 446, 105, "Thank you!" Here we erase the 'thank you' text
                                               under the other gadget. ( Colour 6 ←
                                                  is
               the ASOS window background colour ).
         Ink 3,6 : text 62, 105, "Thank you!" Here we print the 'thank you' text
                                               under our gadget. ( Colour 3 is the
               ASOS white colour ).
      End If Here the first gadget is complete.
      If FL2 = METOO
         Ink 3,6 : Text 446,105,"Thank you!"
         Ink 6,6 : Text 62,105,"Thank you!"
      End If This does the same for the other gadget.
     Exit If FL2 = OKGADGET Here we exit the loop if OK is clicked.

   Now all that's left is to exit the program:

   WINCL[OURWINDOW]
```

5.6 Input fields.

```
   In this tutorial, we will open a window, add an OK button and a single input ←
      field.
   We will also print the content of the input field after each entry. The program ←
       will
   exit on  clicking the OK button.

   First, we create the window and button, like before:

   WINREQ : OURWINDOW = ANS
   WINOP [ 40, 40, 600, 160, 0, OURWINDOW, "Input field" ]
   WINTX [ OURWINDOW, "Click on the OK button to exit!", -1, 2, 0 ]
   ZONREQ : OKGADGET = ANS
   WINGAD [ OURWINDOW, OKGADGET, "OK", 33, 14, 2 ]

   We shall now add the input field to the window. The input field function in ←
      ASOS is
   called ANPUT and can, at times, be a little difficult to understand. First we ←
      request
   a zone using ZONREQ and then, using ANPUT we will create the input field.

   ZONREQ : MYFIELD = ANS
   ANPUT [ OURWINDOW, -MYFIELD, "Text please!", 10, 5, 15, 3, 0, 0, 0, 0 ]

   Let's see what that means:

   ANPUT [ OURWINDOW, -MYFIELD, "Text please!", 10, 5, 15, 3, 0, 0, 0, 0 ]
```

The first parameter is the window in which to create the input field.
The second parameter is the zone of the input field. During the creation of an ANPUT field, this must be the zone number, negative (the - sign).

The third parameter is the default text to display.
The fourth parameter is the X position of the field, like in wintx. (Centering is not supported!)
The fifth parameter is the Y position of the field, like in wintx.
The sixth parameter is the maximum length of the string.
The seventh parameter give the type of input. For now, leave it at 3, which is an unrestricted
string.

The eight, ninth, tenth and eleventh parameter have no meaning during initialization.

Now we need to create an event loop, to see what the user types into the field. This is somewhat
similar to the event-loop above:

```
TXT$="Text please!"

Do
   WTCLK[OURWINDOW]
   If FL2 = MYFIELD
      ANPUT[ OURWINDOW, MYFIELD, TXT$, 10, 5, 15, 3, 0, 0, 0, 0 ]
Ink 6,6 : Text 113,88,TXT$
TXT$=OUT$
Ink 3,6 : Text 113,88,TXT$
   End If
   Exit If FL2 = OKGADGET
Loop
```

The event loop has only a few differences from the original one:

At the very beginning the TXT$ is initialized to the same string as our default string.

The ANPUT command is the same as during the initialization, only now the zone number is
positive. When this is called, the input field will go into input mode, so only after
you have clicked will the input field actually work.

An ASOS input field can do most normal text edit features you'd expect from an input field,
cursor keys, delete and backspace all work. Plus, you can use the mouse to select a postion.
If you press the left mouse button with the mouse cursor over a certain letter in the string,
the cursor will jump to that letter.

5.7 Advanced features.

Most ASOS procedures offer far more options than those that have been discussed above.
You can check those options in the

quick reference guide
.

Power Programs documentation, Page 8

1.9 Advanced usage

Power Programs documentation, A.S.O.S v1.35

6.1 Introduction.

The purpose of this chapter is to expand on several more advanced topics of ASOS. For more information, you can always check the quick reference guide .

6.2 Error-Handling.

ASOS offers generic error handling for all your programs, through the procedure GENERR. This procedure is called by all ASOS procedures on any error, but you can also use it in your programs. By default, ASOS reroutes error handling to GENERR through ADEFAULT. If you wish to do your own error handling, do not forget to remove that entry.

How it works:

You can call GENERR in two ways:

1) Via the On error proc method.
2) Via any AERR=x : GENERR call.

If you use the first, GENERR will only act on known errors. In all other cases, it will display error # 1024, "unknown error". Therefore, you need to add your own lines of handler code for any error which is not already present in GENERR. (A list can be found here)

If you use the second, GENERR will still only respond to known errors and give an error # 1024 for unkown ones, however, there is a fundamental difference in what happens. The first method traps AMOS / OS errors, the second can be used to trap errors occuring in your program.

To add an error to the list, first read the list of errors, to see if the error already exists and to see which error numbers you should use.

Then, use any error as template:

Error 4, Window error:

```
If AERR=1 Then WINTX[0,"Asos error #4: Window error.",-1,3,-1] : Goto ERE
```

This is AERR 1 (window error). You could add any error in this way.

Fatal errors, such as error 6, Software corrupt, are somewhat different:

```
If AERR=3 Then WINTX[0,"Asos error #6: Software corrupt!",-1,3,-1] : RST=True:  ↩
   Goto ERE
```

RST is the reset flag. If it is true, RESET[] will be called.

GENERR calls the Resume Label after exiting, so you know an error has occured. ↩
If you
wish, you can also use L_AERR to see which AERR was the last to occur. Do not, ↩
however
alter this variable, because it is used to determine the end of stack-space as ↩
well.

6.3 Filling out 'user' procedures.

There are a number of 'user' procedures in ASOS:

USERGADDATA
USERRESET
USERHELP

USERGADDATA contains information for cyclegadgets. Here you can fill in the ↩
various
gadget texts you wish them to have. Alter the first data field to suit the ↩
maximum
entries and the following data fields to contain the various cyclegadget texts ↩
you
wish to have.

USERRESET is called when RESET[] is executed. You can put anything nessecary to
restart your program here.

USERHELP is called when GENHELP is called. You can put an online help system ↩
here.
Note: The standard help screen is screen 1.

6.4 ASOS palette.

The standard ASOS palette is as follows:

Colour 0: $001.	(Dark blue)	Default background colour.
Colour 1: $555.	(Medium gray)	Used in gadgets / buttons.
Colour 2: $222.	(Dark gray)	Used in gadgets / buttons.
Colour 3: $FFF.	(White)	Default text colour.
Colour 4: $227.	(Medium blue)	Used in buttons.
Colour 5: $22C.	(Light blue)	Used in buttons.
Colour 6: $300.	(Dark red)	Default window colour.
Colour 7: $000.	(Black)	Used in gadgets / windows / buttons.

```
   These colours can be altered either in ADEFAULT, or by using Palette  ↩
      instructions.
   The choice of these colours can only be altered by editing almost all drawing
   functions in ASOS.

6.5 Writing your own A.S.O.S. procedures.

   Since ASOS is not an extension, you can edit and add any procedure you like.

   There are one or two things to think about:

   All ASOS procedures use GENERR for error handling.
   All ASOS procedures use the ASOS zone commands and structure.

   You should look at the supplied procedures to see how they work.

   If you think your procedure / change is essential, let us know and we will
   consider adding it ( with your name credited of course ) to the product.

Power Programs documentation,                               Page  9
```

1.10 Appendix A: A.S.O.S error list

```
                Power Programs documentation,                           A.S.O.S  ↩
                   v1.35


In this appendix, the complete list of A.S.O.S. errors is found, with a
small explanation what they mean and how to solve them/use them.

Asos Error #1: Config file not found

This error is meant to be used by user programs. It’s given if the
file ’config.asos’ is not found. This file name can be edited.
It’s function is to give you a warning when the configuration
file of your program is not found.

Asos Error #2: File not found

This error is given if the dos error for file not found occurs. It
allows the program to continue even if a file does not exist.

Asos error #3: Out of buffer space!

This fatal error is given if your AMOS program runs out of
variable buffer space. It does not always work properly, because it
might not be detected in time. As with any fatal error, the function
reset[] and the function user_reset[] are called as a result.

Asos error #4: Window error.

This error is given if either a window has wrong properties, has
a too large window-number or a window-number wich is already in use.
The window wich caused this error is NOT opened, but the program
```

continues anyway.

Asos error #5: Zone error.

This error is given if a zone has the wrong properties, a zone with a zone-number wich is already in use or a zone-number wich is too large is tried to be created. The zone will NOT be created, but the program continues anyway. NOTE: The error is also used if a gadget gets the wrong properties.

Asos error #6: Software corrupt!

This error should never happen, but may be used in your programs as a debug feature. It is meant to show that something wrong happened. It is also a fatal error.

Asos error #7: Unknown item clicked.

This error is given if the wtclk[] function, or the other functions having to do with mouse-clicks, detect a zone-hit wich does not exist.

Asos error #8: Out of redraw buffer.

Is this error is given then the window-based redraw buffer has run out. It can be solved by either putting less into the window, combining multiple text-outputs in a single one or by enlarging the redraw buffer.

Asos error #9: Input flag out of range.

This error means an A.S.O.S. function recieved a call in wich one of the properties was out of range (for example a too high window number).

Asos error #10: Window request failed.

The window request given to winreq[] failed. There are no free windows. Check if you really need to open so many windows, or change the maximum number of windows.

Asos error #11: Zone request failed.

The zone request given to zonreq[] failed. There are no free zones. Check if you can free one or multiple zones, or change the maximum number of zones.

Asos error #12: General request failed.

This error is currently not implemented.

Asos error #13: Recursive error!

This fatal error occurs when a single error occurs more then x times. In this case there is a high probability that the same line of code created the error. It is given to prevent those nasty 'out of stack space' errors.

Errors #14-256 (AERR numbers 11- 253) are reserved for future use.

```
Note:

Errors 257-511 and 513-1023 ( AERR 257-511 and 513-1023 ) are reserved
for user programs. Errors 256 and 512 are included for shareware
software and unfinished software. Error 1024 happens after a call
to GENERR without a specific error number, or after a call to
GENERR with an invalid error number.

For instance:

AERR=32768 : GENERR : Rem Incorrect error number yields unkown errors.

Asos error #256: Shareware version!

This is a user error. If you give AERR=256 and call GENERR, you cause this
non-fatal error to occur. It is a message for the user that it’s time to
register their software. It does not influence the rest of your program
in ANY way.

Asos error #512: Option not implemented!

This is also a user error. It can be used to supplement error 256 or it
can be used during programming or beta-release. It is no more than a
message to yourself/your user. It does not influence the rest of your
program in ANY way.

Asos error #1024: Unknown error!

This error is given when GENERR is called without a valid error number in
the variable AERR. It really shouldn’t happen in any of the demos/examples
but it may happen in your programs when you give a wrong error-no, or
if you call GENERR without an error number.

Note:

It is possible in to add errors to the generr procedure.
For information on how to achieve this, read
                chapter 6
                 on error
editing.

Power Programs documentation,                                  Page 10
```

1.11 Appendix B: Quick Reference guide

```
Power Programs documentation,                            A.S.O.S v1.35


Asos general procedures. Revision: 4

Procedure ABOUT

   This procedure opens a window with some information about ASOS
```

Either this procedure, or ADEFAULT should be run at least once.

Procedure ADEFAULT

This procedure opens the default ASOS screen. It also sets the default palette, loads the default ASOS icons and does On Error Proc GENERR, for error handling.

Zone-structure procedures. Revision: 8

Procedure STZN[NUM,X1,Y1,X2,Y2]

This procedure sets (or alters) an ASOS zone. It should not be used in your programs unless you wish to use the ASOS zone detection routine (MSZON) to check on a part of the screen.

Usage:

```
STZN[NUM,X1,Y1,X2,Y2]

NUM = The number of the zone to set. Must be smaller than MAX_ZON.
X1  = The X coordinate of the upper left corner of the zone.
Y1  = The Y coordinate of the upper left corner of the zone.
X2  = The X coordinate of the lower right corner of the zone.
Y2  = The Y coordinate of the lower right corner of the zone.
```

Procedure MSZON

This procedure delivers the zone number the mouse is in, in the variable FL3. It also delivers the window number the mouse is in in the variable FL6.

Usage:

```
MSZON : WINDOW_IN=FL6 : ZONE_IN=FL3
```

Call MSZON when you wish to check wich zone (ASOS zone) the mouse is in.

Procedure RSTZN[NO1,NO2,WNFL]

This procedure resets one or more ASOS zones, as set by STZN. It does not reset AMOS zones.

Usage:

```
RSTZN[NO1,NO2,WNFL]

NO1  = First zone to be reset.
NO2  = Last zone to be reset. If you only wish to reset one zone,
       this number needs to be equal to NO1.
WNFL = If this flag is one, instead of resetting one zone,
   all the zones associated with A.S.O.S. window number NO1
```

```
     are reset. This is used to close windows.

    See also:  STZN


Procedure WINREQ

    This procedure requests a free window zone number.

    Usage:

    WINREQ : WINDOW=ANS

    ANS = The returned window zone number. This is zero if the call
      failed.

    Notes:

      The returned zone number is not reserved, so you cannot
      call this procedure twice in a row without first initializing the
      returned zone number.

    See also:  ZONREQ


Procedure ZONREQ

    This procedure requests a free gadget zone number.

    Usage:

    ZONREQ : GADGETZONE=ANS

    ANS = The returned gadget zone number. This is zero if the call
          failed.

    Notes:

      The returned zone number is not reserved, so you cannot
      call this procedure twice in a row without first initializing the
      returned zone number.

    See also:  WINREQ


Window-handler procedures. Revision: 7

Procedure WINOP[X,Y,X2,Y2,TP,NUM,TI$]

    This procedure opens an ASOS window on screen. It may also
    set a number of zones, depending on window type number.

    Usage:

    WINOP[X,Y,X2,Y2,TP,NUM,TI$]

    X   = Upper left corner X coordinate ( will be rounded to nearest 8 )
```

```
Y   = Upper left corner Y coordinate ( will be rounded to nearest 8 )
X2  = Lower right corner X coordinate ( will be rounded to nearest 8 )
Y2  = Lower right corner Y coordinate ( will be rounded to nearest 8 )
TP  = A number describing the type of window. Currently supported are:
      0 = Normal window. No Gadgets will be created.
1 = Closable window. Contains a close gadget in upper left corner.
2 = Resizable window. Contains a resize gadget in lower right corner.
3 = Closable, resizable window. Contains a close gadget in upper left
    corner. Also contains a resize gadget in lower right corner.
      4 = Error window. Not to be used except by procedure GENERR and
    RESET.
NUM = The window number. As gained by winreq, or smaller than MAX_WIN.
      This number needs to point to a free window zone.
TI$ = (Optional) String containing the title of the window. This title
      must fit the window.

See also:  WINREQ
           WINCL
```

```
Procedure WINCL[NUM]

   This procedure closes a window, and resets all it’s gadgets. A closed
   window is removed from the screen.

   Usage:

   WINCL[NUM]

   NUM = Window zone number to be closed. Can be -1 to close ALL windows
         on screen.

   See also:  WINREQ
              WINOP
```

```
Procedure WINTX[WIN,TX$,X,Y,MD]

   This procedure adds text to an open ASOS window, as well as adding that
   text into the redraw buffer of that window.

   Usage:

   WINTX[WIN,TX$,X,Y,MD]

   WIN = A valid ASOS window.
   TX$ = Any valid AMOS string. This string has to fit within the window.
   X   = The x position relative to the topleft of the window. A value of
         -1 will center the string in the window. ( 1 unit is 8 pixels )
   Y   = The y position relative to the topleft of the window. ( 1 unit is 8
         pixels )
   MD  = The text mode you wish to use:
         0 = Standard text.
   1 = Inverse ( Background-in-white instead of white-on-background )
   2 = Shade ( text will be ’ghosted’ out )
   4 = Underline ( A line will be drawn under the text )
   These modes can be combined by adding them together ( i.e. 6 for
   shade-underlined text. )

   Notes:

      Multiple WINTX[] calls can be made to the same horizontal line ( i.e.
      WINTX[1,"hello",10,10,0]
      WINTX[1,"world",16,10,4] will result in:
      hello world being printed in window 1, starting at position 10,10 )

      WINTX[] should not be used for often changing data on screen. The
      reason for this is simple: WINTX[] calls are buffered into the
      ( limited ) redraw buffer. Thus, printing 10 different values on
      the same position will result in showing all these 10 values in the
      order printed with each redraw. What's worse is that this can result
      in 'Out of redraw buffer' system messages. Use the AMOS text command
      for these purposes.

   See also:  WINOP
              WINRED

Procedure WINCLR[WIN,FL]

   This procedure is used to clear a window. All text, gadgets and graphics
   in the window will be removed. Note: Depending on the value of FL, the gadgets
   and window text will remain active.

   Usage:

   WINCLR[WIN,FL]

   WIN = A valid ASOS window
   FL  = This flag influences how the window will be cleared.
         0 = The inside of the window ( up to the border ) will be redrawn,
       excluding any gadgets, text or graphics. Gadgets and text will
       be removed from any redraw buffers or zones.
   1 = The window will be redrawn, excluding any gadgets, text or graphics.
       Gadgets and text will still be available in their redraw buffer and
       zones.
   2 = The window will be redrawn, excluding any gadgets, text or graphics.
       Gadgets and text will be removed from any redraw buffers or zones.

   Notes: Using method 1 while gadgets exists will result in the redrawing of
          these gadgets as the user presses them.
    Using method 1 while wintx[] text still exists in the window, will cause
    this text to reappear whenever a redraw[] is used by the system or user
    program.

   See also:  WINTX
              WINGAD
        REDRAW
        WINRED

Procedure WINSZ[WIN]

   This procedure is used to allow the user to resize a window. When called,
   the user gets a 'growbox' outline of the current window, which can be
   resized using the mouse.
   Usage:

      WINSZ[WIN]

      WIN = A valid ASOS window.

   Notes: This function should be called whenever a resize gadget is pressed
          by the user.

    The window content is reset by calling this function, so any gadget
    will have to be replaced by the user-software. The same goes for text.

    Wintx[] text which is placed in a resizable window can exceed window
    boundaries.

   See also:  WINOP

Gadget-structure procedures. Revision: 11

Procedure WINGAD[WIN,NUM,GD$,X,Y,R]

   This procedure creates a text gadget inside a window.

   Usage:

     WINGAD[WIN,NUM,GD$,X,Y,R]

     WIN = A valid ASOS window.
     NUM = A valid, unused ASOS zone. You can use ZONREQ to get such
           a zone number.
     GD$ = The text in the gadget. For example, "OK".
     X   = The x position of the gadget, relative to the top-left of
           the window. ( 1 unit is 8 pixels )
     Y   = The y position of the gadget, relative to the top-left of
           the window. ( 1 unit is 8 pixels )
     R   = The type of gadget:
           0  = Borderless gadget. The gadget will appear as a gray block
          with the text in it.
     1  = Thin Bordered gadget. The gadget will have a thin black
          border around the gray block with the text in.
     2  = 3D-Look gadget. The gadget will have a 3D look to it.
     3  = ANPUT gadget. Reserved for the ASOS input routine. Do not
          use.
     4+ = Text only gadget. The gadget will appear exactly as standard
          text. You can, however still click on it.

   See also:  WINCLR
              WINOP

Procedure CYCLEGAD[WIN,NUM,GD1,GD2,X,Y,R]

   This procedure creates content cycling text gadget in a valid window. For
   this function to work, the content must be added to USERGADDATA as outlined
   in chapter 6, on filling 'user' procedures.

   Usage:
     CYCLEGAD[WIN,NUM,GD1,GD2,X,Y,R]

     WIN = A valid ASOS window.
     NUM = A valid, unused ASOS zone.
     GD1 = The first gadget entry to use, from USERGADDATA.
     GD2 = The last gadget entry to use, from USERGADDATA.
     X   = The x position of the gadget ( see WINGAD ).
     Y   = The y position of the gadget ( see WINGAD ).
     R   = The gadget look ( see WINGAD ).

   See also:  WINGAD
              WINCLR
        CYCLEUPD
        CYCLEGET

Procedure CYCLEUPD[NUM]

   This procedure updates the contents of the named CYCLEGAD. It should be
   called after every 'click' on the gadget.

   Usage:

     CYCLEGAD[NUM]

     NUM = A valid, used CYCLEGAD.

   See also:  CYCLEGAD
              CYCLEGET

Procedure CYCLEGET[NUM]

   This procedure gets the current value of the named CYCLEGAD.

   Usage:

      CYCLEGET[NUM] : GADSTATUS = ANS

      NUM = A valid, used CYCLEGAD.
      ANS = The current content value. For example:

            A cyclegad with GD1 = 10 and GD2 = 14 will have possible
      content values 1 ( which equals to GD1 (10) ) through
      5 ( which equals to GD2 (14) ). In other words, the content
      of a cyclegad is equal to the difference between GD1 and
      the current value. This in turn is equal to the amount of
      'clicks' between the initial state and the current state.
      See chapter 5 on 'gadget use' for more information.

   See also:  CYCLEGAD
              CYCLEUPD

Procedure INVGAD[NUM]

   This procedure redraws the named gadget in it's 'clicked on' state.

   Usage:
      INVGAD[NUM]

      NUM = A valid, used gadget.

   Notes: This procedure is used mostly internal and should not really
          be nessecary, unless you plan to write your own event handler.

   See also:  RESTGAD

Procedure RESTGAD[NUM]

   This procedure redraws the named gadget in it's default state.

   Usage:

      RESTGAD[NUM]

      NUM = A valid, used gadget.

   Notes: This procedure is used mostly internal and should not really
          be nessecary, unless you plan to write your own event handler /
    redraw handler.

   See also:  INVGAD
              REDRAW
        WINRED

Procedure BUTTON[WIN,NUM,TYPE,X,Y,RADIORANGE]

   This procedure is used to create a button in a window. There are a number
   of button types which can be created.

   Usage:

     BUTTON[WIN,NUM,TYPE,X,Y,RADIORANGE]

     WIN        = A valid ASOS window.
     NUM        = A valid, empty zone number. You can use ZONREQ to get
                  such a number.
     TYPE       = The type of button to be created:

                  0 = Undefined. Do not use.
            1 = Close gadget. Use this to generate a close gadget
          in the window, if it is not already present.
            2 = Resize gadget. Use this to generate a resize
          gadget into the window, if it is not already
          present. Note: the window will not become a
          resizable window, so text e.d. will still have
          to fit.
            3 = User button. This button can be used as a switch,
          or as a radioswitch.
     X          = The x value of the button, relative to the top-left of the
                  window. ( 1 unit is 1 pixel ).
     Y          = The y value of the button, relative to the top-left of the
                  window. ( 1 unit is 1 pixel ).
     RADIORANGE = This value sets the 'Radioswitch' range to which the button
```

```
              belongs. This can be any positive number above zero. If a
      button has a RADIORANGE value above zero and it is clicked,
      all other buttons with the same RADIORANGE value will be
      cleared, so that only the last clicked button applies.

   Notes: More information about creating buttons, their values and the
          RADIORANGE variable can be found in chapter 5, under 'creating
    buttons'

   See also:  BTUPD
              BTDRAW
        WINOP

Procedure BTUPD[NUM]

   This procedure changes the state of the named button and, if applicable,
   the state of all associated RADIORANGE buttons as well.

   Usage:

     BTUPD[NUM]

     NUM = A valid button zone. If the button was active, it will be
           deactivated. If the button was inactive, it will be activated
     and all associated RADIORANGE buttons will be deactivated.

   See also:  BUTTON
              BTDRAW

Procedure BTDRAW[NUM]

   This procedure (re)draws a valid button in it's window. It is used
   mostly internal, but could be used in custom redraw handlers and the
   like.

   Usage:

     BTDRAW[NUM]

     NUM = A valid button zone number. The button will be drawn in it's
           current state.

   See also:  BUTTON
              BTUPD

Input-handler procedures. Revision: 13

Procedure ANPUT[WIN,NUM,DEF$,NX,Y,MX,SPC,CU,CD,CL,CR]

   This procedure generates an input field in a ASOS window. The way this
   function works is a bit special and can be a bit hard to understand.
   It is recommended that you read chapter 5 on 'input fields' before
   using this function.

   Return values:

   OUT$ = The returned string, changed or not.
```

```
LU   = If this is true then Cursor up is pressed.
LD   = If this is true then Cursor down is pressed.
LF   = If this is true then Enter has been pressed.
CX   = This variable contains the last X postion that the
       input cursor was on.

All these variables are global variables which can be used to get
the contents of the input field, or to link multiple input fields
together.

Usage, initialization:

  ANPUT[WIN,-NUM,DEF$,NX,Y,MX,SPC,CU,CD,CL,CR]

  WIN  = A valid ASOS window.
  NUM  = A valid, empty ASOS zone. Such a zone can be aquired using
         ZONREQ.
  DEF$ = The default string to be used. This string must be shorther
         than MX.
  NX   = Starting position of the ANPUT field, relative to the top-left
         of the window ( 1 unit is 8 pixels ).
  Y    = The Y postion of the ANPUT field, relative to the top-left
         of the window ( 1 unit is 8 pixels ).
  MX   = The length of the ANPUT field ( 1 unit is 1 character ).
  SPC  = The type of input to expect:
         0  = Undefined. Do not use.
   1  = Character string. May contain a..z, A..Z, 0..9, ' '.
   2  = Numerical string. May contain 0..9.
   3  = Open string. May contain all characters.
   4  = Complex numerical string. May contain -, ., 0..9.
  CU   = Undefined during initialization.
  CD   = Undefined during initialization.
  CL   = Undefined during initialization.
  CR   = Undefined during initialization.

Usage, redraw:

  ANPUT[WIN,0,DEF$,GD(NUM,2),GD(NUM,3),PSP(NUM,1),PSP(NUM,2),0,0,0,0]

  WIN         = A valid ASOS window.
  DEF$        = The last string to be input into the field.
  GD(NUM,2)   = This is equal to NX. You can also use the old value of NX
                here.
  GD(NUM,3)   = This is equal to Y. You can also use the old value of Y
                here.
  PSP(NUM,1)  = This is equal to MX. You can also use the old value of
                MX here.
  PSP(NUM,2)  = This is equal to SPC. You can also use the old value of
                SPC here.
  CU,CD,CL,CR = These are undefinded during the redraw phase.

Usage, clicked:

  ANPUT[WIN,NUM,DEF$,NX,Y,MX,SPC,CU,CD,CL,CR]

  WIN  = A valid ASOS window.
  NUM  = A valid, ANPUT zone.
```

```
     DEF$ = The string to be edit.
     NX   = The X position of the input field.
     Y    = The Y position of the input field.
     MX   = The length of the input field.
     SPC  = The expected input type ( see above ).
     CU   = If this is 1, the global variable LU will be set to true if
            the user presses cursor up. This can be used to link input
      fields together.
     CD   = If this is 1, the global variable LD will be set to true if
            the user presses cursor down.
     CL   = If this is 1, the global variable LU will be set to true if
            the user presses cursor left, while on the first input position
      of the input field. CX will be set to -1.
     CR   = If this is 1, the global variable LD will be set to true if
            the user presses cursor right, while on the last input position
      of the input field. CX will be set to 0.

   Usage, draw active:

     ANPUT[WIN,-9999,DEF$,NX,Y,MX,0,0,0,0,0]

     WIN  = A valid ASOS window.
     DEF$ = The text to be drawn.
     NX   = The X position of the input field.
     Y    = The Y position of the input field.
     MX   = The length of the input field.


   Notes: ANPUT should be used in the following order:

    - First call with the initialization parameters.

    - At each 'click' on the input field, call with the clicked
      parameters.

    - At the end of each event loop, call with the the redraw
      parameters.

   The draw active version should not be used, execpt in a new
   event handler.

Procedure WTCLK[WIN]

   This is the standard ASOS event handler, it can be used as a template
   for your own event handler. It waits for mouse clicks and interprets them
   for the supplied window number. Gadgets and buttons are updated accordingly.
   Redraw functions may be called by this function.

   Usage:

     WTCLK[WIN]

     WIN = A valid ASOS window.

   Result:

     FL2: Contains the clicked zone number.
```

Notes:

This procedure waits for one click then exits, thus a call of this function to a window without any other zones will never return.

Procedure WATKEY

This is the standard wait for input function for use in your programs. It waits for both mouse keys and keyboard entry.

Usage:

```
WATKEY
```

Result:

```
ANS$: Contains the pressed key as string. ( For instance: "A" )
      If ANS$ is empty ( "" ) then the mouse key has been pressed.
```

Error-handler procedures. Revision: 11

Procedure GENERR

This procedure is ASOS's standard error handler. All errors are by default trapped to this location (On Error). If the error is a known error, the error window will contain a description of the error. A list of errors can be found in Appendix a. The error handler can be called by your own procedures as well.

Usage:

```
AERR = x : GENERR

AERR = An ASOS error code.
```

Result:

```
RST = x: If this variable is true, the error was a fatal one.
         you should probably quit or restart your program on
   errors of this type.
```

Notes:

This procedure is automattically called by most ASOS functions in case of error. It can also be called by any run-time error. This is defined in ADEFAULT.

This procedure attempts to 'intelligently' trap errors, so that a program will not run out of stack space. If the same error happens to often, the procedure will state a special 'too many errors' type error, which is treated as a fatal error.

On fatal errors, the procedures RESET and USERRESET are called.

Procedure RESET[NUM]

```
   This procedure handles the 'resetting' of ASOS. Since ASOS does not really
   reset, all it does is close all windows, clear all zones and flags, as well
   as the screen. It can also be used to quit a program.

   This procedure calles the user function USERRESET.

   Usage:

     RESET[NUM]

     NUM = 0: This reset is for voluntary reseting. It asks wether you wish to
              reset before actually doing anything.
     NUM = 1: This reset is for fatal errors. The reset is no longer voluntary.
     NUM = 2: This reset can be used for a non-voluntary reset, where there
              has not actually been any error.
     NUM = 3: This reset takes form of a Quit question ( yes/no ).

   Result:

     Reset does not return.

   Notes:

     Since reset[] does not return, you should be carefull around quit and other
     voluntary resets. You could use USERRESET to set a flag or something so you
     know the reset has actually happened.

Procedure QUIT

   This procedure calls Reset[3] for you.

   Notes:

     See reset[]

Redraw-handler procedures. Revision: 4

Procedure REDRAW[NUM]

   This procedure is the ASOS redraw handler. It can be used to redraw ASOS  ↩
      windows
   and / or screens.

   Usage:

     REDRAW[NUM]

     NUM = 0: Clears the screen and redraws all ASOS windows.
     NUM = -1: Redraws all ASOS windows, without clearing the screen.
     NUM = x ( where x>0 ): Redraw ASOS window x.

   Notes:

     Redraw only redraws standard ASOS windows, gadget and buttons.

Procedure WINRED[NUM]
```

```
   This procedure redraws a single window, including all WINTX[]texts
   which have been written into it.

   Usage:

     WINRED[NUM]

     NUM = x : Window to redraw.

Procedure CLREDRAW[NUM]

   This procedure is used to clear the redraw-text buffer of windows.

   Usage:

     CLREDRAW[NUM]

     NUM = 0: Clear the redraw buffer of all active ASOS windows.
     NUM = x ( where x > 0 ): Clear the redraw buffer of window x.

Procedure CLEAR

   This procedure clears the screen. Any ASOS windows on it are just
   invisible, they are not removed using this function.

Help-handler procedures. Revision: 1

Procedure SETHELP[WIN,NUM]

   This procedure sets a zone to be seen as 'help' button, for the other
   help procedures.

   Usage:

     SETHELP[WIN,NUM]

     WIN : A valid ASOS window.
     NUM : A valis ASOS gadget zone.

Procedure DELHELP[WIN]

   This procedure removes the 'help' button from a window.

   Usage:

     DELHELP[WIN]

     WIN : A valid ASOS window.

Procedure GENHELP

   This procedure opens a screen and calls USERHELP. The user help
   function should then display any available information, in a
   context sensitive way.

   Usage:
     GENHELP

External procedures. Revision: 1

Procedure GROWBOX

   This procedure, which is a version of the GROWBOX procedure found
   in many AMOS examples, is used to draw bounding boxes.

Procedure EXEC[C$]

   This procedure can be used to execute an Amiga program.

   USAGE:

      EXEC[C$]

      C$: The complete command path and any command parameters.

Power Programs documentation,                                  Page 11
```

1.12 Appendix C: Known bugs and limitations

```
                Power Programs documentation,                            A.S.O.S  ↩
                   v1.35


Like any software product, ASOS v1.35 contains bugs. There are also a few
procedures which do not work completely.

Known bugs:

The ANPUT[] procedure has a number of known bugs:

  - When pressing [DEL] at the last possible position in any input field,
    the key will not work. Instead, you will have to move past the last
    character and use [BACKSPACE].

  - When ANPUT[] is used for numbers only, the display sometimes moves to
    the left after exiting the field, only to move back to the right when
    re-entering the field.

The WTCLK[] procedure does not handle close / resize buttons accurately in
all cases.

There are likely more (small) bugs in ASOS v1.35. If you find one, send us
a detailed bug-report.

Known limitations:

ASOS only really works correctly is used on screen 0. ASOS doesn’t really like
double-buffered screens.

The event-handler (WTCLK[]) only handles one window at a time and cannot
be used concurrently.
```

```
If you find other procedures that work without problem, but not upto the named
specifications ( see the
                quick reference guide
                 ), send us a detailed report.

Power Programs documentation,                                Page 12
```

1.13 Appendix D: List of examples and demos.

```
Power Programs documentation,                          A.S.O.S v1.35


Currently there are no real examples, other than the tutorials and the demo.
Combining these does give an overview of ASOS's capabilities. In future versions,
examples will be included.

A note about Mandelbrot.amos, Mandelbrot_old.amos and Mandelbrot:

This 'package' has been included to show you the program which inspired ASOS in ↩
   the
first place. The programs are not very good and contain more than a few bugs, but
they also contain the very first GUI system we wrote. Enjoy!


Power Programs documentation,                                Page 12
```

1.14 Appendix E: History.

```
Power Programs documentation,                          A.S.O.S v1.35

   ASOS v1.35:            First public release

      Removed multiple bugs from the gadgets system.
      Added cyclegadgets.
      Added generic help support.
      Changed the way buttons work.
      Added standard icons for close buttons and the like.
      Removed registration limits.
      Removed multiple bugs from the generic input field system.
      Removed a bug from WTCLK[] dealing with buttons.

   ASOS v1.34:

      Changed the palette.
      Removed bugs from the window system and the buttons system.
      Implemented RESET.
      Updated the generic error handler.

   ASOS v1.30:            Base version
```

```
      Added the generic input field system.
      Added buttons.
      Removed several dozens of bugs accros the entire package.

   ASOS v1.00 and lower:   Amos window system

      Implemented every section but buttons.
      Toyed around with icons, but later removed them.
      Tried to create a process communication system, so ASOS could
            become a full GUI system, instead of an AMOS 'extension'
      Build the core package.


Power Programs documentation,                                  Page 12
```

1.15 Appendix F: Future.

```
Power Programs documentation,                            A.S.O.S v1.35

   Plans for v1.36 onwards:

      Create a colour system.
      Update ANPUT to work better.
      Remove bugs.
      Implement better button support.
      Better help system.

   And much more...


Power Programs documentation,                                  Page 12
```