# *Interface*

## Das Resource Construction Kit


no Software GmbH

***Author: Olaf Meisiek - 1994***

***Scan: Dhelberg – English Translation: DrCoolZic - 2019***

*V1.0 – December 2019*

# Interface

## The resource editor

***Author: Olaf Meisiek – Edition 1994***

***Original Scan: Dhelberg – English Translation: DrCoolZic - 2019***

Distribution: **nol Software GmbH** - Copyright 1994 by **nol Software Ltd**, Prüm - Made in Germany

## Copyright



## Guarantees

The software and manual have been created with the utmost care. Only with program errors, which make working with an interface impossible, **nol Software GmbH** provides free replacement after the return of the system diskettes.

## Disclaimer of liability

**nol Software GmbH** assumes no liability for sequences which are due to program error or incorrect entries in the manual. The reproduction of trade names, trade names or other markings in this book does not justify the assumption that they may be used freely by anyone. It may also be registered trademarks or other protected marks, if they are not marked as such.

# Interface Features

**Interface permits the creation and modification of Menus, Dialogs, Alert, Icons, etc. in Resource files**

By its easy service and a mighty function extent Interface is the standard for Resource Construction kits.

- 100% GEM, runs under all TOS systems (ST/TT/Falcon/Medusa)
- Support of multitasking by switchable desktop, drag drop etc.
- Use according to clipboards: Cut/Copy/Paste of text and graphics
- Window Dialogs and clear design the Dialog box
- The Resource size is (almost) boundless
- Saving of the Resource files in the formats RSD, DEF, DFN, RSH and HRD

**Programming languages**

- Export formats for C, Pascal, Modula 2, BASIC and GFA BASIC
- Direct integration of Resource files in your own programs
- In the test mode your own Dialog routines can be integrated

**The icon editor**

- Icons can be created in 2,4,16, and 256 colors
- Separated editing of mask and normal / selected icon
- Import of Windows icons as well as IMG graphics
- Treatment functions: Scrolling, Centering, Mirroring, Cut/Copy/Paste
- Character functions: Line, Point, Rectangle, Circle, Filling
- Compilation of desktop icon files (ATARI, Gemini, no Desk***)***

**Modern Dialog windows made easy with MyDials**

Together with Interface you receive MyDial, a complete C-Library. It allows an integration of the Dialog routines from Interface in your own programs.

- Flying-Dialogs, Popups, extended object types for clear design of Dialog boxes, e.g. title boxes and frames
- Advanced editing for text up to a length of 256 characters
- Switchable 3D-representation in the modern “grey look”, as well as support Functions for window Dialogs
- Complete keyboard operation for your own programs
- Finished Dialogs can be immediately checked in interface in the test mode

# Contents

# 1 Introduction

## 1.1 Review and Outlook

After Interface saw the light at the ATARI fair 1991, an amount of improvement and wishes have reached us. In the current version we have tried to realize most of them. So here again our thanks to all who did not let up and managed to convince us, and especially Olaf Meisiek, the programmer of Interface - of their desire to change. We have adapted the program as far as possible to the new multitasking operating systems such as Mag!X, MultiGEM and MultiTOS. If you have suggestions for improving Interface that are of general interest, we would be happy to hear from you.

## Overview

The purpose of this manual is to enable you to use the various functions that **Interface version 2.33** offers. Although great importance was attached during the development of the program to provide simple and consistent operation, a brief training is required to master all Interface's functions. The purpose of this manual is to explain in detail the concept and operation of the program. It is organized in the following chapter:

**Chapter 1 - Introduction**

Is the introduction you're reading right now.

**Chapter 2 – Working with the interface**

Working with Interface, explains in the first section what is a resource, describes Interface's desktop and explains the special functions of the mouse and Dialogs of Interface. The second section introduces you to the most important technical terms and shows you how to build and design a simple Menu and Dialog. The third section then goes into detail and explains the intricacies of designing a resource. Finally, the icon editor is explained.

**Chapter 3 – Interface's reference**

Is the reference part that summarizes all the Menu options, as well as explanation about saving a Resource.

**Chapter 4 – Icon editor's reference**

Is the reference part for the Icon Editor.

**Chapter 5 – Adjusting Interface**

Explains how you can customize Interface to your own needs (keyboard layout, header file, file name extensions).

**Chapter 6 – The Interface's File Selector**

finally explains the operation of the file selection of Interface.

**Chapter 7 – Creating an EXTOBFIX proram**

Explains how to program your own 'EXTOBFIX' program for your extended object types.

**Chapter 8 – Interface "Remote controlled"**

Remote-controlled interface, points to another way to extend Interface: The Accessory pipeline.

**Chapter 9 – A resource program as an example**

Shows a small C program that loads a resource, displays and evaluates a Dialog.

**Chapter 9 – The MyDial library**

Documents the MyDials library. It provides you with the capability to put in your programs "flying Dialog", Popups, keyboard control in Dialogs, color icons and the like. It also explains the 'XRSRC' routines that support color icons under any TOS version.

This manual does not explain one thing, namely the basic use of resource files. This would go far beyond the scope of the manual. Instead, we recommend purchasing two books that you can hardly do without when programming. The first, the 'ATARI Profibuch' from the Sybex Verlag (ISBN 3-88745-888-5), is an absolute must for every programmer. It documents on 1,491 pages (no typo) the hardware and software of ATARI STs, STEs and TTs. The second book comes from the Hüthig Book Verlag and is called 'From beginner to GEM professional' (ISBN 3-7785-1792-9). It includes a very good C library in the source code (on disk), which can be used for example. used by Interface and the MyDials. The MyDials are a function library that we deliver with Interface.

## 1.2 Dealing with the manual

In this manual, the writing has a particular importance. For example, if we want to tell you that the 'Control' key should be pressed together with the 'O' key, we simply write ‘Control O’. If we describe an option from a Menu, the text does not say “Open…” from the “File” Menu ..., but the option **File: Open...**. Buttons and similar elements are also displayed in a highlighted font, so that you can already orientate yourself on the design.

## 1.3 Registration Card

Important note: Please return the attached registration card to our sales department immediately. Interface is a product that is constantly evolving. Due to the complexity of the program we cannot rule out, despite the greatest care, that small errors occur during the application. You should then notify us, and we will ensure that this error is corrected as soon as possible and that the corrected program is made available to you. Extended versions of Interface can be obtained for a small fee as an update or upgrade, if you are registered as a user with us, so have sent your registration card. Our telephone and written help is also available only to registered users.

## 1.4 Criticism, advice and action

Also for criticism - which can certainly be positive - and suggestions for improvement, we are grateful. But we would like to make a small request: Describe possible mistakes, suggestions and wishes as exactly as possible.

Especially in the event of an error we ask you to specify your computer configuration exactly. This also includes the naming of all programs that were started from the Auto folder or as Accessory. In multitasking operating systems, it is also important to know which programs ran parallel to the interface.

Whenever you - in the context of Interface - are faced with a situation in which you can get stuck without help, we are at your disposal. Just call us during normal office hours.

However, most questions can be answered by reading the manual or the help texts that provide the main Dialog of Interface. Before a call, check whether you do not receive the desired information via the index.

If you want to ask in writing for advice, we ask you to describe your difficulties as comprehensively as possible. So you avoid misunderstandings. If possible, please attach a floppy disk with the appropriate resource so that we can check how you can get the desired result.

## 1.5 You should read that

Among other things, your program disk contains a file with the descriptive name `README.TXT`. You should read it, because in this text you will find the latest information on Interfaces that may differ from this manual. In this case, the text found in the 'README' file applies.

# 2 Working with Interface

In this section, you will read an introduction to working with **Interface**. After a description of the general operation and some basic explanations, it continues with two exercises. In these exercises, you will learn everything about building a simple resource file.

## 2.1 Interface installation

Before you start Interface for the first time, you should make a backup copy of the program diskette. In the case of a later floppy error saves you a lot of trouble. Experience shows that floppy disks (hard disks, printers, etc.) are always broken when they are urgently needed. Of course you are not allowed to sell or give away copies of Interface (or the manual). If you want to copy floppy disks, you may want to refer to the manual of your computer. Since Interface is not copy-protected, you can also install the program (consisting of the files `INTRFACE.PRG` and `INTRFACE.RSC` and optionally `EXTOBFIX.PRG`) on a hard disk.

The scope of delivery of Interface includes more files, but the three just mentioned are quite enough for the beginning. Only if you work with special libraries (function collections), you need more files. These may be in packaged form (as `*.TOS`) on the interface disks. Copy them to your hard disk and start these “programs” by double clicking. These are self-extracting archives, which means that after executing the program you will find the unpacked files on your hard disk.

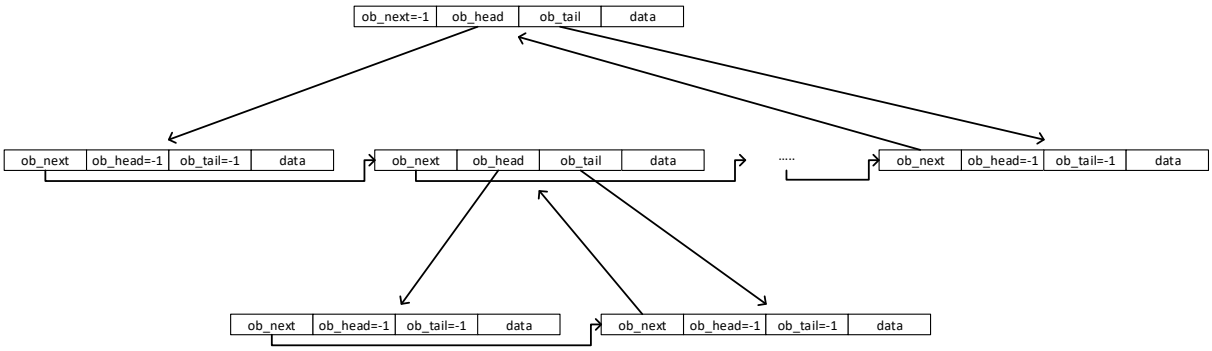
## 2.2 Overview

**What is a Resource?**

Interface is a program that can be used to develop resource files (Menus, Dialogs, Alert boxes, etc.) for GEM programs. Resources in GEM programs are normally everything that appears on the screen except windows and their contents, e.g. Dialogs, Pulldown, Menus, desktop icons or Alert boxes. Resource data are generally not in the main program, but are stored in a separate file with the extension RSC. This has several advantages:

- A resource file can be developed up and changed without programming knowledge.
- The resource file can (at least partially) be changed without having to recompile the main program.
- A program can support different languages without having to change the program code. For each language, only a separate resource file needs to be set up.

Resources consist of 'objects'. An object is a single element of a Menu or Dialog, e.g. an empty frame, text-containing frames, text or buttons, icons, etc. ... To create a Pulldown Menu or a Dialog, combine several objects into a so-called 'object tree'. The term 'tree' derives from the fact that the linking of the objects with each other is a tree-like, it has branched structure:




The first object in a tree is called 'parent', the objects inside a parent are the 'children'. Children objects can also themselves contain other objects to which they then behave like a parent object. The first object of a complete object tree is also called the 'root' object.

A program used to build and modify resource files is commonly called a resource construction set (RCS) editor. With Interface you can create the following tree types:

- Pulldown Menus: These are groups of choices available to the user in a program. The individual Menu items are usually text.
- Dialogs: A Dialog is used to display and / or retrieve information (entries must be made).
- Alerts: An Alert is a special kind of Dialog. It is used to indicate important hints, warnings or errors.
- Free Strings / Images: Here you can save individual texts or graphics in a resource, which have to be managed by the program itself.

The ATARI RCS from version 2.0 (part of the official development package of ATARI) still knows the tree type: 'PANEL', which hardly differs from a Dialog. According to Digital Research (the developer of the ATARI RCS), objects in Dialogs are always accurate to the letter and objects in panels are always positioned with pixel precision. Since this is not supported by any other resource editor and it is also possible to position objects pixel-exactly in Dialogs, this tree type is not explicitly available in Interface. It corresponds rather to a Dialog that was constructed in the pixel grid.

## 2.3 Terminology used in this manual

Just to make sure we talk the same language, we would like to introduce some terms used in this manual. If you are already familiar with the ATARI ST, you will already know most of them.

- Mouse click: Press the left mouse button briefly. A 'mouse-click on ...' means moving the mouse cursor to a certain area and then pressing the mouse button briefly. Sometimes instead of saying 'mouse-click', we simply say 'Click'.
- Double click: Two mouse clicks, with a very short pause, as when starting a program from the desktop.
- Drawing: Marking an area with the mouse. Specifically: place the mouse cursor at the beginning of the area, press the left mouse button and hold it until the mouse cursor points to the end of the area.
- Button: A text in a frame to click. For example, the **OK** fields in the Dialog. But a button also means other elements in the Dialog that can be clicked on (see below).
- Radiobutton: In some Dialog boxes of Interface there are (round) buttons, of which - in a group - only one can be selected i.e. the selection of a "button" turns off the others. The buttons behave like the buttons of old radios - hence the name. In Interface, such buttons can also be selected by clicking on the associated (adjacent) text.
- Checkboxes: Are buttons that can be selected or not. If you select them, the display will change to a crossed-out rectangle. Checkboxes can also be selected by clicking on the associated text.
- Icon: Graphic symbol, e.g. a floppy disk, a trash can or similar symbols shows.
- Select: Select, or click on an object, usually a field in a Dialog.
- FlyDial: "Flying Dialog" have been the state of the art of modern programs for ATARI for some time now. A Flying Dialog is marked by a "dog's ear" in the upper right corner of the window. If you click on this corner, the Dialog can be moved to the desktop.
- MyDial: The MyDials are a collection of functions included with Interface. With them you can easily install flying Dialog, keyboard controls in Dialogs, Popups or color icons (and much more) in your C and Basic programs. In this manual the MyDials objects are discussed in more details in a separate chapter that also describes an application.

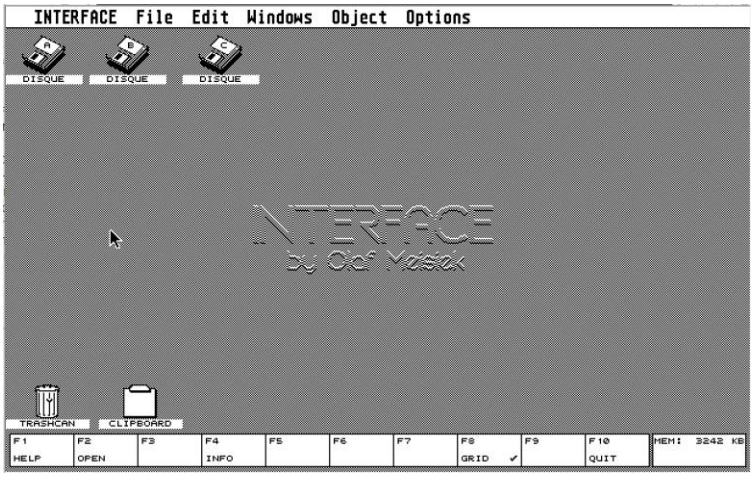
In this manual we sometimes used some simplified expressions. For example, if we want to tell you that the 'Control' key should be pressed together with the 'O' key, we simply write 'Control O'. If we describe an option from a Menu, the text does not say "**Open…**" from the "**File**" Menu ..., but the option **File: Open...**. Buttons and similar elements are also displayed in a **highlighted font**, so that you can already orientate yourself on the design.

We sometimes use Menu, Alert, or Dialog instead of Menu bar, Alert box, Dialog box.

## 2.4 Icons on the Desktop

Interface has its own Desktop, which is displayed immediately after startup and provides the main management elements in graphical form. The desktop color itself can be defined by changing the color palette in the tree "DESKCOL" in the resource.

The Desktop starts below a Menu bar. It contains icons for the drives, a recycle bin, a clipboard, as well as icons for loaded resource files, and a function key toolbar. The icons on the desktop can be moved as desired.



The only exception is the function key block, which shows you which function key triggers which option. It is always displayed at the bottom of the screen (cannot be moved):

| Button | Name | Menu Option |
|---|---|---|
| F1 | HELP | File: Help... |
| F2 | OPEN | File: Open File… |
| F3 | CLOSE | File: Close |
| F4 | INFO | File: Info... |
| F5 | SAVE | File; Save |
| F6 | FIND | Edit: Find... |
| F7 | FLAGS | Edit: Set Flags... |
| F8 | GRID | Options: Use grid |
| F9 | TEST | Edit: Test Object Tree |
| F10 | QUIT | File: Quit |

If you double-click on a drive symbol, the associated window opens, in which all folders and resource files that are on the drive are displayed.

The Recycle Bin is used to delete files and objects. To delete something, it just has to be dragged onto the wastebasket with the mouse and drop it there.

The clipboard is used for objects from Dialog boxes and Menu bars. You simply drag objects onto the clipboard, or drag them from the clipboard to a Dialog box or Menu bar. If the clipboard icon is to be moved, you have to select it first (mouse click or rubber box) and then move it around. As long as it is not selected, you can “drag” its content. It goes without saying that you can copy several objects onto the clipboard at the same time. If you hold down the Shift key while releasing the mouse button (mouse cursor over the clipboard), the objects will be copied to the clipboard. In another case, they are moved to the clipboard. The object(s) remains in the clipboard until a new object is copied to it, i.e. you can drag multiple copies of the same object from the clipboard into one or more Dialog box. The clipboard described here is only internal to Interface, i.e. it has nothing to do with any other existing clipboard folder. When performing the operations describe above for text or icon the standard GEM clipboard is used. All functions of the clipboard can be performed with the Menu options: **Edit: Cut**, **Edit: Copy** and **Edit: Paste**.

Loaded resource files are represented by an icon above the function key block. By double-clicking on such an icon, a window with the object tree of the corresponding file can be opened or, if it is already open, brought into the foreground. In addition, resource file icons can be dragged onto a drive or the recycle bin to respectively save or erase the associated resource.

A tip for those who want to switch off the interface desktop: place the desktop in the window (see reference **Options: Preferences...**), save the parameters, reload the interface, close the desktop window, save the parameters again. For all other starts then no desktop appears. You can reopen the desktop by clicking through all the copyright messages (extravagant, but what to do ...) or by not putting the desktop in the window. However, it is better to shrink the desktop as a window so that only the trash, the clipboard and one or two resource files are visible. If the desktop is in a window, you can also select and move objects under MultiTOS if the desktop is in the background. This can be done without MultiTOS. The objects have to be clicked only with the left and right mouse button.

## 2.5 Opening files

Resource files can be loaded by typing ‘Control O’ via the keyboard, by clicking the Menu item **File: Open**, by clicking on the **OPEN** button of the function bar, or by tapping the F2 key.

Drag and drop from the operating system (for example MultiTOS) is supported. Resource files that are dragged to any Interface window are opened. IMG files and WINDOWS or OS/2 icon files can be dragged onto an icon editor window.

## 2.6 The Menu

The Interface’s Menu does not differ from the menu of other programs. Except for one special feature: at the end of each line are usually two to three characters. This is done because (almost) all options which are listed in

the Menu entries can also be called up via the keyboard, and the characters behind the option indicate which key combination is to be pressed for this purpose.

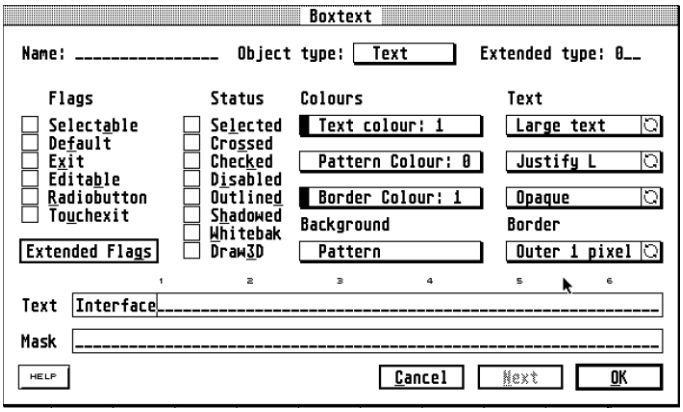
Since a “normal” keystroke leads to an entry in a text, additional keys must be pressed to avoid misunderstandings when selecting a Menu option.

The following characters in the Menu and their meaning:

| | |
|---|---|
| ^ | for the button Control |
| ~ | for the Alternate button |
| ⇧ | for one of the shift keys |

For example, to open a resource: press the ‘Control’ and ‘O’ keys simultaneously.

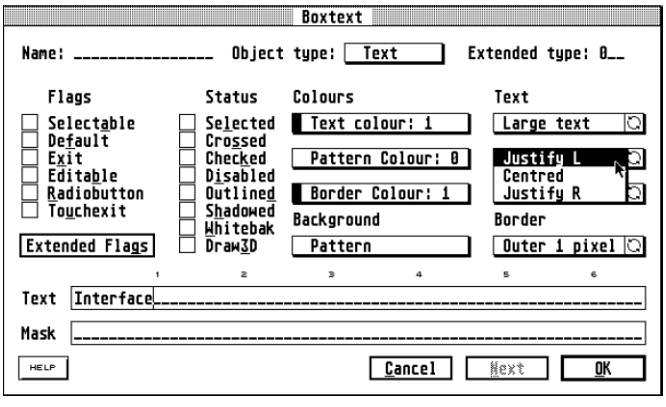
## 2.7 Dialog Boxes




The Dialog boxes of Interface are basically no different from those of other programs or those from the desktop. But you can do a lot of very practical things with Interface's Dialog boxes that are not possible otherwise. For example, the Dialog box shown here contains several additional functions.

First, and most conspicuously, is the “mover” above the Dialog. Interface displays its Dialogs optionally in windows, so that you can move the Dialog and, above all, work without restrictions in multitasking. You can deactivate this with the **Dialog** option in **Options: Preferences...** (see Dialogs (in window)). In this case you will find a 'donkey ear' in the upper right corner of the Dialog, which serves the same purpose. You can click on this corner and move the Dialog while holding down the mouse button. There is another special feature: If you press a Shift key at the same time, the Dialog disappears and the background becomes visible. An extremely practical thing, if one of the short-term memory fails and the necessary information is just hidden from the Dialog to be edited.

In all cases, Interface remembers the position of the Dialog boxes or windows and opens them at the last selected position until the end of the program. If you take a closer look, you will notice that some letters are highlighted in a Dialog. This is to make it clear that pressing the key corresponding to the underlined letter together with the *Alternate* key would have the same effect as if this option had been clicked on. As a result, the interface can be almost completely controlled via the keyboard, for example buttons such as **Cancel** can basically be triggered with Undo key and the **Help** function with Help key. Somewhat more remarkable are the round “buttons” that do not exist in normal Dialog boxes. These are the Radiobuttons already mentioned above. Although they look elegant, they are not very large and therefore not very easy to select, so we made sure that you can also click on the text next to the button and get the same effect as if you clicked on the button itself. And another special button can be found in the Dialogs of Interface: Checkboxes. These are buttons that you can select or not. If you select them, the display changes to a crossed rectangle, just like the “Please tick” fields in a Dialog.




The Dialog box for Boxtext parameters shows (for example) another element, so-called 'Popup-Menu'. They appear in a Dialog as buttons with a shadow (i.e. Text alignment), clicking on the button displays a selection of possible values, and you can - as for a Menu options - select one just by clicking it. If you want to cancel the selection, just click next to the Menu. Popup-Menus can also be operated via the keyboard, as well as using the mouse. The bar, that marks an entry, can be moved through the popup with the vertical cursor keys,

with Control Home (move to beginning), and Shift Control Home (move to the end). Return selects an entry as if it was clicked with the mouse, Undo or Esc cancels the popup as if by a mouse click next to the Menu.

Some Popup-Menus have an additional control element, namely a box on the right edge of the popup that shows a *circular arrow*. When you click in this field, the next option in the Popup-Menu is selected (top to bottom). This allow you to use less mouse click(s) for Popups-Menu with few entries.

### Extended editing functions

GEM provides the user with various editing options in Dialogs, such as: setting the cursor with the mouse on a specific input field. Interface offers in addition to the well-known GEM functions even more:

| | |
|---|---|
| Shift + Cursor left | Move the cursor to the beginning of the line. |
| Shift + cursor right | Move the cursor to the end of the line. |
| Control + Cursor left | Move cursor one word to the left. |
| Control + Cursor right | Move cursor right one word. |
| Insert | If any characters are allowed under the cursor in the field, a Dialog appears with all the special characters that cannot generally be entered via the keyboard. Select the desired character with the mouse. |
| Control C | Copy text from the current edit field to the GEM clipboard. |
| Control X | Copy text from the current edit field to the GEM clipboard and then delete the edit field. |
| Control V | Insert text from the GEM clipboard. |
| Click on | The cursor is placed in front of or behind the selected letter. |

## 2.8 The Mouse

Most of the mouse functions in interface, such as click and drag, work the same way as in the GEM desktop or in other programs that stick to the standard of the desktop. Nevertheless, all important mouse functions are presented here once again. 'click' means a short press of the left mouse button, while 'drag' describes moving the mouse while holding down the left mouse button.

| **Function** | **Effect** |
|---|---|
| Click | Selects an object under the mouse. A selected object is displayed inverted by Interface. |
| Shift + click | Selects additional objects if one or more is already selected. Shift + click on a selected object cancels the selection again. |
| Control + click | Selects the parent of an object. |
| Double-click | Opens a file, a tree or an object for editing. |
| Drag | Brings an object from the parts box into a window or moves all selected objects, such as files, trees or objects. |
| Shift + drag | Copies the selected trees or objects. |
| Control + drag | Copies the parent of an object. |
| Shift + Control + drag | Copies the parent of an object with all children. |
| Alternate + drag | Draws a rectangle in a windows. All objects in the rectangle are selected after the operation. |

Any combinations of Shift, Control and Alternate are allowed. In addition, the described functions can also be used in inactive, i.e. background windows can be applied while holding down the right mouse button at the same time.

## 2.9 Window

The content of Interface windows can be moved not only by clicking the sliders with the mouse, but also with the cursor keys. If a shift key is pressed at the same time, the entire window contents can be scrolled through. With Control Home and Shift Control Home you get to the top or bottom of the window content.

## 2.10 Interface limitations

Interface is not able to work with arbitrarily large or any number of resources or objects. The limits are very generous. The maximum number are:

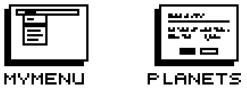
| | |
|---|---|
| Resource files | 15 |
| Trees of a resource | 32767 |
| Objects in a tree | 1024 |
| Images per resource | 10240 |
| Object names per resource | 32767 * 1024 |

The maximum number of structures (OBJECT, TEDINFO, ICONBLK, BITBLK and USERBLK) of a resource is 5120 (per structure).

The maximum length of all strings must not exceed 160 KB.

The length of a resource file itself is defined as unsigned long, i.e. theoretically it could be four gigabytes in size. In practice, it is limited by the maximum number or length of the structures just mentioned. How to work in your programs with resource files larger than 64 Kbytes is explained in chapter 8.

## 2.11 A Simple Exercise




In this chapter, you will become familiar with the basic functions of Interface.

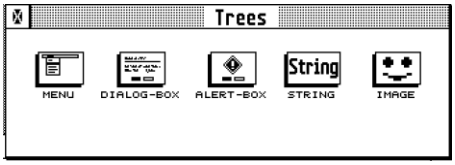
Based on an illustrated example, you will get to know the Interface's functions for creating and editing a resource, that contains a Menu bar and a Dialog box.

### 2.11.1 Creating a new resource

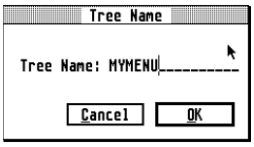
Start Interface and select the entry **File: New** from the **File** menu. A new icon with the name 'UNTITLED' (to indicate that you can only save it with **File: Save as ...**) appears on the desktop. In addition, an empty window opens with the same name, in which later the Object trees are displayed.

### 2.11.2 Building a new Menu

From the **Edit** Pulldown Menu, select **Edit: New Trees...** A box or a window (can be adjusted with object boxes (in window) ) appears on the screen, in which all available tree types are displayed.




From there drag the tree 'MENU' to the window that belongs to the icon 'UNTITLED'. A Dialog will appear asking you to enter a name for the new Menu.



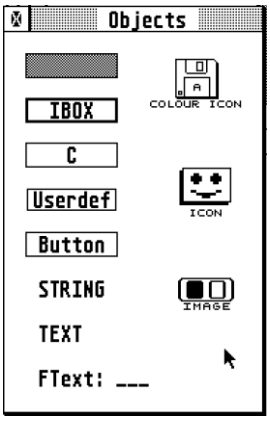

Name the new Menu tree 'MYMENU' and end the entry by clicking on the **OK** button on the window or by pressing the Return key.

#### 2.11.2.1 Opening the Menu tree

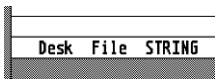
The window of the newly created resource file now shows a Menu icon with the name 'MYMENU' '. To edit this Menu, double-click on the icon. This will bring up another window titled 'MYMENU'. In this window, you will see a Menu bar that already contains two entries 'Desk' and 'File'.

#### 2.11.2.2 Adding new Title to a Menu

To add another Menu item, select **Edit: New Objects....** A box will appear on the screen that contains the different types of objects.




Drag the 'STRING' object to the right of the 'File' entry in the Menu window. If you have done everything right, now the new Menu item 'STRING' should be visible next to the Menu item 'File'.

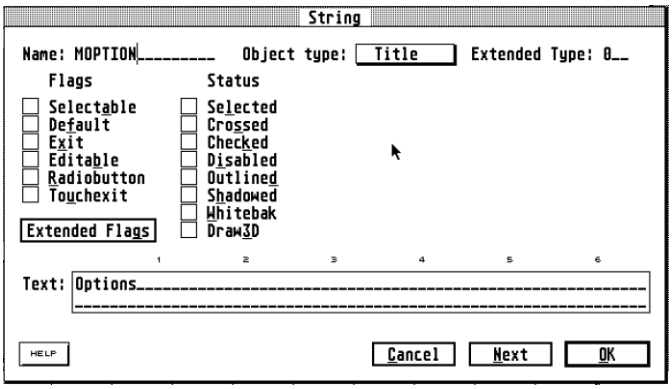



Double click on the entry 'STRING'. A Dialog appears in which all data for the new object can be changed.

Use the vertical cursor keys or the Tab key to move the cursor to the ‘Text’ input line of this box. In this line the text 'STRING' should be displayed. Delete this text by pressing the Esc key and instead enter the text 'Options'.

Finally, move the cursor to the upper left input field ‘Name’ and enter the name 'MOPTIONS' there.

Further settings for the Menu title are not necessary as they are made automatically by Interface. To exit the Dialog, click on the **OK** button or press the Return key.




To make new entries in the Options Menu, you must first click on the '**Options**' entry (in the ‘MYMENU’ window). This inverts the title and a small white rectangle, the Menu box, appears below it. Call the object box again and drag the object 'STRING' into this Menu box (further processing will be discussed later). You do not have to worry about the size of the box, it will be automatically adjusted by Interface.


You should also insert new entries in the **'File'** Menu. If you click on this Menu, you will notice that there is already an entry there, namely **Quit ^Q**. Copy it three times by holding the Shift key and moving it with the mouse. As a next step, each new entry is changed into a meaningful name.

Double-click on the first entry in the **'File'** Menu and change the text from **Quit ^Q** to **Open... ^O**. The three dots behind the text indicate, to a user of the program, that after choosing this option, a Dialog (here, for example, the file selector) will appear to ask the user for further information. Since in a “good” GEM program the most important menu functions should also be available via the keyboard, behind the 'Open ...' is the name of the equivalent keyboard command ‘^O’. The '^' stands for the Control key.

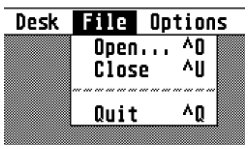
According to the GEM programming guidelines, there must be two spaces in front of a Menu entry (to allow check marks to appear in front of an entry), followed by one space. But you do not have to worry about it yourself, Interface inserts these spaces automatically for you. Now enter the name 'MOPEN' in the field ‘Name’ and then end the Dialog with the **OK** Button or by Return.

Note: The Keyboard shortcuts in Menu items have the following abbreviations:

| | | |
|---|---|---|
| Control | ^ | This character can be found directly on the keyboard. |
| Alternate | ⎇ | The character that is in the upper right corner of most windows. To be able to enter it, you must press the Insert key and select the 7th character in the top left corner of the ASCII table with the mouse. |
| Shift | ⇧ | Also to be chosen from the ASCII table. |

Execute the same step to change the second entry in **Close ^U** with the name 'MCLOSE'.

To separate the last entry better from the first two, change the ‘Text’ of the third entry to '-'. Before you leave the Dialog, click on the field ‘Disabled’ with the mouse. A cross should be seen in the box next to this field. This setting makes the new text highlighted and not selectable. In addition, Interface automatically adds as many dashes as necessary to create a continuous hyphen. If you have done everything right, the Menu should now look like this:




In the manner described above also change the new entry 'STRING' under the **Options** Menu to **Settings... C**.

If the keyboard shortcuts '^O', '^U' and '^Q' are not exactly one below the other in the 'File' Menu, simply add a few spaces at the appropriate places. This can be automated by Interface, if in **Options: Parameters...** you select the option **Align shortcuts in Menu entries**.

### 2.11.2.3 Testing the Menu

Select **Edit: Test object tree** or press the function key F9 to test the new Menu.

Interface opens a new window, that contains at its top the Menu bar that you want to test. If you click on the Menu bar at the top of the screen, the Menu responds to mouse movements as if it was a “real” Menu. Now click the entry **Options: Settings... C**. A small Alert box should appear, showing you the name and object’s number of the selected Menu item. If you want to test further, select **OK** in this Dialog.

There are two ways to end the test: Select a Menu item and then click **Cancel** in the Alert box (see above) or close the test window in which the Menu is displayed.

Note: the keyboard shortcuts '^O', '^U' or 'C' entered in the test window works, because Interface reads these key combinations directly from the Menu and interprets them. If you use this Menu later in your own program, you will need to write a routine to intercept and evaluate those keystrokes. A complete example in C can be found in the book 'From beginner to GEM pro' by Dieter and Jürgen Geiss. MyDials provide a similar routine that is used in Interface.

Our sample Menu is now finished. To close the Menu tree window, you must click on the top left corner of the associated window.

## 2.11.3 Building a new Dialog-box

Select **Edit: New Trees...** again. A Tree window appears on the screen, in which all the available tree types are displayed. Drag the 'DIALOG-BOX' icon to the window that already contains the 'MYMENU' icon. Give the name 'PLANET' to this new tree.

### 2.11.3.1 Opening the Dialog tree

Now open the 'PLANET' tree by double-clicking on the corresponding icon. It opens a window with the title 'PLANET', in which an empty box with a frame can be seen. This box is the root object of the Dialog tree.

### 2.11.3.2 Adding new Objects to a Dialog

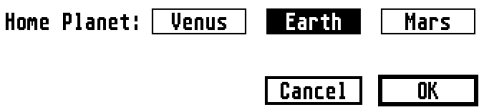
Call up the Objects box: you can do this with the mouse via the **Edit** menu or with the key O (exceptionally only one key – not a combination). Whenever you pull an object out of it, the object box disappears. If necessary, call it up again with O. As an option the tree and object boxes can also be displayed as windows, so they are always available. You can set this in the **Option: Parameters...** menu (see object boxes (in window)).

Drag a STRING object in the top left corner of the dialog window and an IBOX object to its right.

Increase the width of the IBOX frame so that it almost reaches the right edge of the window (you can enlarge the IBOX objects by dragging the mouse at the bottom right corner of the frame - more information to come). Then drag a Button object from the object box into the left part of the IBOX frame. Open the STRING object with a double click and change the text 'STRING' to 'Home planet:'.

Since the object does not need to get a name, click on the **OK** button immediately afterwards.

If the new text ('Home planets:') of the STRING object should overlap the Button object, simply drag the IBOX object containing the button further to the right. If that does not work, because the object already reaches the edge to the right, then you have to reduce it a bit beforehand (drag the lower right corner to the left). For your convenience we show you how the finished Dialog should look like.




Open the Button object by double-clicking and replace the text 'Button' with the text 'Venus', then enter the text 'PLVENUS' as the object name in the 'Name' field. In the Button Dialog select (i.e. with a cross) only the 'Selectable and' Radiobutton checkboxes.

- 'Selectable' means that the 'Venus' button can be selected (it is inverted when selected).
- A 'Radiobutton' has - like the buttons of an old radio - the following properties: Within a parent object (here: the IBOX frame) only one radio button can be selected at a time. If you click on an unselected button, the previously selected button will be deselected automatically. There **must** always be a selected button in a group of Radiobutton.

Confirm the setting with the **OK** button of the Dialog.

There are two more Radiobuttons in this Dialog. To do this, first copy the 'Venus' button by dragging it with the mouse while holding down the Shift key to the right of the first button. Then repeat the operation for the third button. Important: All the buttons must be located **inside** the IBOX object. If there is not enough space, please enlarge this box. More about IBOX topic will follow soon. Change the text of the second button to 'Earth' and give it the name 'PLEARTH'. The third button is changed to 'Mars' with the name 'PLMARS'. Since in a group of radio buttons one should **always** be selected, now open the button 'Earth' with a double-click. Then click on the status 'Selected' and exit the Dialog with **OK**. Now the button 'Earth' is selected and will be displayed inverted accordingly.

### 2.11.3.3 Enlarging or reducing the size of objects

All objects within a Dialog box/window (a window in which a Dialog tree is edited) can be resized.

To change the size of an object, you can drag the mouse at the bottom right corner of the object until the desired size is reached. This generally invisible “drag corner” has the size of a letter (the corner can be made visible – see Show Sizing-box ). If the object gets small, then the size of the “drag corner” is set to half the height or width of the object.

Objects can be resized in the four directions. You can still “touch” the object only in the lower right corner with the mouse to resize it, but you can then drag the mouse to the left or over the object. Interface then automatically switches to the corner closest to the mouse.

If an object completely encloses other objects after enlarging, Interface asks if these objects should be adopted as children. If this is answered in the affirmative, the enlarged object becomes the parent of the covered objects.

### 2.11.3.4 The IBOX and its special meaning

First of all, an explanation of why all radio buttons must be in this box: If you use more than one group of radio buttons in a Dialog, these groups must be in different parent objects (boxes). Otherwise, all radio buttons would be considered as a group, and there would be no way to determine membership in a particular group.

Since there is only one group in our example Dialog, you could save yourself the IBOX object. But if you want to add another group of Radiobuttons to this Dialog, they have to be in different parent objects. Therefore, it makes sense to group radio button into an IBOX parent objects from the beginning. You should minimize the size of the IBOX object as much as possible.

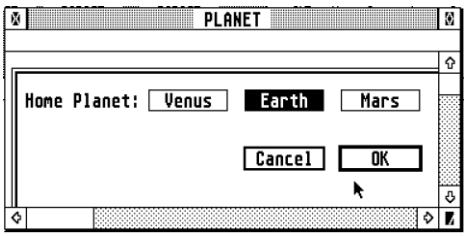
Since the IBOX object serves only to group the radio buttons it should not be visible in the finished Dialog. Open it with a double click. In the Dialog that appears, you will see a box with the caption “Border” at the bottom right. Click the Outside 1-pixel box and choose ‘No border’ from the Popup Menu that appears. If you close the settings Dialog with **OK**, the IBOX should no longer be visible. However, it can still be selected and moved.

> *Note: 'Exit' buttons are important objects with which a Dialog can be exited. Most Dialogs have at least two exit buttons: 'OK' and 'Cancel'.*

Open the object box again with the O key and drag the Button object out to the lower right corner of the new Dialog. Then copy this new button with the mouse (click and press Shift-key) to the left of itself.

Now open the right Button, change its text to 'OK' and name it 'PLOK'. Make sure that the flags ‘Selectable’, ‘Exit’ and ‘Default’ are selected for this button.

- 'Exit' causes the AES function `form_do()` to exit when clicking on such a button.
- 'Default' means that the function of this button can also be triggered by using the Return key, which is indicated by a thicker border of the button. Logically, this flag can only be assigned to **one** button per Dialog.




Then open the left button, change the text to 'Cancel' and name it 'PLCANCEL' Turn on the ‘Selectable’ and ‘Exit’ attributes.

The finished Dialog should look something like the picture here.

### 2.11.3.5 Testing the Dialog tree

The Dialog is finished. Choose **Edit: Test Object Tree** or press F9 to test it. The test Dialog window is drawn in the middle of the screen.

Click on the buttons **Venus**, **Earth** and **Mars** one after the other. If you have given the 'Radiobutton' attribute to all buttons, only the last clicked button will be selected, all others will be deselected immediately.

Then press the Return key. Since the **OK** button has the attribute 'Default', it will be selected. Since he also has the attribute 'Exit', GEM terminates the Dialog and Interface shows in a small Alert box which indicates which button was used to exit the Dialog. In this Alert box you can now choose whether you want to further test the Dialog or if you want to stop the test.

Click the window of the new Dialog in the top left corner to close it and finish editing the Dialog.

## 2.11.4 Deleting files, trees or objects

To delete files, trees or objects, select them and drag the selected items to the trashcan. When the trashcan is inverted (i.e., it is just under the mouse), release the mouse button.

Deleting an object from a Dialog window can be undone by pressing the 'Undo' button immediately afterwards. But this works only with simple objects, not with whole trees or files - so be careful when deleting!

### 2.11.5 Saving the resource file.

To save the newly created resource as a file, select **File: Save as….** The file selector will appear.

Press the Esc key to clear the name input field where the cursor is located and enter the name 'DEMO'. After pressing the Return key, Interface saves the following files:

- DEMO.RSC: This is the actual resource file. It stores all data except the object names.
- DEMO.HRD: This file stores the names of all objects. Interface needs this file to map the names to the correct objects. In addition, parameters associated with the resource in this file, e.g. the output material is stored.
- DEMO.H: This file also contains all names and their associated object numbers, but as simple ASCII text. It can be taken over by a C program with the command '#include'.

Interface can provide data for the programming languages C, Pascal, Modula, Basic (e.g., Omikron) and GFA Basic output. It is also possible to adapt the output to another programming language (e.g., Assembler or Fortran). To instruct Interface, to select a specific format when saving, select the resource file icon on the desktop and select the option **File: Info...** (see Info of a selected resource file).

Header files are only saved if labels have been changed or deleted, the object hierarchy has been changed, or if the Resource Info Dialog has been left with OK. This is likely to please all compiler users whose compilers would otherwise recompile the entire project.

## 2.12 Editing a resource in detail

In the following we will go into detail about editing a resource. The demo resource, which was described in the previous part of the exercise, serves as reference. If you did not understand the exercise, you will find a resource named DEMO.RSC on one of the interface disks, which has the same structure.

### 2.12.1 Editing a Menu

A GEM program generally has a Menu bar that provides the user with the most important options. In our example it is the tree named 'MYMENU'.

#### 2.12.1.1 The Menu Title

To expand a Menu bar with a new Title, first open the corresponding tree by double-clicking it. Then open the object window with **Edit: Objects...** and drag a STRING object to the desired position in the Menu bar.

Only STRING objects can be dragged into the Menu window. If you want to have icons or similar in the Menu bar (which do not really belong there!), you have to close the window, select the symbol of the Menu tree and convert it to a Dialog tree with **File: Info...**. After that you are responsible for the complete design of this tree, i.e. Interface can no longer ensure or monitor a correct format in it.

If you want to change the text of a Menu Title, double-click on the corresponding title. The Dialog for the string editing appears in which you can change or enter the name of the object and the text. Spaces in front of and behind the text are automatically set by Interface when **Optimize spaces in Menu entries** is switched on in the **Options: Preferences...** menu (see Optimize spaces in Menu entries), so you do not have to worry about this. The flags and statuses that can be set in this Dialog are irrelevant for Menu items.

To change the order of the Menu Titles, click on a title and move it roughly to its new position, e.g. on the title, which he should appear after. A title can only be moved within its Menu bar or to the recycle bin. The associated Pulldown Menu is also moved along. Moving a Title to the Recycle Bin also clears the Pulldown Menu.

When copying a title by moving while holding down the Shift key or using the Edit Menu options, the corresponding Pulldown Menu is not copied (!) because it belongs to a different object hierarchy. Titles can also be copied to Dialog trees or the Pulldown Menu of another Menu tree.

#### 2.12.1.2 The Menu Items

A Pulldown Menu opens when you click on the associated title with the mouse.

To create a new Menu Item (entry), open the object box and drag a STRING object to the desired position in the Pulldown Menu (only STRING objects can be dragged into a Menu Pulldown window). To change the text of a Menu entry, double-click on the entry to be changed. This open a Dialog in which you can change or enter the name of the entry and its text.

Spaces in front of and behind the text are set independently by Interface when **Optimize spaces in Menu entries** entry in **Options: Preferences...** menu is switched on (see Optimize spaces in Menu entries ), so you do not have to worry about them anymore. Always two spaces are inserted before the entry. In addition, if the option **Align shortcuts in Menu entries** (in the same Dialog) is selected, any shortcuts that have been entered will be automatically aligned right-justified. Interface considers the following rules for shortcut detection:

- The shortcut starts behind the first space found from the right.
- At least one control character (Shift, Control or Alternate) must be in front of the key specification, or the key information must follow immediately; but only special keys like Help, Insert etc. are allowed.
- After the control character(s) must follow a 1-character key, or an indication of a special key.

If these conditions are met, the shortcuts are arranged according to the following rules:

- There must be at least two spaces before each shortcut.
- If there is a '...' at the end of the text (for example, in 'Open...'), then one Space in front of the shortcut is sufficient in this line.

The allowed characters for the control keys are in the interface resource in the free string 'SHORTCUT', which you can change and save as amended. The allowed special keys are in the free string 'SH_KEYWORDS'. There must be a '/' in front of and behind each special key in this free string.

As a result, almost all conceivable shortcuts of Interface are recognizable. *The only exception is a single letter with no control key*[1]. We deliberately excluded this possibility so that Interface would not mistakenly arrange incorrectly a character from a Menu item without a shortcut (e.g., “Image 1”).

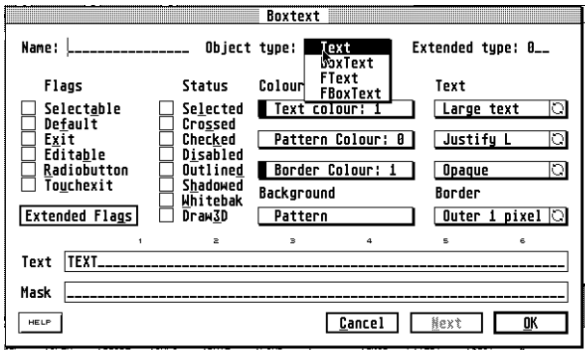
An entry can only be moved within the Pulldown Menu or to the recycle bin, but can also be copied to Dialog trees, the Menu bar or Pulldown boxes of another Menu tree.

## 2.12.2 Editing a Dialog

Dialogs represent the most common form of communication between user and program. In our example, there is only one Dialog tree named 'PLANET'. To change it, double-click on the corresponding icon.

### 2.12.2.1 Overview

To add a new object, open the object box and drag an object of the required type into the Dialog window. If an object is to be modified (for example, after a double-click on it), a Dialog appears in which the data of the object can not only be displayed, but also edited. All Dialogs are similar in structure; differences arise only in that some objects cannot have all the attributes. For example, a button does not have color nor pattern, moreover, the edge strength of its frame cannot be set (because it is not intended by GEM).




In the following all options are discussed. If one of the enumerated options does not exist for a particular object, you cannot give this attribute to this kind of selected object.

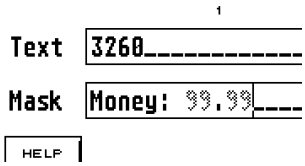
- Name: This field contains the name of the object; under which it can later be addressed by a programming language. All object types can be given a name.
- Object Type: This entry contains a popup Menu with which the object type can be changed later. However, a type can only be changed to a similar type (for example, change a STRING to BUTTON or TITLE), so that no data is lost during the conversion. This entry is available in all Dialog boxes. If you want to convert an object into a different object type (for example a STRING into TEXT), you can also exit the Dialog and click on the object, to display the ‘Object Popup Menu’. If a type conversion is possible, the name of the other object type appears in the last line of the popup. By selecting this entry, the object is converted. But eventually some data of the object will be lost.
- Extended Type: GEM allows the programmer to use their own object types. To be able to use different self-defined object types, they are numbered consecutively. A separate object type can have a number between 1 and 255. This number is entered in the OBJECT structure in the high byte of `ob_type`. Interface has an interface that allows the user to include their own routines for drawing such objects during the test

[1] It possible to add a shortcut without a control character: Interface detects a single letter shortcut if the letter is preceded by ‘… ’ (e.g. ‘Setting… C’), or if the letter is aligned with other shortcuts in the Menu (DrCoolZic).

function. How to define and use your own object types can be found, for example, in the book 'Vom Anfänger zum GEM Profi'. Including your own routines to represent extended object types in Interface is explained later in this guide.

- Flags: The flags describe the properties of an object. Since an object can have several properties - in some cases - the flags can be selected by checkboxes. We will talk about the different flags in detail.
- Status: The status describes the representation or the state of an object. Since several properties can also be combined here, the status must also be defined by check-boxes (more on that later).
- Extended flags: GEM uses only part of the possible bit combinations for flags and status. With the Extended Flags button you can open a Dialog in which you can set (for your own object types) the unused bits.


- Colors and background (pattern): The colors of individual object elements as well as the filling pattern of the background can be set in four Popup Menus. The color can be adjusted separately for text, frame and fill pattern. If interface was started in a color resolution, the available colors are displayed in the popups. Otherwise, the color can be read from the texts of the Menu, which are based on the GEM default setting.
- Text: Objects that contain text can be given the attributes: Large/Small, Left/Center/Right, and Opaque/Transparent using Popup Menus. To the right of the current entries is a small circle called a Circlebutton. If you click on it, the next popup entry will be selected.
- Border: This popup allows you to set the thickness of the frame of an object and whether the frame should be displayed inside or outside the object.
- Text input: The text of an object can be entered using the editing options described in the introduction. If the input field should be longer than one line, both lines are merged into a single, longer line after the input of Interface.
- Mask: The option of specifying an input mask exists only for TEXT objects and is only useful there for the object types **FTEXT** and **FBOXTEXT**. When these types of objects have their 'Editable' flag set, keystrokes can be entered in such objects, with the mask defining which characters are accepted upon input. In addition, the mask may contain text parts used for formatting the input. For example, let's assume we want to create an input field in which only numbers are allowed, that starts with the text 'Money:', and that displays a default value of '32 .60'. For that the text and mask fields must have the following contents:




In the Text field: '3260' and
in the Mask field: 'Money: **99.99'**.

To enter the bright nines, you must hold down the Alternate key while entering the numbers. After exiting the Dialog, the object should be displayed as 'Money: 32.60'. In the text field you enter the text that will be used later in the placeholder (in the example above, the bright nines). You must enter a text that is at least as long as all wildcards used. If you do not want this text to be displayed, you must first enter a '@' character. GEM will not display all subsequent letters and place the cursor over the first character. The text must be entered because it will later be replaced by the user's input. If the text is forgotten, GEM simply writes all the characters entered by the user to other memory locations and can thus possibly crash a program. Although Interface tries to anticipate this by automatically entering null bytes, this does not work, for example, if you process the resource file with another resource editor. All characters entered normally in the mask field will later be displayed normally. In contrast, the characters entered with Alternate in the mask field are displayed brightly and determine the permissible input data. They will later be replaced by the contents of the text box. The following characters are allowed as placeholders:

Money: 32.60

| | |
|---|---|
| '9' | only digits are allowed (can cause a crash in TOS 1.0 due to an error in the operating system when entering an underscore '_'). |
| 'A' | only capital letters and spaces |
| 'a' | only letters and spaces |
| 'N' | capital letters, numbers and spaces |
| 'n' | letters, numbers and spaces |
| 'F' | All characters which may belong to a filename and additionally ':? *' |
| 'f' | All characters that belong to a filename |

'P' Any characters that can be associated with a pathname

'p' Like 'P', but without '?' and '*'

'X' All characters are allowed

'x' Like 'X'. however, the characters are converted to uppercase letters

All wildcards must be entered with the Alternate key pressed, as mentioned above. In the box text Dialog, the characters 'A' - 'Z' and 'a' - 'z' are allowed for `te_pvalid` (the inputs for the mask) so that you can use new input types for your own custom objects. Interface, or MyDials supports only the wildcards listed above.

In each Dialog box you can go directly from one object to the next with the next button. In the process, all the data in the Dialog box is transferred before the data of the next object is displayed there.

The **HELP** button displays a help page explaining all extended object types of the MyDials EXTOBFIX file. The BoxText Dialog also displays a help page for all `te_pvalid` placeholders in the TEDINFO structure.

### 2.12.2.2 Object types and the object box

GEM knows a total of fourteen different object types with the new color icons of TOS 4.01, which you will find in the object box of Interface.

| Object Type | Meaning |
|---|---|
| G_BOX | Rectangular box for displaying rastered or colored areas, e.g. in the slider of a list box. |
| G_TEXT | Formatted text in two sizes (in the small and in the “normal” system font) and three orientations (left-aligned, centered and right-aligned). Alignment is possible because the object size may be larger than the area occupied by the text. |
| G_BOXTEXT | Corresponds to G_TEXT with the difference that a box is drawn around the text. |
| G_IMAGE | A monochrome graphic without a mask. |
| G_USERDEF | This object type allows you to define your own functions for output. The MyDials use them for several types of objects (Checkbox, Radiobutton, Dialog title, color icons, …). |
| G_IBOX | Invisible rectangle with no inner surface, so that it is only visible if the frame has a nonzero strength. It is used to combine several objects. |
| G_BUTTON | Centered text in a rectangle, its thickness depends from the flags DEFAULT, EXIT and TOUCHEXIT. |
| G_BOXCHAR | A rectangle that contains a single letter. In the Dialog for object definition (see above), the input field Characters appears: at the position where the text options are otherwise. |
| G_STRING | A string without the attributes of the G_TEXT and G_BOXTEXT objects. |
| G_FTEXT | Formatted text in the “normal” system font that can be edited by the user. |
| G_FBOXTEXT | Corresponds to G_FTEXT with the difference that a box is drawn around the text. |
| G_ICON | A monochrome graphic with mask. |
| G_TITLE | The title of a drop-down Menu. |
| G_CICON | A color icon. |

The object type G_USERDEF is supported directly by Interface. To do this, Interface creates an OBJECT structure whose `ob_spec` pointer points to a USERBLK structure. For each object, a separate USERBLK structure is created in the resource.

`ub_parm` can be entered directly in Interface. `ub_code` contains a NULL pointer, i.e. after loading such a resource, you must first enter a pointer to your own character routine in `ub_code` of all USERDEF objects. Otherwise the AES would crash immediately when drawing such an object.

The support of self-defined objects is further enhanced by the possibility to include your own objects in the object box of Interface (FlyDial-corner, OK-button, ...). To do this, load the interface resource and copy the desired object to the 'OBJECT' tree. The sort order does not matter; it just depends on the new object ends up in the inner box in which the other objects are located. The new object must become a direct child of the inner box and must not contain any children.

It will not be displayed as a G_USERDEF object after the start of Interface so that data is not inadvertently changed by the fitting routine (e.g. object size, flags, ...).

If the object box is in a window, you can pull out objects even if the box is in the background. To do this, you must click on the object with the left and right mouse buttons at the same time. MultiTOS eliminates the simultaneous pressing of the right mouse button.

#### 2.12.2.3 Object flags

The flags of an object determine its properties. An object can have several properties, e.g. be selectable (SELECTABLE) and end the Dialog with the selection (EXIT). They are summarized here:

| Object type | Meaning |
|---|---|
| Selectable | The user can select the object by clicking. |
| Default | Highlight BUTTON objects graphically, meaning that pressing that Return key will be equal to clicking this object (when calling the AES function `form_do()`). Within a Dialog box, you should only set this flag for **one** object. |
| Exit | `form_do()` ends when the user clicks the object. |
| Editable | The object is editable. This flag **may only be given** to FTEXT and FBOXTEXT objects. If you do not follow this rule, the AES crashes when executing the Dialog box! |
| Radiobutton | The object is a so-called Radiobutton. This is similar to the selector buttons of an old radio (hence the name), of which only one can be pressed at a time. Translated to the AES, this means that of all objects with this flag set and that are in the same hierarchical level of an object tree, only one can be selected. If another one is selected, all others are automatically deselected. |
| Touchexit | If such an object is touched with the mouse button held down, the function `form_do()` immediately aborts without the mouse button having to be released first. |
| Extended | These flags are not used by GEM and may be used for their own purposes (e.g., extended object types). |

#### 2.12.2.4 object status

The status of an object determines its state. An object can have multiple states, e.g. be selected (SELECTED) and shaded (SHADOWED). They are summarized here:

| Object Status | Meaning |
|---|---|
| Selected | The object is displayed inverted. This status is automatically set for the object with which a Dialog was exited. |
| Crossed | A white cross is drawn inside the object. |
| Checked | A tick is drawn in the upper left corner of the object. |
| Disabled | The object is drawn bright and is not selectable. |
| Outlined | An additional frame is drawn around the object. |
| Shadowed | A shadow is drawn under the object, which is visible on the right and under the object. Buttons that are drawn with this status indicate a popup. |
| Draw3d | A three-dimensional effect be created by drawing the icon three times at an angle. Only for ICONBLK structures and only as of GEM 2.0. DRAW3D is not supported by any TOS version. |
| Whiteback | Also only for ICONBLK structures and only from GEM2.0: with white background the icon mask is not drawn. WHITEBAK also works on all TOS versions available for the Atari, but has never been documented by Atari. Atari Desktop itself uses WHITEBAK in windows to speed up screen output. |

#### 2.12.2.5 Coping and Moving Objects

Moving an object is simple: Just drag it to its new position with the mouse. That's it.

When moving an object, you can now move the object “under” another object. It is not automatically assigned to the target object as a child if you respond yes to the “Do you want to change the object structure?” Alert box. This Alert appears only if the moved object and the target object share a common parent. This is to prevent that an object may be accidentally outside the parent.

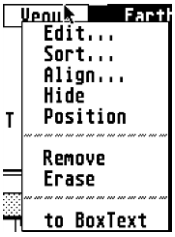
If you move several objects within a window, the objects remain selected even after the move action, so that you can move them further if necessary and do not have to re-select them all again. In addition, Interface leaves the object order unchanged, only the concatenation is eventually changed.

If one or more objects are to be copied, move the objects to another position and press the Shift key when releasing the mouse button. The copy action will not be performed if the object has not moved at least one pixel.

#### 2.12.2.6 Selecting multiple objects

To select multiple objects for editing at the same time, there are two options. First, hold down the Shift key and click on any objects you want. Second, hold down the Alternate key and drag a frame around the desired objects: after releasing the mouse button, all objects completely within the frame are selected. Several selected objects can be moved, copied, deleted and opened together. With a common opening (for example with F2) all selected objects can be processed one after the other without clicking again. In addition, with several selected objects, the object flags can be changed at the same time (see Set Flags...).

### 2.12.3 The Object Popup Menu



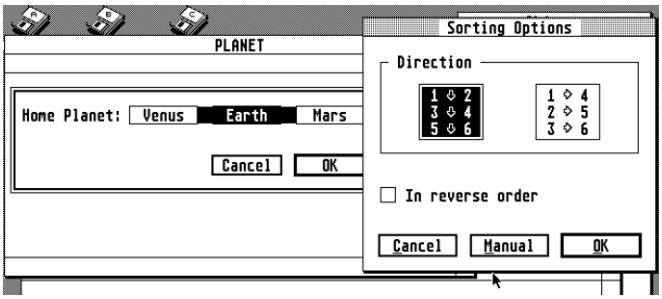

If an object is clicked only once, a Popup Menu appears, which provides some special functions. However, it can also be switched off with the option **Object Popup** in the **Options: Preferences...** Dialog (see Object Popup). All functions contained therein can also be accessed via the Menu bar in the **Object** Menu. This makes the selection procedure of Interface “compatible” to other programs (clicking on an object selects it instead of making the popup appear). The **Erase** option in the Object Menu also allows deleting trees from a resource file window.

#### 2.12.3.1 Edit…

Has the same function as **File: Open**, **Object: Edit**, or a double-click on an object. This entry is available in case you should fail to double-click.

#### 2.12.3.2 Sort…

GEM always draws the objects of a Dialog in the order in which they were copied to the Dialog. Now, if you had copied in the headline last, that headline, though it's at the top, would always be drawn last. Of course this does not look very good, and to prevent that, you can sort the order of all the objects in a Dialog. In addition, the order of the objects is also important for the programming, because the sorting can ensure that behind or underlying objects get a continuous numbering. This makes it possible not to query all these objects individually in the program, but to test whether an object index lies between the first and the last object index. All levels of a Dialog can be sorted individually. In our box ('PLANET') there are two of them. The first level is the large frame (the root object) in which all other objects are located. The second level is formed by the IBOX object because it contains other objects. The sorting only affects the children of the selected object. The Children of Children are not co-sorted. This could, for example, allow the objects of the root object to be drawn from top to bottom, while the objects in the IBOX are drawn from left to right.



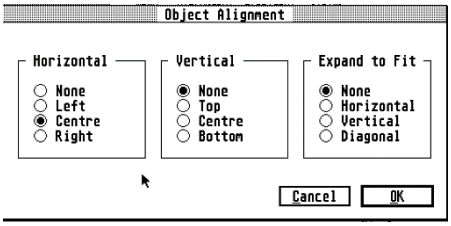

To sort, click once on the root object[2] and select Sort in the following Popup Menu. The displayed Dialog (see image) offers you two possibilities, which are symbolized by arrows. Either all children of the selected object can be sorted from top to bottom (left field) or from left to right (right field). The Reverse Order button reverses the order, i.e. the children are

[2] If you have difficulty to select the large frame, just Control click inside the frame (DrCoolZic)

sorted from the bottom to the top or from the right to the left. For very special cases, there is also the option Manual, with which the sequence can be defined as desired. To do this, Interface terminates the Dialog and displays all eligible objects inversely. Sorting is now done in the order in which you click on the objects with the mouse. An object selected in this way is displayed as “normal” again, so that you have an overview of which objects still have to be selected. Important: all affected objects, i.e. all children of the parent object, must be clicked to terminate the function. To cancel prematurely, press Esc or the right mouse button.

For our exercise, select the left button (sort from top to bottom) and end the Dialog with OK. Click on the invisible IBOX object with the radio buttons. If you cannot find it, it's best to click between the 'Venus' and the 'Earth' button. Then select the item 'Sort' in the popup and close the sort Dialog with OK.

### 2.12.3.3 Align…




An object can hereby be centered horizontally and / or vertically (relative to the parent). In addition, in the displayed Dialog it is possible to adapt the object size to the size of the parent object.

### 2.12.3.4 Locking

If an object is not exactly on character coordinates, it can be moved to a position using this function.

### 2.12.3.5 Hide

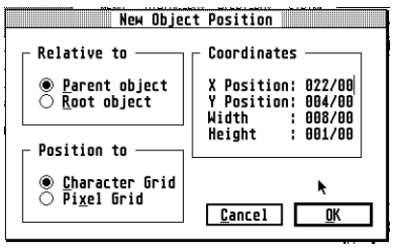
This option sets the HIDETREE flag in an object, which cannot be changed in the “normal” edit Dialog. This hides the object and all its children and is neither visible nor selectable with the mouse.

Entries in ‘EDITABLE’ FTEXT objects are still possible, but are no longer displayed on the screen. This feature (or bug?) can be used for password input. To make a hidden object visible again, you must use the following unhide function.

### 2.12.3.6 Unhide

This Popup entry only appears if there are any (child) objects in the clicked (parent) object in which the HIDETREE flag is set (see above). After selecting the entry, they are displayed again.

### 2.12.3.7 Position




With this an object can be placed by direct input of coordinates. In the appearing Dialog you can enter all coordinates (x, y, width, height) yourself. Interface ignores but nonsensical values. There are also a few more buttons in the Dialog with the following meaning:

- In the group ‘relative to’ the radio buttons parent object and root object: If the parent object is selected, the coordinates refer to the parent of the object. If root object is selected, it refers to the root object, i.e. they are relative to the whole tree.
- In the group ‘position to’: The ‘character grid’ button causes the object coordinates to be evaluated in character (letter width x height, i.e. in general 8 x 16). The deviation from this can be entered in pixels behind the ‘/’ bar. If the ‘pixel grid’ button is selected, the object coordinates are evaluated in pixels.

### 2.12.3.8 Remove

The selected object is deleted, but all its children remain in Dialog. There is no undo possible!

### 2.12.3.9 Erase

The selected object and all its children are deleted. The deletion can be undone by the Undo key.

### 2.12.3.10 Changing object type

Some objects can be converted to another object type. The following conversions are possible:

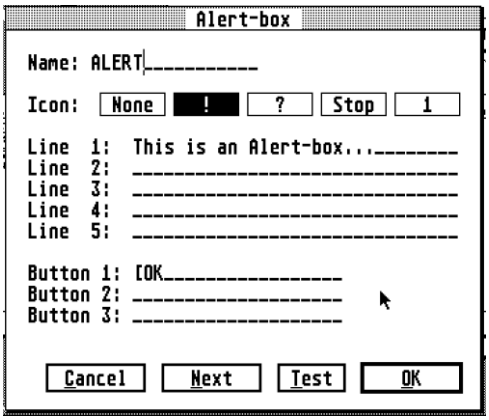
- STRING into TEXT
- BUTTON into BOXTEXT
- IMAGE into ICON and vice versa. However, data can be lost because the individual object formats sometimes differ greatly. In the conversion from ICON to IMAGE, e.g. lost the letter, the icon text and the mask of the icon.

### 2.12.3.11 Last but not least: Esc and Undo

Sometimes it can happen that GEM does not draw a Dialog correctly in a window. When scrolling up or down, for example, it can happen that the edge of a Dialog is suddenly only 1 pixel thick instead of 2 pixels. This is not a bug of Interface, but a strange practice of GEM. If this (very rare) error should happen to you, you can force the complete window to be redrawn with the Esc key. This key is also especially useful if you want to check the new Dialog structure (the effect on drawing) after sorting a Dialog.

Moving operations within the same window as well as deleting and resizing objects can be undone immediately afterwards with the Undo key. Since the undo function of Interface only copies the deleted or moved objects back to their old screen position, the object sequence is not necessarily brought back to the old state. Therefore, you should re-sort the Dialog after an undo to be on the safe side. If you have enlarged an object and adopted a new object, the adopted object still belongs to the enlarged object after the Undo operation. Therefore, the parent object can only be partially resized because the adopted object cannot protrude from the parent object.

## 2.12.4 Editing an Alert



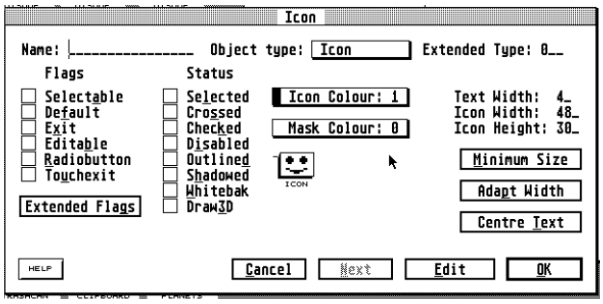

Interface displays all the data of an Alert box within a special Dialog. In the upper part you can select the desired icon, including a maximum of 5 text lines and 3 button texts.

GEM can display an Alert box with three different icons: ‘exclamation mark’, ‘question mark’ and ‘stop sign’. In addition, you can also do without an icon. These four options can be selected directly. In addition, there is a fifth button in which a character can be entered from the 36 values: 0-9 or A-Z. The values one to three correspond to the above icons, while zero corresponds to no icon. You should only enter the other values if you use your own routines to display Alert boxes (for example, FlyDials or MyDials). GEM cannot cope with higher values than 3 and either displays a broken graphic within the Alert boxes or crashes immediately.

The 5 text lines of the Alert Dialog actually form a single large input field. When you have reached the end of a line with the cursor and enter further characters, text and cursor appear in the next line. The Enter, Delete, and Backspace keys also work across lines. Thereby it is e.g. possible to drag a text that is in the second line to the first line. If you want to achieve the opposite, that is, the text should be from the cursor position to the next line, you can press Shift Insert.

Of course, these features may not provide the convenience of an editor, but they should be a great deal easier. The same editing options are available in the multi-line input fields for buttons and free strings. In the Alert Dialog there is the button “Test”. This does not require you to leave the Dialog extra to test the Alert box.

## 2.12.5 Editing an icon or image



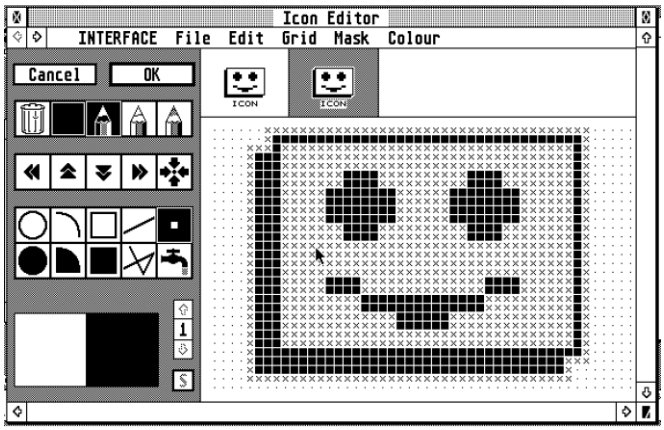

One of Interface's “cream pies” is its icon editor, which lets you edit icons, color icons, and images.

If one of these objects is to be changed (for example, after a double-click on it), a Dialog appears in which the data of the object can not only be displayed but also **edited**. The Dialogs are similar in structure; differences arise only in that images cannot have all the attributes. For example, an image does not have text, and does not have mask.

In the following all options are discussed. If one of the enumerated settings options for an image does not exist, obviously you cannot give it this attribute. We generally speak of icons but this include images and color icons. You already know most of the inputs from the comparable Dialogs for other object types. They are only briefly mentioned here:

- Name: This field contains the name of the object; under which it can later be addressed by a programming language.
- Object Type: This entry contains a Popup Menu with which the object type can be subsequently changed from 'Icon' into 'Coloricon' and vice versa.

- Extended Type: GEM allows the programmer to use own object types as well. To be able to use different self-defined object types, they are numbered consecutively. A separate object type can have a number between 1 and 255. This number is entered in the OBJECT structure in the high byte of `ob_type`.
- Flags: The flags describe the properties of an object. Since an object can have several properties - in some cases - the flags can be selected by checkboxes.
- Status: The status describes the representation or the state of an object. Since several properties can also be combined here, the status must also be defined by checkboxes.
- Extended flags: GEM uses only part of the possible bit combinations for flags and status. With the Extended Flags button you can open a Dialog in which you can set (for your own object types) the unused bits.
- The icon color as well as the color of the mask can be set in two Popup Menus. If Interface was started in a color resolution, the available colors are displayed in the popups. Otherwise, the color can be read from the texts of the Menu, which are based on the GEM default setting.
- Below the color setting for the icon mask, the icon itself is displayed. If the icon exceeds a certain size, only the top left part of it will be displayed.
- An icon or color icon also includes two text elements, namely a ‘drive letter’ and an ‘explanatory text’ whose length is specified in text width. To make entries in the corresponding fields, you must position the cursor there using the cursor keys. The mouse cannot be used because it moves the two fields.
  The explanatory text (usually under an icon) can be aligned horizontally with the Center Text button so that it is positioned in the middle of the icon. The text length can be changed directly with the button 'Adjust width', without having to leave the icon Dialog.
- You can define the size of an icon in the fields: Icon width and Icon height. Since the width must be a multiple of 16, Interface may correct your inputs to round to the nearest multiple. The largest possible icon you can manipulate with Interface is 640 x 400 pixels.
  If you reduce the size of an icon at a later time, the width and height are changed so that data is removed on the right and bottom. Interface will alert you if this would cause an actual image loss (it may be that you are removing blank parts of the image).
- In order to avoid wasting memory, icons and images can be minimized as far as possible by the button **Minimum Size** without losing data. The image is pushed so far to the top left, until no white border exists. Then the right and bottom edges are cut off. Finally, the image is centered again.




Finally, you have access to the icon editor from this Dialog. To do this, either click on the icon displayed in the Dialog or on the **Edit** button.

In the icon editor, an icon can be changed, redrawn or cut out of an image graphic. As soon as the window is closed or the **OK** button is clicked on, the new data is adopted. **Cancel** rejects the changes.

The last change of the picture in the icon editor can be undone with Undo.

The editor presents itself in a window with its own Menu bar. To activate the Menu line, you have to click on one of the titles. The window is divided into three areas: on the left you will find the drawing function and the color selection, above a display of the icon in original size, the 'Preview' and below the actual drawing area.

The preview above the canvas has two functions: First, it shows how the icon looks in normal size against different backgrounds. Second, you can click the icons (independently) to see what the icon looks like in selected state.

### 2.12.5.1 Drawing with the editor

After opening the editor, the option to draw individual black dots is automatically selected. If you now click on a white field on the drawing area, it will turn black - if you hold down the left mouse button and move the mouse, all other fields that are “touched” by the mouse cursor will also turn black. If you click on a black box, it will turn white and all fields touched by the cursor while holding down the mouse button will also turn white.

The drawing of color icons works analogously. After selecting a color, in the editor, click on a field with any (other) color to draw in the selected color. If you click on a field that is already displayed in the selected color, it deletes this field. The currently selected character color is displayed to the left of the trash can. Below the **OK** button you will see three crayons, of which - if you did not change anything - the left pen with the black tip is selected (see above). It indicates that you are drawing with the selected color and that the color change just described is set. If you select the pencil with the white tip, you can no longer draw but only delete. If the right pen with the “gray” tip is selected, you draw the mask. This pen works like the colored pencils, i.e. you can use the same function to set mask points and delete (any points).

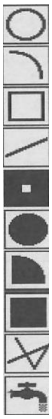
### 2.12.5.2 Digression about Mask

Mask? What's that? Icons do not only consist of the actually displayed picture, they also contain a second image called a 'mask'. The mask is automatically displayed when an icon is selected. Whenever an icon is selected, all points of this icon, in which the mask is set, are inverted. Interface draws (invisible to you) under each color point of an icon also a mask point.

If an icon is not selected, all points where only the mask is visible will be drawn white.

### 2.12.5.3 Drawing Functions

The three pens are taken into account for all drawing functions. This means, for example, that you can draw a filled circle with one color, delete it or draw it as a mask. The following overview shows the different drawing functions.



Circles and ellipses
Circular or elliptical arcs
Squares and rectangles
Lines
single points
filled circles and ellipses
filled circular or elliptical arcs
filled squares and rectangles
Polygons (lines)
Fill in the area

All geometric drawing functions work in the same way. The first mouse-click defines the center of circles, the starting point of arcs, lines, and polygons, and any corner of rectangles. After the first click, you can release the mouse button and determine the appropriate figure by moving the mouse, which is taken over with the second mouse click in the editor. The drawing can be aborted before the second mouse click by a click with the right mouse button. To set a single point and to fill a surface, a simple mouse click is all that is needed.

The last change of the picture in the icon editor can be undone with Undo.

### 2.12.5.4 Changing drawing area

The drawing area of the editor generally shows an enlargement of the icon. Because of this it can happen, with large icons (and a small monitor), that an icon is not completely visible. Therefore, Interface supports three different magnification factors that can be selected in the **Grid** Menu. **Large** means that 8 x 8 pixels are drawn for each dot. **Medium** stands for 6 x 6 pixels and **Small** for 4 x 4 pixels per dot.

If that's not enough, you can hide the preview above the artboard. To do this, switch off the option **Edit: Icon Preview** in the Icon Editor.

### 2.12.5.5 Moving an icon

Above the drawing functions palette, there are five icons with arrows pointing in different directions. By clicking on the first four icons (from the left) you move the displayed image in the direction in which the arrow points. The “arrow cross” serves to center the image within the drawing area.

### 2.12.5.6 The clipboard

The icon editor supports the GEM clipboard. All icons and images are saved and loaded in XIMG (color image graphics) and ICO (IconEdi) formats.

**Edit: Copy** or Control C copies an icon to the clipboard.

Using the keyboard, you can also use Shift Control C to copy a section of the editor onto the clipboard. After you have pressed the key combination, you can draw a frame on the drawing surface. Its content is copied to the clipboard. You can cancel this function with a right mouse click.

**Edit: Cut** or Control X copies the icon to the clipboard and then deletes the canvas.

**Edit: Paste** or Control V transfers the content of the clipboard to the editor. If the image to be pasted is smaller than the drawing area of the icon editor, you can place it exactly with the mouse beforehand. In this case, the new graphic is drawn transparently (!) over the old graphic, i.e. white parts of the inserted graphic do not delete any points of the old graphic.

If the image to be pasted is larger or the same size as the “old” image, it completes the old graphics completely.

Tip: If you load a Windows or OS/2 icon in the Icon Editor, there is often the problem that the “only” icon is available in 16 or 256 colors. There is a little trick to take over at least the outlines in monochrome without effort: First, copy the 16-color icon with **Edit: Copy** to the clipboard. Then call the file selector (with Control O) to find the '`CLIPBRD`' folder (which will be on drive A or C).

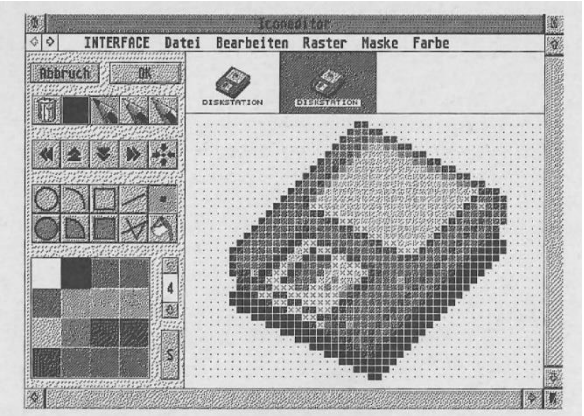
In the folder you should now find two files: `SCRAP.ICO` and `SCRAP.IMG`. If you delete `SCRAP.ICO` with the file selector and select in the monochrome resolution level of the color code **Edit: Paste**, the first plane of the image file will be inserted. The outlines of the color icon should now be visible and serve as a template for the monochrome resolution.

The whole works, because Interface inserts an image only in the Icon editor, if no ICO file in the Clipboard exists. An ICO file also contains the monochrome resolution of the icon, which is still empty for the Windows or OS/2 icon. By deleting the ICO file, you force Interface to insert the IMG file.

### 2.12.5.7 Color Icons and Animated Icons

So far we have not discussed the special features of color icons because it was not important for the basic operation of the editor. So now the past is being made up.

Color icons are available in the *Falcon's desktop* and in *MultiTOS*. With the color-coded routines included with Interface, you are able to use color icons in your own programs **in all TOS versions**. A color icon generally has some special features that you should consider. First, a color icon can consist of up to four different icons for different resolutions. Interface supports icons in 2, 4,16, and 256 colors. So that your programs can also display in monochrome icons, you should therefore provide color icons with at least two images (one monochrome, the other in color). Second, color icons can represent a *selection image*, that is, a selected icon does not display the mask, but a second image. ATARI refers to such icons as 'animated'. If no selection image is available, selected icons are linked with a dotted (black) mask.




If you drag a color icon into a form using **Edit: Objects...**, this icon already consists of four images. One picture in monochrome and sixteen colors as well as the corresponding selection pictures.

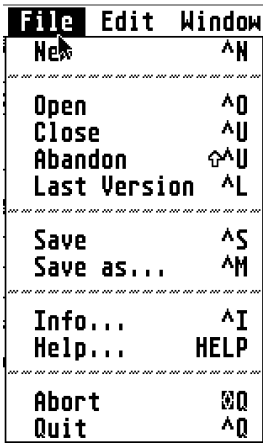
In order to switch between the different pictures, you have two options: In the color Menu of the editor you can change the resolution to the one you want. If there is no picture for this resolution, it will be created. With **Color: Selected Icon** you can switch between the “normal” picture and the selection screen. If no selection image exists, it will be created.

The buttons next to the color selector on the left edge of the editor have practically the same function. The button with the ‘**S**’ switches between the normal icon and the selection screen. If the selection screen is displayed, the button is also selected. Use the arrows above and below the number ('4' in the picture) to toggle between the different resolutions. The displayed number indicates how many plans the currently edited image has.

The additional options that are available for color icons can be found in the Icon Editor’s Reference.

# 3 Interface's Reference

## 3.1 The File Menu




### 3.1.1 New

With this function you can create a new resource file. The new resource will be named 'UNAMED' to indicate that it can initially only be saved with **File: Save as....** An icon for the file is displayed on the Interface desktop.

### 3.1.2 Open

If icons or objects are selected, they can be opened with this function as with a double-click. Otherwise, the file selector appears, in which a resource file can be selected for loading. If the name specified in the file selector is not found, a new resource file with this name will be created.

### 3.1.3 Close

If a window is opened when this option is called, it will be closed. If, in addition, the corresponding icon of a resource file is selected, the file is removed from the memory after a security inquiry.

### 3.1.4 Abandon

The resource file that belongs to the topmost window is removed from memory. If the file was changed, you will be asked if it should be saved.

### 3.1.5 Last version

This option discards the current content of the resource that owns the topmost window and reloads its last saved content. If the resource was changed, you will be asked if you really want to do that.

### 3.1.6 Save

If the icon of a resource file is selected on the desktop, the corresponding file is saved. Otherwise, the file to which the topmost window belongs is saved. Header files are only saved if labels have been changed or deleted, the object hierarchy has been changed, or if the Resource Info Dialog (**File: Info...**) has been left with **OK**. This is likely to please all compiler users whose compilers would otherwise recompile the entire project.

If the resource file has been newly created and therefore has the designation 'UNAMED' (see above), Save is highlighted and cannot be selected. In this case, the file must be saved with **File: Save as ...**.

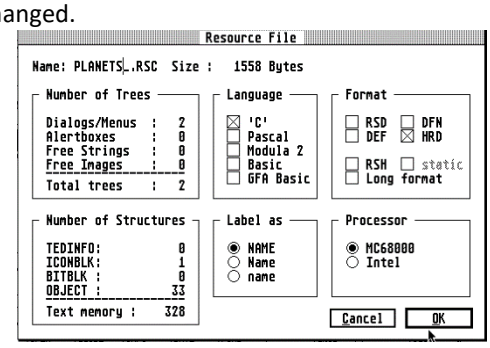
### 3.1.7 Save as...

With this function, a resource file can be saved with a new name to be entered in the file selector. If a new name has been entered, the previous name in the corresponding icon will be replaced by the new name.

### 3.1.8 Info...

If an icon is selected on the desktop, a Dialog with information about this icon appears. If no icon is selected, you will get information about the topmost window. If no window is open, the Copyright box of Interface will be displayed. In some information Dialog boxes data can also be changed.

#### 3.1.8.1 Info of a selected resource file[3]




A Dialog with a lot of information about the resource file appears.

The following information can be changed:

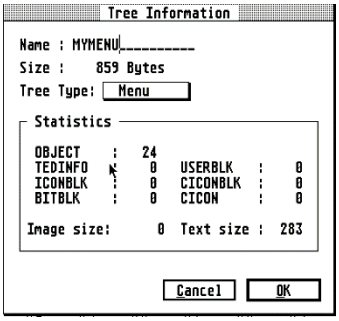
- The name of the file
- The format of the definition file: DEF, DFN, RSD or HRD (see Chapter 2, Saving Resource file). In the definition file, Interface stores all the important data of a resource (for example, labels and presets). *If possible, you should use the HRD format* because most information can be stored in this format. The HRD format is also the only format that allows labels with 16 letters (otherwise 8).
- The programming language

[3] If you quit this dialog with the **OK** button this will force the Header files to be saved even if not changed. Therefore, in most cases you should use the **Cancel** button (DrCoolZic).

- The processor type (ATARI-GEM or MSDOS-GEM)
- The ‘case’ of labels: all uppercase, first letter in uppercase, all lowercase.
- The RSH source code output. If you select the static button, each variable of the source code output will be preceded by a static word. This defines all variables in module locally, and you can manage several resources in the C-Source in a program without naming conflicts. If you activate the Long format button, the resource is output in a special format that allows (practically) any length of resources. Nevertheless, the resource format remains largely compatible with the old format; Interface only changes all variables of the resource header from 'WORD' to 'LONG' so that more than 64 KB can be managed.

On the Interface diskettes you will find routines in C and Basic, with which you can load these resources.

#### 3.1.8.2 Info of an object tree




The following information can be changed:

- The name of the tree
- The tree type. It is possible to convert the type:
  - Alerts into free strings,
  - free strings into Alerts (if allowed),
  - Menu trees into Dialog trees and
  - Dialog trees (if possible) into Menu trees.

The latter option should not be used, as Interface can no longer check and correct the structure of a Menu tree if it has previously been converted to a Dialog tree. At most, this option is needed if you have icons etc. that you would like to accommodate objects in Menu lines. However, you should avoid this as far as possible, since such objects in Menu lines basically have nothing to do here.

### 3.1.9 Help...

Normally, a Dialog will appear here explaining the function of the Control, Shift and Alternate keys. If no icon of a resource file has been selected and the uppermost window does not belong to any resource file, there is no help information.

### 3.1.10 Abort

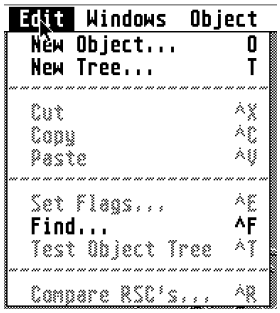
Interface is left without the configuration being saved (even if “Auto-save” is activated).

### 3.1.11 Quit

With this entry Interface can be left. If you have made any changes to a resource before exiting, you will be asked if the file should be saved. In addition, Interface stores a configuration file on request (see Auto-save preference), so that you will find the same working environment at the next program start as when exiting the program.

## 3.2 The Edit Menu

### 3.2.1 New Objects ...




A Dialog appears in which the different object types are contained. In Chapter 2 you will find a list of all object types with their meanings. If you need an object, drag it from this Dialog to your working window. In Menu bars and associated Pulldown Menus only the object STRING is allowed. All other objects are ignored. Objects can only be dragged onto ‘Dialog tree’ or ‘Menu tree’ windows. If the object box is in a window, you can also extract objects under MultiTOS if the box is in the background. Without MultiTOS, you can achieve the same effect by simultaneously pressing the left and right mouse buttons while clicking on an object.

### 3.2.2 New Trees ...

This Dialog contains all ‘tree types’:

| | |
|---|---|
| Menu | It contains Pulldown Menus in which the user of a program can select various Menu options. Menu options consist of STRING objects. |
| Dialog-box | A Dialog is used to display information and enter data. A Dialog can contain all types of objects. |

| | |
|---|---|
| Alert-box | An Alert is a special kind of Dialog. It is used to display important messages, warnings or errors. An Alert, just like free strings, consists only of text, from which the GEM then generates the alert in the `form_alert()` function. |
| Free String / Image | Here individual texts or graphics can be stored in a resource, which should not be managed by GEM but by the program itself. |

Trees may be dragged either to a resource file window or to the icon of a resource file. If the tree box is in a window, you can also extract trees under MuItiTOS if the box is in the background. Without MultiTOS, you can achieve the same effect by simultaneously pressing the left and right mouse buttons while clicking on a tree.

### 3.2.3 Cut, Copy and Paste

Using these items, you can copy all selected objects from Dialog and Menu blocks to the Interface clipboard, or from there, insert them at the mouse position.

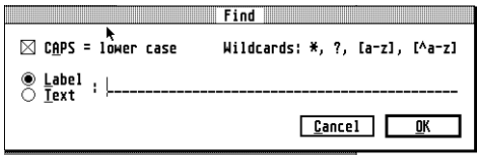
### 3.2.4 Set Flags...

If several objects have been selected in a Dialog window, you can set the attributes (Selectable, Exit, etc.) for these objects simultaneously with this function, without having to open each object individually.

After selecting this option, a Dialog appears in which the current status of the attributes is displayed almost as in the normal object Dialogs. One minor difference is that attributes can be set differently for multiple objects. To indicate this, the checkboxes of such attributes are filled with a dotted pattern (tristate button).

If you now select or deselect a button in this Dialog, this attribute is set or deleted for all selected objects. If you set the box to “dotted pattern”, this attribute will not be changed.

### 3.2.5 Find...




Here you can search for a specific object name (a label), or text in all selected resource files. If no file is selected, the file that belongs to the topmost window will be searched for. In the search interface scans the entire resource. The case distinction can be switched on.

The following wildcards are allowed in the search string:

| | |
|---|---|
| * | Any number of arbitrary characters |
| ? | Any character |
| [a-z] | Only certain characters are allowed at this point. [c-e] would denote that only the characters “c”, “d” and “e” are permitted here. |

Some examples:

| | |
|---|---|
| *text* | searches for all passages containing the string 'text'. |
| text* | searches for all text that begins with 'text'. |
| te?t | searches for all text beginning with 'te', containing any character in the third position, ending with 't'. |
| te[w-y]t | searches all passages starting with 'te', either 'w ',' x' or 'y' in the third place, ending with 't'. |

If Interface finds the entered name, there are two options: If the name you are looking for is a tree name, Interface opens the corresponding tree window and selects the tree. If the searched object is an element of a Menu or Dialog tree, Interface opens the tree, lets the object blink 3 times and then selects it.

### 3.2.6 Test Object Tree

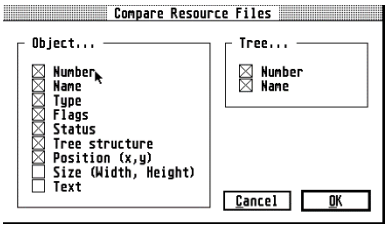
If the icon of a Menu, Dialog, or Alert tree has been selected or a corresponding window is at the top, the tree can be tested with this option. If a Dialog, or Alert tree does not have a button to exit, Interface will alert you. The function cannot be executed then. If the tree being tested is left by clicking a button or Menu item, Interface displays the object number and, if present, the name of the object clicked.

If you use your own drawing routines for self-defined object types (extended `ob_types`) and these routines have been reloaded by Interface (`EXTOBFIX.PRG`), then these objects are correctly drawn and executed in the test function. These reloaded routines can also replace the complete Dialog, and Alert trees test routine. These

routines can be switched off with **Options: Preferences...** if necessary(see Display External object) . For more information on self-defined object types and the EXTOBFIX.PRG, see Chapter 5. If a Dialog is to be tested and contains an 'Editable' object that is not of the type FText or FBoxText, a warning is issued because the AES is editing such an object crashes.

Since it is possible for such an object to be specially treated by an EXTOBFIX file, the test process can still be called via the Alert box. So be careful! If a Menu tree is to be tested and is switched on in the Preferences... - Dialog of the Options-Menu Arrange Menu-Keyboard Shortcuts, Interface issues a warning if shortcuts are used twice.

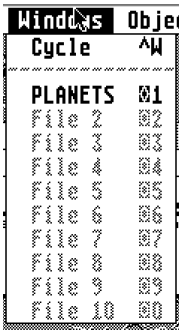
### 3.2.7 Compare RSC's…




Two resource files can be compared. The comparison makes it easier, for example, to process foreign-language resources, which are easier to maintain and errors can be found more quickly. To trigger the function, you must select two resource files and then call the resource comparison[4]. The option can also be selected if exactly two resources have been loaded or if the tree windows of exactly two resources are open.

When a particular tree of one of the resources is selected or opened, the comparison can be made optionally from the beginning or from this tree.

## 3.3 The Window Menu




### 3.3.1 Cycle

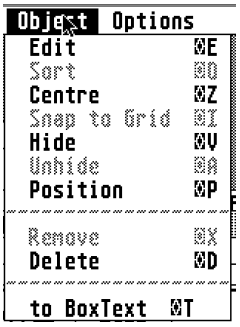
With this function, the lowest window can be brought up. Multiple applications allow all open windows to be brought to the foreground one after the other. If you operate this option via keyboard (Control W), it is possible to get to each window quickly without many mouse actions. The desktop window is not "cycled".

### 3.3.2 File n

In this Menu, the first 10 loaded resource files can be selected The purpose of this action is that you can also open windows for resource files with the desktop switched off. Of course you can also use it to quickly top (or open) the window of a particular file.

The resource files 11 to 15 can still only be accessed via the Interface desktop.

## 3.4 The Object Menu




All entries in this Menu correspond to the Object Popup Menu (see chapter 2) and are used to call these functions when the Popup is switched off (see Object Popup).

### 3.4.1 Edit

Opens the Dialog with the settings for the selected object. You can also call this Dialog by double-clicking on an object (see Edit).

### 3.4.2 Sort

Opens the Dialog for sorting the objects within a parent (see Sort…).

### 3.4.3 Center

Opens the Dialog for aligning an object relative to its parent or root object (see Center).

### 3.4.4 Snap to Grid

Aligns the selected object to the selected grid, so it may shift to the left and above (see Snap to Grid).

### 3.4.5 Hide

Sets the HIDETREE flag for the selected object (and its children), that is, the object is no longer visible and cannot be selected (see Hide).

[4] You can filter the type of trees as well as the type of objects you want to be compared. If differences are found, the differences will be displayed one by one (DrCoolZic).

### 3.4.6 Unhide

Make a hidden object visible again (see Unhide).

### 3.4.7 Position

Opens a Dialog in which coordinates and size information can be entered directly for the selected object (see Position).

### 3.4.8 Remove

Removes the parent object in the selected object, but leaves its children in the form (see Remove).

### 3.4.9 Delete

The selected object and all its children are deleted. The deletion can be undone by the Undo key (see Delete).

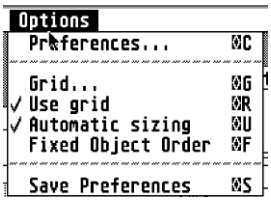
### 3.4.10 To BoxText

The object can be converted to another object type (see To BoxText). The following conversions are possible:

- STRING in TEXT
- BUTTON in BOXTEXT
- IMAGE in ICON and vice versa. However, data can be lost because the individual object formats sometimes differ greatly. E.g. in the conversion from ICON to IMAGE, lost the letter, the icon text and the mask of the icon.

## 3.5 The Options Menu

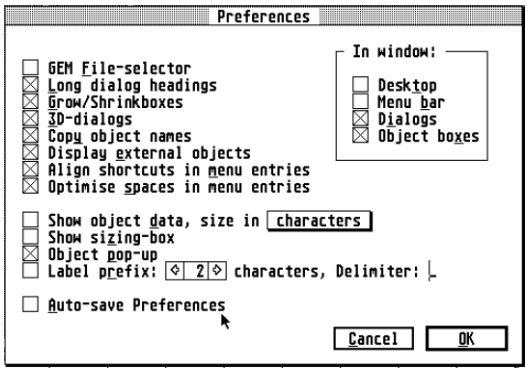
### 3.5.1 Preferences ...




In the Preferences... Dialog, you can change a number of parameters that influence the work with Interface. There are the following settings:

#### 3.5.1.1 GEM file-selector

Interface uses its own file selector (file selection). If you do not like it or you prefer to use your own, you can turn it off and turn on the original GEM file selector (or another file selector that replaces GEM).




#### 3.5.1.2 Long Dialog headings

This option is the result of a disagreement. Earlier (in the first development phase) Interface used the FlyDials of Julian Reschke. During that time, Interface had a long underscore underneath each box heading that almost always reached the right edge of a Dialog. Unfortunately, this did not meet the approval of Julian Reschke, who insisted (and was allowed to) on a FlyDial-compliant look (with a short underline). Finally, Interface was developed with the MyDials and since we liked the long underscores much better, they are always on. Anyone who likes the FlyDial headlines can switch off the long headlines here. To see the change, you must first save the parameters after selecting this option and then exit the program and start again.

#### 3.5.1.3 Grow/Shrinkboxes

When opening and closing windows and Dialogs, a larger or smaller rectangle is always drawn. If you want to save time, you can switch it off here.

#### 3.5.1.4 3D-Dialogs

Switches off or on the display of the 3D Dialogs in Interface.

#### 3.5.1.5 Copy object names

If this option is selected and you copy objects or entire trees to another resource file, the object names will be copied as well. If certain names already exist in the new file, Interface will point you to this and give you an opportunity to enter a different name.

#### 3.5.1.6 Display External object

If a program file with the name EXTOBFIX.PRG is in the directory of Interface, it will be loaded at program start. This file can provide interface drawing routines for extended object types (extended ob_types). It can also replace the test functions for Dialog and Alert boxes (see Creating an EXTOBFIX program).

If you do not want to use the functions of this file, you can switch it off here.

#### 3.5.1.7 Align shortcuts in Menu entries

When this item is selected, any keyboard shortcuts (for example, '^ Q') in Menu items are automatically aligned right-justified so that all shortcuts are aligned.

In addition, when calling the test function (F9), Interface checks whether keyboard shortcuts have been used twice and then issues a warning.

#### 3.5.1.8 Optimize spaces in Menu entries

According to the GEM conventions, there should be 2 spaces in front of each entry in a Pulldown Menu followed by 1 space. Interface automatically ensures that the correct number of spaces is inserted or deleted. If you want to have control over the spaces in front of and behind the entry, you can deactivate this optimization here.

#### 3.5.1.9 Show object data, size in characters / pixels

If this item is selected, the info line of a Dialog window shows which object is under the mouse arrow (the number), how it is called and how wide and high it is. The width can be displayed either in letter coordinates (default setting) or pixel coordinates. When specifying letter coordinates, a number is specified behind it that indicates the pixel deviation from the letter grid. If you hold down the Control key, the object data of the parent of the object lying under the mouse will be displayed.

For example, 'PLERDE: 2 w = 34/00 h = 05/09' means that under the mouse is the object number 2 with the name 'PLERDE' and that it has a width of 34 characters. The height is 5 characters + 9 pixels. The object data display is also displayed in Menu trees. In addition, all 16 characters of the object name are output.

Strangely, if the object data display or the sizebox is turned on, Interface will experience double-click detection problems when two events arrive simultaneously (MU_BUTTON and MU_M2). This is due to the somewhat “idiosyncratic” behavior of evnt_multi.

We've tried to do something about it by replacing the MU_M2 event with a MU_TIMER event. After a single click, Interface now waits to see if another click immediately follows. If so, both clicks are combined into a double click. Double clicks are now recognized much better. Unfortunately, as a side effect, it sometimes takes up to half a second to respond to a single click. This only happens if it is a “suspicious” single-click (for example, if two events were reported simultaneously).

#### 3.5.1.10 Show Sizing-box

Anyone who could never find the magnification corner of an object can now visualize it with this option.

#### 3.5.1.11 Object Popup

This option turns off the popup that otherwise appears when selecting an object. All functions contained in it can therefore also be reached via the **Object** Menu. This makes the selection process of interface “compatible” with other programs (clicking on an object selects it instead of making the popup appear).

#### 3.5.1.12 Label prefix

If this item is selected, Interface will require each newly entered label to have an adjustable number of characters of the tree name. A separator can also be defined. The letters are only hung in front of the label if the object has not yet had a name.

An example: A new object in the tree 'SETTINGS' should be named 'OK'. Label prefix is set to 3 characters, the separator is set to '_'. This assigns the object the name 'SET_OK'.

#### 3.5.1.13 Auto-save preference

When this option is turned on, all interface settings and most of the current desktop configuration data are automatically saved when exiting Interface with **File: Quit**. As a result, you will find Interface again in the same state after a later start.

The following is stored:

- The position of all icons
- The size, position and slider position of all resource file windows (this includes the tree window, Dialog window and Menu window)
- The position of the tree and object boxes
- The size, position and slider position of the desktop window - The paths of all loaded resource files
- The position of each open tree window. When you load a resource, the window opens in the old location. Interface behaves similar to the ATARI desktop.

If necessary, all coordinates are adjusted in the event of a change in resolution, since still all objects can be reached.

#### 3.5.1.14 In window

##### 3.5.1.14.1 desktop

Under MultiTOS it could be annoying that Interface creates its own desktop. In order not to have to switch between different desktops of different programs there, you can force Interface with this switch to put your own desktop in a window. The change will not be active immediately. You first have to save the parameters, exit interface, and reload it.

##### 3.5.1.14.2 Menu bar

The practical benefits may be questionable. But because Interface has this capability (for the small color resolutions), it has now been provided.

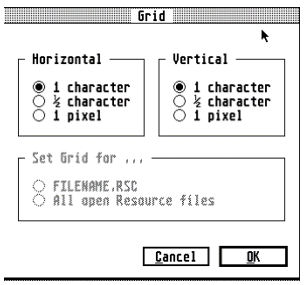
##### 3.5.1.14.3 Dialogs (in window)

To prevent other processes in multitasking operating systems such as Mag! X, MultiGEM or MultiTOS from being suspended in their screen outputs when Interface displays a Dialog box, this option places all Dialogs in windows. If a window is no longer available, Dialog boxes are displayed as normal Dialog boxes.

##### 3.5.1.14.4 object boxes (in window)

So you do not always have to call the object boxes again via 'O', you can put them in a window and then leave them on the desktop. Together with the right mouse button you can then drag objects out of the object boxes in the background. Under MuItiTOS, pressing the right mouse button is not necessary.

### 3.5.2 Grid…




When moving objects or changing the size of an object, Interface uses a grid in which the coordinates can be determined. The size of this grid can be set separately for the vertical and horizontal directions. The settings are 1 character, 1/2 character and 1 pixel. So that GEM can always display resources correctly, regardless of the screen resolution selected, the grid size for both directions should always be 1 character. If you do not use this setting, there is a risk that your program will display the Dialogs correctly in resolutions with a different system font (for example, less than 640 x 400). Only if you want to consciously use a pixel-precise positioning, you should turn them on. The XY grid for object locations is managed separately for each resource. When using the HRD format, the current grid is saved with the resource.

### 3.5.3 Use grid

With this option you can switch the set raster of all resources to pixel raster (acts like a switched raster). If you select again, the old grid settings will be restored. The meaning is a fast switching between letter and pixel grid.

### 3.5.4 Automatic sizing

If you change the text of a STRING object and the new text is shorter or longer than the old one, Interface normally adjusts the object size of the new text length. However, when entering shorter texts, it may sometimes be desirable to maintain the previous length. Therefore, with the size adjustment option, the adaptation can be switched off in the event that the new text length is shorter.

### 3.5.5 Fixed object numbers

With this option you can prevent object numbers from changing during actions. Instead, Interface only changes the concatenation of objects, both in Dialogs and in Menu trees. If this option is selected, Interface no longer changes the position of unsorted Menu tree entries.

When a new object is inserted into a tree, it is appended to the end of the tree and therefore does not change the object numbers of other objects in that tree. Sorting in trees is also possible. Only the object chaining is "relocated", all objects remain in their place.

When this switch is set, it should be easy to change resources of foreign programs without harming the program. It can only lead to errors when a program follows the object concatenation of its own resource. But that would almost never happen.

### 3.5.6 Save Preferences

This option saves all parameters described under **Options: Preferences....** Save parameters should be used if you have switched off the automatic saving of the parameters.

## 3.6 Saving the resource file

In order to be able to use your old resource files also under Interface, Interface supports all known file formats of other resource construction kits. The formats can be read as well as saved. To instruct Interface, to select a specific format when saving, select the resource file icon on the desktop and select the option **File: Info...** (see Info of a selected resource file). In the Dialog that appears, you can select the format. Interface automatically recognizes the format the next time it is loaded and saves the resource file again in the selected format.

### 3.6.1 Different resource formats

Below is a list of the different parameter file types and the programs that write these files.

DEF: this file format is used by the first Digital Research RCS.

RSD: corresponds to the DEF format. It was introduced by the Kuma RCS. Since Modula used files with the extension DEF, Kuma renamed the DEF format to RSD without further ado.

DFN: this format is used by the newer Digital Research RCS from version 2.0. Unfortunately, this format no longer saves for which languages header files should be output. This format is only important if you want to develop interface resource files for MSDOS programs because the only MSDOS RCS can only read DFN files.

HRD: this is the newest format; it was introduced by the WERCS RCS. It is the most space-efficient and stores most of the information (such as raster, aliases, etc.). It should always be preferred to the other formats if possible, since only in this format 16 characters long object names can be saved. All other formats only support 8 characters for the object names.

#### 3.6.1.1 DEF and RSD format

The DEF format is the resource format originally introduced by Digital Research.

| Offset | Length | Description |
|---|---|---|
| 0 | 2 | Number of records in the file (only in the first record not equal to 0) |
| 2 | 2 | Header output: 1 = C, 2 = Pascal, 4 = Modula, 8 = Basic, 16 = GFA<br>4 1 For objects: Index of the corresponding tree |
| 5 | 1 | Index of the tree or object |
| 6 | 1 | 1 if object, 0 if tree |
| 7 | 1 | type code (see below) |

| | | |
|---|---|---|
| 8 | 8 | Name of the object / tree filled with zero |

| Type | Codes |
|---|---|
| 0 | object in the tree or Menu |
| 1 | PANEL (replaced by interface by type 3) |
| 2 | Menu tree |
| 3 | Dialog tree |
| 4 | Alert-box |
| 5 | Free-string |
| 6 | Free-Image |

### 3.6.1.2 DFN format

The DFN format is essentially the same as the DEF format. The difference is that the high and low bytes of the entries are swapped and the bytes 3 and 4 (header output) omitted. This file format is compatible with Digital Research RCS on MSDOS machines.

### 3.6.1.3 HDR format

The Wercs format is split into an initial record and length-changing episode records. Note that the type definition is different than the other definition files.

**The Head of the HRD file**

| Offset | length | description |
|---|---|---|
| 0 | 2 | Version: Currently = 1 |
| 2 | 1 | Auto name: 1 if Wercs should automatically assign names (not used by interface, therefore always 0). |
| 3 | 1 | language flag: 1 = C, 2 = PASCAL, 4 = MODULA, 8 = BASIC, 16 = GFA, 128 = RSH output (in interface only) |
| 4 | 1 | Auto-Pos: 0 = Auto-Pos switched off, 1 = Half-Pos switched on, 2 = Auto-Pos switched on |
| 5 | 1 | Label Notation: 0 = big, 1 = normal, 2 = small |
| 6 | 1 | Auto Size: 0 = Auto Size Off, 1 = Auto Size On |
| 7 | 1 | Reserved |

**The next record**

| Offset | length | description |
|---|---|---|
| 0 | 1 | Type: see below |
| 1 | 1 | Reserved |
| 2 | 2 | tree index, i.e. number of the tree |
| 4 | 2 | object index |
| 6 | ? | Name of the object completed with 0 |

| type | codes |
|---|---|
| 0 | Dialog |
| 1 | Menu |
| 2 | Alertbox |
| 3 | Free Text - Free String |
| 4 | free bit blocks - Free Image |
| 5 | object |
| 6 | End of File |

7 If record is prefix (???)

### 3.6.1.4 The icon format ICO

Interface uses the ICO format when copying an icon to the GEM clipboard in the Icon Editor. The ICO format was introduced by Stefan Münch in his program 'IconEdi' and has nothing in common with the Windows or OS / 2 ICO format. We now quote the explanation of the format from the IconEdi manual:

IconEdi stores icons and images in a format that is based on common GEM structures, but is still very easy to handle and implement. Files of this format have a header that starts with an identifier (a “magic number”), gives information about version and header length, and points to an object block. From there it is further ‘pointered’. The pointer always refers to the beginning of the file, and the addresses are in Motorola format. The file extension should be 'ICO'.

The head looks like this (the notation is based on those used in the professional book):

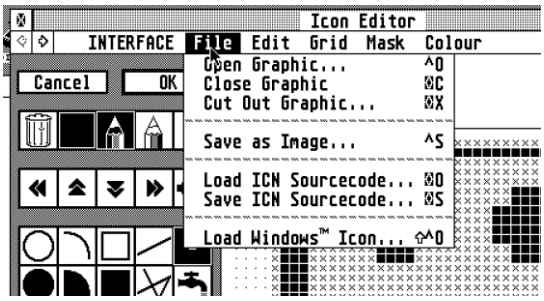
```
typedef struct {
        char icon_magic[4];       /*contains “ICON”; $49434F4E */
        int icon_version;         /* Version number, as in TOS */
        int icon_headerlength;    /* Length of the header in words, Standard: 5 */
        in *object_block          /* pointer to the object block */
} ICONHEADER
```

The decision as to which object contains the file is made on the basis of `ob_type` and `ob_spec` points to a corresponding structure. IconEdi only supports icons and images; In principle, however, every object can be saved in this format.

# 4 Icon Editor's Reference

## 4.1 The File Menu

### 4.1.1 Open Graphic...




The icon editor can load graphics in image format (monochrome and in color). From these graphics, you can then use **File: Cut Out Graphic...** to transfer parts of the graphic to the editor.

Interface opens a window with the graphic after loading and automatically selects the option Cut graphic.

Color images are displayed in the current palette or under TrueColor in the Resource palette, so you can see which colors the cut-out block will be used by the icon editor. Interface does not color-convert the XIMG graphic to the Icon Editor palette. So eventually you have to previously convert such graphics to the palette of the resource with a drawing program.

If you want to insert a block from a 256-color graphic into an icon with 16 colors, only the first 4 plans (16 colors) will be cut out and taken over into the icon editor (logical). As a result, all colors above color 15 are incorrectly adopted. After all, one has the advantage that all outlines of the desired graphic are already recognizable, and you just have to rework something.

### 4.1.2 Close Graphic

An image is kept in memory until it is removed from memory with close graphics. Then you can load another graphic with **File: Open Graphic**.

### 4.1.3 Cut Out Graphic...

With the option **File: Cut Out Graphic** you can mark an arbitrarily large area (however maximally as big as the icon) from a previously loaded image and transfer it to the icon editor.

If you select this option and no image has been loaded yet, a file selection will appear, allowing you to select the desired image file. If you select this option even though the window with the image was previously closed, it will reopen before cutting. As a reminder, closing the window does not remove the graphic from memory.

### 4.1.4 Save as Image...

With this option you can save icons as an image and, for example, edit it in a graphics program.

The icon editor uses the XIMG format, which is compatible with the monochrome image format, but also contains an RGB table. A description of the XIMG format is available e.g. in the book 'From beginner to GEM pro' by Dieter and Jürgen Geiß. The XIMG file also stores the Resource's RGB palette.

### 4.1.5 Load ICN Sourcecode...

The Load ICN source code... option allows you to load ICN format icon files, such as is stored by the RCS of ATARI.

### 4.1.6 Save ICN Sourcecode...

If you want to transfer the data of an icon directly into a C program (for example, for a new mouse form), you can save the graphic as C source code in ICN format. If the graphic is an icon, you will be asked in advance if you want to save the graphic or mask data.

In this way also color icons can be stored in the source code. The ICN format remains unchanged for compatibility reasons, only more graphic data is saved.

The number of used plans can be reached by:

```
planes = DATASIZE / ((ICON_W / 16) * ICON H);
```

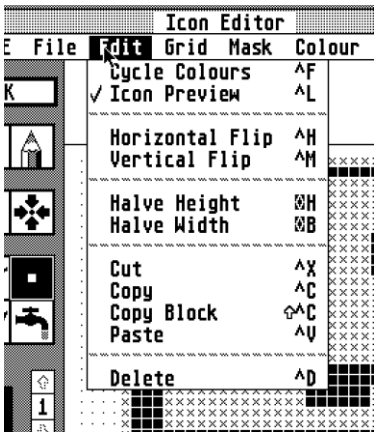
The graphic data is stored in device independent format. Before you can see the icon, you have to change it to "device-dependent" format with `vr_trnfm`. Of course, the mask still contains only 1 plane.

### 4.1.7 Load Windows™ Icon...

When editing color icons, you can use this option to load a Windows or OS/2 icon. Interface supports Windows icons with 2, 16 and 256 colors. Windows icons with 256 colors are only available as of Windows 3.1. OS/2 2.0 icons cannot be loaded yet, as the format seems to be incompatible with older OS/2 versions.

Windows and OS/2 icons are in many mailboxes and can certainly be obtained from public domain senders. As a result, you can already use a variety of color icons on the Atari.

## 4.2 The Edit Menu




### 4.2.1 Cycle Color

If the Color Scroll option is not selected and you click a black dot in the editor, it turns white and turns black the next time you click it. When color is selected, a black dot in the editor first turns gray, i.e. he is assigned to the mask. With the next click he turns white, then black, gray again ...

### 4.2.2 Icon Preview

If the canvas of the editor is too small for you, you can switch off the display of the preview with **Edit: Icon Preview**. The area claimed by the preview is then additionally used by the editor.

The setting of this option is saved with **Options: Save Parameters...**.

### 4.2.3 Horizontal Flip

Mirrors the graphic in the editor on the vertical axis, i.e. pixels that were previously visible on the top left are then visible on the top right.

### 4.2.4 Vertical Flip

Mirrors the graphic in the editor on the horizontal axis, i.e. pixels that were previously visible in the upper left are then visible in the lower left.

### 4.2.5 Cut

The icon is saved as a (color) image and in ICO format in the GEM clipboard and then deleted in the editor.

### 4.2.6 Copy

The icon is saved as a (color) image and in ICO format in the GEM clipboard.

### 4.2.7 Copy block

Part of the icon is saved as a (color) image and in ICO format in the GEM clipboard. After selecting the option, you must mark the area to be saved in the editor. The action can be canceled with the right mouse button.

### 4.2.8 Paste

The image from the clipboard is transferred to the editor. If the image to be pasted is smaller than the one currently being edited, you can place it with the mouse beforehand. In this case, the inserted graphic is drawn transparently (!) over the “old” graphic, i.e. white parts of the inserted graphic do not delete any points of the original graphic. If the image to be pasted is larger or the same size as the old image, it completely replaces the old image.

### 4.2.9 Delete

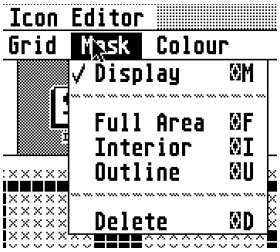
The drawing area as well as any existing mask is completely deleted.

## 4.3 The Grid Menu

The options of the grid Menu change the appearance of the icon in the drawing area.

- Large means that 8 x 8 pixels are drawn for each dot.
- Medium means that 6 x 6 pixels are drawn for each dot.
- Small means that 4 x 4 pixels are drawn for each dot.

## 4.4 The Mask Menu




You can only select the options of this Menu if you edit an icon since images do not have a mask. The mask is indicated by small crosses in the editor.

### 4.4.1 Display

The mask is only displayed if **Mask: Display** is selected. Only in this case, it can also be edited with the drawing functions.

### 4.4.2 Full area

The mask fills the entire area of the icon, i.e. all points of the mask are set.

### 4.4.3 Interior

The interior of an icon, i.e. all white areas completely enclosed by colored areas are filled with the mask.

Tip: For icons, the “color” white is synonymous with colorless, so that the background is visible. If you want to use white as a color, use the mask.

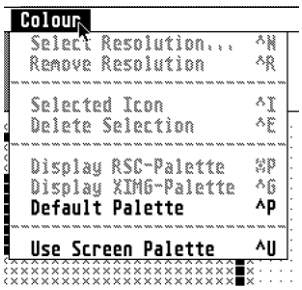
### 4.4.4 Outline

Each point of the graphic is enclosed by points of the mask.

### 4.4.5 Delete

The mask is deleted. Only under black dots remains the mask.

## 4.5 The Color Menu




Naturally, the options in the Color Menu are only available if you are editing a color icon.

A color icon is actually not a single icon, but consists of several icons for a variety of color numbers. So you can, for example, define a “Coloricon” that has only one bitplane, i.e. is monochrome. When loading a color icon resource, the operating system will look for the icon for each color icon that best suits the currently active resolution.

### 4.5.1 Select resolution...

Opens a Dialog in which you can select the color plane (“plane”) of a color icon that you want to edit. If the corresponding color level does not exist, it will be created.

### 4.5.2 Remove the resolution

Deletes the color plane displayed in the editor. If a selection screen exists for this resolution, it will also be deleted.

### 4.5.3 Selected icon

For icons with selection screen, you can switch between the icon and the selection screen using **Selected Icon**. If no selection screen exists in the selected resolution, it will be created.

### 4.5.4 Delete selection

The selection screen of the current resolution is deleted.

### 4.5.5 Display RSC-Palette

The system's current color palette is replaced by the resource's palette (which is stored with the resource). This will allow you to see which palette was used in a resource. Under TrueColor resolutions, this item is not selectable because Interface can always display the resource's palette there.

### 4.5.6 Display XIMG-Palette

The current color palette of the system is replaced by the palette of the currently loaded image (XIMG). This will allow you to see which palette was used in a color image.

### 4.5.7 Default Pallet

This option allows you to turn on the palette that interface found at program startup.

### 4.5.8 Use Screen Pallet

The system's current RGB palette is taken over (and stored with) the resource.

The problem with color icons is that you cannot guarantee that everyone will use the same color palette on their machine. Although the color values for the first 16 colors are more or less precisely defined, all overlying colors may differ from computer to computer (user to user). Of course, you cannot stop anyone from redefining the first 16 colors.

For a program to draw the color icons in the correct colors, it must of course know which color palette these icons were drawn with.

The supplied routines ('XRSRC') ensure that the drawn icons are drawn correctly in all resolutions in which the pixel is directly contained in the RGB value. These are all resolutions with 15 and more plans (> = 32768 colors). Of course, this can only work if you have included an RGB palette in the resource.

The RGB palette is an extension of the resource format of Interface. ATARI did not provide an RGB palette itself. The resource format remains compatible with the ATARI format and can still be easily read by the color icon AES (Falcon and MultiTOS). The RGB table is simply ignored.

Theoretically, the included color-code routines could of course set the RGB palette for the color icons in all resolutions, but we did not do so because we suspect that the user was thinking of modifying his palette. In addition, it would be horrible under multi-tasking operating systems, if each program would constantly switch the set color palette.

The first 16 colors should be defined as follows (see also the 'ATARI Profi book', 10th edition on page 284 or 'From beginner to GEM pro', 1st edition on page 76-77):

| No. | Color | Red | Green | Blue |
|---|---|---|---|---|
| 0 | white | 255 | 255 | 255 |
| 1 | black | 0 | 0 | 0 |
| 2 | red | 255 | 0 | 0 |
| 3 | green | 0 | 255 | 0 |
| 4 | Blue | 0 | 0 | 255 |
| 5 | cyan | 0 | 255 | 255 |
| 6 | yellow | 255 | 255 | 0 |
| 7 | Magenta | 255 | 0 | 255 |
| 8 | Light gray | 192 | 192 | 192 |
| 9 | Dark gray | 128 | 128 | 128 |
| 10 | Dark red | 182 | 0 | 0 |
| 11 | Dark green | 0 | 182 | 0 |
| 12 | Dark blue | 0 | 0 | 182 |
| 13 | Dark Cyan | 0, , | 182 | 182 |
| 14 | Dark yellow | 182 | 182 | 0 |
| 15 | Dark magenta | 182 | 0 | 182 |

These colors also largely correspond to the Windows and OS / 2 icon colors used. Only if these colors are set correctly can Interface display these icons in the correct colors.

You can set these colors in the control panel. The interface floppy disk also contains a new CPX module for XControl, which works on every TOS and on every graphics card, and with which the colors can also be set correctly.

# 5 Adjusting Interface

Interface has been programmed so that you can easily customize it to your own needs by modifying Interface's resource file. But make sure you make a backup of the resource file before such a change! If you change the interface resource file, *you can only change the text of objects*. *Never move individual objects and never change the size of the objects* with the mouse, as this may change the order of the objects and make Interface inaccessible to its resource.

Interface also provides reliable help for such changes: the option **Options: Fixed object numbers**. This Menu item prevents object numbers from being changed by your actions. Instead, Interface only changes the concatenation of objects, both in Dialogs and in Menu trees.

## 5.1 Keyboard customization

All keyboard calls of the interface Menus and the keyboard operation of Dialogs can be changed later. To do this, load the resource file from Interface and locate the tree in which you want to change the keyboard customization. In a keyboard-friendly Dialog object, there is a square bracket to the right in front of the corresponding letter. To assign a new key to this object, you must put this parenthesis in front of another letter. The letter is then underlined later. As you can see, only one key combination can be used, which corresponds to one letter of the object text. In addition, you should be careful not to mark a letter in two different objects, otherwise only one (the first) can be selected via the keyboard. If you want to change a shortcut in the Menus, you just have to write the new shortcut behind the corresponding Menu item. Again, an entry may occur only once, of course. You can detect multiple occurrences by testing the Menu (**Edit: Test object tree**), if you previously selected the Arrange Menu shortcuts in the **Preferences...** Dialog of the Options Menu.

## 5.2 Adaptation of file extension

### 5.2.1 Change of the Resource Format Extensions

Open the tree “RSCINFO” in the interface resource. There you will see the buttons RSD, DEF, DFN and HRD. If you prefer to save the RSD file with the extension XYZ, you simply have to change the text of the RSD button to XYZ.

### 5.2.2 Change of header file extensions

All extensions of the header files are in the interface resource as free strings with the names BASEXT, MODEXT1, MODEXT2, GFAEXT, PASEXT and CEXT (for Basic, Modula etc.). Just change the text of the corresponding free string into the new extension.

## 5.3 Changes for other languages

### 5.3.1 Changes for Basic

For the basic header file, the interface resources use the free strings VORTEXT, TXTZEILE and NACHTEXT. Wherever the placeholder “% s” is located in these free strings, the object name or the tree name will be used later. The main free strings and their meaning:

- VORTEXT: The first line of the header file. In the place of the placeholder, the Free String FRRSHEAD will be used later. The placeholder of FRRSHEAD then contains the name of the resource file.
- TXTZEILE: Assignment of the individual labels. 1st placeholder: Name of the label, 2nd placeholder: Number of the label, 3rd placeholder: Free String FRRSOBJ with the object type and the name of the tree.
- NACHTEXT: The last line of the header file The placeholders conform to the conventions of the C language. in VORTEXT, then you have to write the individual lines one after the other and separate them by the two characters ‘\n’ (line feed in C).

### 5.3.2 Changes for Lattice C

For all Lattice C fans, the interface resource contains the free string `OBSPEC_CAST` with the content (`LONG`) for the source code output. Since Lattice C cannot cope with (`LONG`), but must be obfuscated (VOID *) before `ob_spec`, it can be redefined by changing the interface resource.

### 5.3.3 Changes for Modula-2

Almost all strings of the Modula header output are now in the interface resource and can therefore be freely defined.

## 5.4 Change the Alerts

The default buttons and **Cancel** buttons of the Alert boxes are configurable. For example:

When exiting the program, Interface asks if the changed resource file should be saved. In this Alert, the exit button is always selected, i.e. if you hit return, the file will not be saved, but interface will quit immediately.

If this is too dangerous for you, load the interface resource and insert a dot as the first character before the save button of the Alert. As a result, Interface ignores the default setting, and takes this button as the default button.

If you want to operate the leave button in the Alert box with the undo key, then please insert a colon as the first character before the 'leave' text.

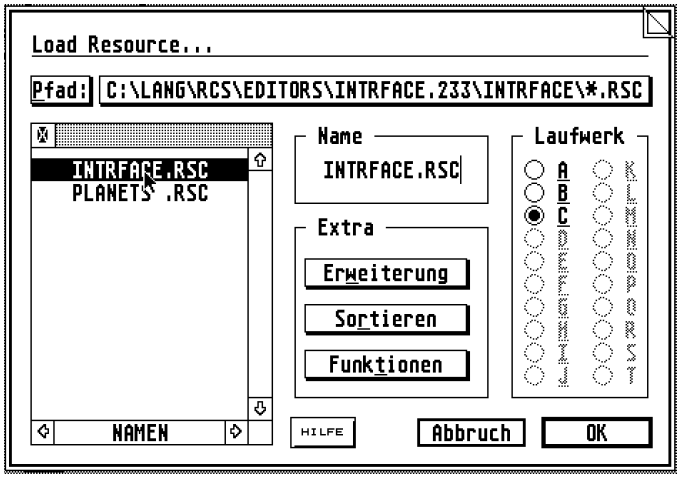
This configuration option is feasible with all Alert boxes.

Again briefly: '.' stands for the default button (is selected with Return) and ':' indicates the cancel button (is selected with Undo).

# 6 The Interface's File Selector

The file selector of Interface can be switched on or off with the option GEM file selector in the Dialog of the Menu **Options: Preferences...**. If the option is selected, Interface uses the file selector of the operating system or another alternative file selection such as 'Selectric' or the modular FSEL file manager of 'nol'.

Working on a computer - whether you like it or not - also has something to do with managing data, i.e. files. We have taken great pains to make the interface as comfortable as possible in this respect and have, for example, integrated a *document management system*. In addition, we have developed a new file selection that is very easy to use and has a number of special features.




If the Interface's file selector is active, the programs report all file functions using the box shown here.

As you can see from the illustration, the file selection offers, besides all elements of the GEM original, a number of other possibilities, even if some are still hidden behind Popup Menus.

The title bar of the file selection informs you not only which Dialog is currently active, but also for which option it was called. Below the title, the current access path (Pfad) appears, which you can edit in the usual way. However, there are some special features that will be discussed later.

## 6.1 The File Window

The center of the dialog box is the File pane, which displays the files of the selected floppy or hard disk drive. Since not all files on a drive can always be displayed within the window, there is a scroll bar on the right that allows you to scroll through the names to be displayed.

By the way, the first difference between the slider and the GEM file selection already exists. Interface remembers the position of the white slider field when leaving the file selection, so that the same files are displayed again the next time you call it up. This is especially interesting for hard disk owners who have a lot of files in one folder, as they don't have to "delve" through all the files every time. In addition, when moving the slider, the corresponding part of the file list is displayed immediately.

At the top of the file window you will find a "grey" bar that shows the selected file extension, as well as the close icon, which you also know from normal windows. By clicking on the bar, you can reload the file directory from the current drive. This is necessary if you have changed the diskette or entered a new access path using the keyboard without finally pressing Return. The Close symbol is used to close the displayed folder and to display the files from the next parent directory (if available).

Let us return to the names displayed in the window. There are two groups here, the files on the one hand and the folders on the other, the latter being identified by a small symbol in front of their name. Each group is sorted alphabetically.

## 6.2 Selecting files

As this is a file selection box, the individual entries can be selected by clicking with the mouse. Clicking on a folder causes it to be opened and its contents to be displayed in files and possibly other folders in the file window. To close a folder, click on the icon at the top of the window, as described above.

As soon as you click on a file name, however, it is highlighted in black and additionally transferred to the text field with the heading 'Name' to the right of the window. This field always shows the name of the file that would be loaded or saved if the OK button were selected. If you know exactly which file you want to load, you can simply enter its name in this field and then click **OK** (or press Return, which has the same effect).

Another quick way to select a file is to double-click the name of the file that is visible in the window. The name will then be accepted and the file selection will be exited automatically with **OK**.

The operating system of the ATARI computers not only manages the name of each saved file, but also its length, creation or modification date and time. Although these data are completely ignored by the standard

selection box, they can be displayed with Interface. For this purpose, there is a text field at the bottom of the window with arrows on both sides. When the file selection is called, there is always a name here, which means that the file names are displayed. If you click on the right arrow, the entries Length, Time and Date appear one after the other, the order is reversed with the left arrow. In addition, the corresponding information is of course displayed instead of the file names.

## 6.3 Access paths and masks

To be able to access tables of contents and files, the operating system needs a so-called access path. This path consists of the drive on which the files are located and possibly the names of the folders to be searched, separated by the '\' character. The drive letter must be followed by a colon.

For example, a possible path would be:

```
G:\Interface\DOCUMENT\
```

This path specification means that on drive 'G:' the contents of the folder `DOCUMENT` should be displayed, which in turn is located in the folder Interface.

Normally you change the path by clicking on the folder or by closing it again (via the close icon). However, as you can see from the image of the file selection box, the access path is displayed in an input field (Pfad) at the top of the box, so that you can also edit it using the keyboard. If your input requires more characters than currently visible, the input field will scroll to the left during input. If you have entered a new path, you still have to read in the corresponding directory by clicking on the title bar of the file window (see above), or by pressing Return.

In most cases the direct input of the path is not necessary, because all elements of the access path can be set with the mouse. This also applies to the current drive whose data is to be displayed. For this purpose, there are radio buttons on the right side of the file selection for the drives registered with the operating system, which can be activated by simply clicking on them. The drive letter will then be copied to the access path and the new directory will be read in automatically. A special feature is that the file selection of Interface remembers the paths of all drives and after switching from e.g. G: to A: and back to G: again displays the contents of the last folder set there.

An element of the access path has not been mentioned so far. If you have taken a close look at the image of the selection box, you may have noticed that the character string `*.RSC` appears at the end of the path displayed there. This is an access mask (sounds more complicated than it is) that determines which filenames are visible in the window. The folder names are basically not affected, i.e. they are always displayed.

As you probably know, file names always consist of two parts. The first part, with up to eight letters, is the actual name of the file. This is followed, separated by a period, by a name extension of a maximum of three characters, also known as a file identifier. This identifier is used to divide files into certain categories. In the above example, files with the extension 'RSC' are to be displayed, i.e. resource format.

But what does the asterisk in front of the dot mean? This is a placeholder that stands for a name of your choice. With the access mask `*.RSC` all files with the extension ‘RSC’ are displayed, regardless of their name. An example: the mask `'E*.RSC'` causes that of all files with the extension ‘RSC’ only those files are shown which start with 'E'.

There is another placeholder, the question mark '?', which can be substituted for any letter. So you can use the mask `*.P?C` to display only (graphic) files with the endings 'PAC', 'PIC', etc. A last (extreme) example: the specification `'C????.AC?'` would ensure that only files are displayed whose five characters long name begins with 'C' and which have an identifier that includes three characters and begins with 'AC', such as `'CLICK.ACX'` or `'CLOCK.ACC'`. What's new is that you can specify more than one extension, so with `'*.RSC,*.TXT'` you can select both resource documents and text files. The desired extensions must be separated by commas. Several extensions can also be selected with the mouse if the Shift key is pressed when selecting from the popup.

When a file function is called, a certain access mask is always specified. Nevertheless, it may be necessary to enter a different mask, e.g. if you want to restrict the display to certain names or want to access a different file format - with a different identifier - for certain functions. The input of a new access mask or an identifier can be done via the path text field (Pfad) and is activated after clicking on the selection window. Interface offers some simplifications, which will be explained below. As an attentive observer and reader, you will have noticed that no word has been lost about the representation of the input field for the access paths. Obviously they are Popup Menus. That's right. Interface remembers up to eight arbitrary access paths that can be selected from a

Popup Menu after you have clicked the Path: field. The corresponding directory is read and displayed immediately.

The first time you call the Popup Menu, you will notice that no entry has been made. This is up to you, after all you should enter your “preferred” paths here. To do this, double-click on the path text field to open the Popup Menu, which - because you have not yet entered a path - is absolutely empty. You can now select any line by clicking on it. The current path is then transferred to the selected position in the popup and can be selected from this point on. Proceed in the same way to change an entry. Below the field for entering the name is the actual core of the file selection. What is titled here with special, you will not want to miss in the future any more.

## 6.4 Extra Group

### 6.4.1 Erweiterung (Extension)

The Extension Popup Menu provides a selection of the most important file identifiers. The choice of one of the extensions adds this as extension to the path specification and sorts the file selection immediately again, so that you can save in most cases the “detour” over the text input. Several extensions can be selected with the mouse if the Shift key is pressed when selecting from this popup.

### 6.4.2 Sortieren (Sort)

The Sort (Sortieren) Popup Menu determines the order of the entries in the file selection. While the options Name, Date and Length hardly need an explanation, two sentences on Type and Unsorted. When sorting by file type, the files are sorted alphabetically by their ending. Files of the same ending are listed unsorted, i.e. as they were entered in the management list on the diskette or hard disk (rather unsorted).

### 6.4.3 Funkionen (Special functions)

This means that you know all the functions that are important for selecting files. However, the file selection of Interface offers a few additional options in the Functions Popup Menu that can make file handling much easier.

#### 6.4.3.1 Namen ändern (Change file name)


You can use the Change Name function from the Functions Popup Menu to change the name of a file. To do this, you must click on the desired file before calling the function in the popup, so that the file name is highlighted in black. The Dialog box shown below then appears, containing the previous file name as an editable default. If you exit the Dialog box with Ok, the name is changed accordingly and the file list is re-sorted. If the selected name already exists, a corresponding message is displayed.

#### 6.4.3.2 löschen (Delete files)

If you no longer need a file, you can remove it from the file directory using the Delete function from the Functions Popup Menu. To do this, you must click on the desired file before calling the function in the popup, so that the file name is highlighted in black. A confirmation prompt then appears in which you can confirm that you really want to delete the file. If you confirm this prompt, the file will be irretrievably deleted (or do you know a program that reliably restores deleted files?)!

#### 6.4.3.3 Kopieren (Copy files)

A very useful feature that you won't find in many other programs is copying files. This is done by selecting a file in the File pane and then selecting the Copy function from the Functions Popup Menu. Several Dialog boxes will then appear, informing you step by step about the actions you need to take.

If a file with the same name already exists on the target path, you will be asked if you want to overwrite this file, losing its previous contents.

> *Attention! Although overwriting may make sense, you should check carefully whether important data is not being deleted!*

#### 6.4.3.4 Neuer Ordener (Create folder)

You can use the New Folder function from the Functions Popup Menu to create a new folder in the directory currently displayed. The Dialog for entering names already shown above will appear. If you exit the Dialog with Ok, the folder is created and the file list is re-sorted. If the selected name has already been assigned to a folder, a corresponding message is displayed.

#### 6.4.3.5 Formatieren (Format diskettes)

The formatting of floppy disks is also possible from Interface. To do this, choose Format from the Functions Popup Menu.

In the Drive group, select the floppy disk drive in which the floppy disk to be formatted is located. The Format on both sides button indicates whether the disk should be formatted with approx. 360 or approx. 720 KB. Press Ok to start the option.

#### 6.4.3.6 Show drive information

Last but not least, you can use the Disk Info function from the Functions Popup Menu to display some data about the current drive and path.

In order to find the path settings every time, you start the program, you should go to **Options: Save Parameters...** save the parameters to which the paths belong.

## 6.5 keyboard control

And like other Dialogs in Interface, the file selection can also be completely controlled via the keyboard. For better keyboard control the 'Control Mode' was introduced. When the control key is pressed at the same time, some keys get a new/different function:

| | |
|---|---|
| Cursor up/down | selects the first entry in the file list and then allows the selection of all other folders and files. If necessary, the file is scrolled automatically. |
| Shift Cursor up/down | scrolls the file list “page by page”. |
| Shift Cursor left/right | changes the display in the file list from name, length, time and date to the next option. |
| Clr-Home | shows the beginning of the file list |
| Shift Clr-Home | shows the end of the file list |
| Return | selects a file or opens a folder (if selected). |
| Backspace | switches to the parent folder. |
| Spacebar | Rereads the directory. |
| Letters | Selects a file directly. If, for example, the letter E is pressed, the first file beginning with E is selected. The next character input further restricts the selection, e.g. to EN... The current mask is displayed in the header of the file list. |
| Shift Letters | Works like the input of letters, but on the extensions. |
| Esc | Deletes the selection by entering letters. |
| Tab S | witches permanently to control mode. The word 'CONTROL' is displayed at the top right of the Dialog. Another press on Tab switches the control mode off again. |

# 7 Creating an EXTOBFIX program

## 7.1 Overview

Interface provides numerous features in its Dialogs and Menus that are not supported by the operating system. It uses the MyDials supplied with Interface, which you can also integrate into your C and Basic programs. In order for Interface to support other libraries when testing a Menu or Dialog, the EXTOBFIX program has been introduced. For a better understanding, it is recommended to first deal with the MyDials.

What is EXTOBFIX?

Interface tries to load the program `'EXTOBFIX.PRG'` directly after the program start in its own directory. If this program could be loaded, Interface replaces parts of the tree test routines with routines of the EXTOBFIX program. The EXTOBFIX can replace the customization of G_USERDEF objects as well as the complete Dialog test routine and the Alert test routine. If you use your own object types in your program (such as Checkboxes, Radiobuttons, etc.), you can draw these objects while testing your own routines, which must be in an `EXTOBFIX.PRG`.

## 7.2 Basic structure

Please look at the C-source code `EXTOBFIX.C` in the folder `MYDIAL.EXT` of the interface-original floppy disk. The EXTOBFIX can of course also be written in other programming languages, but in C it is easiest.

### 7.2.1 The main routine

The main routine does nothing else but issue a warning message and quit the program if it was not started by Interface (see `main()`). Since the EXTOBFIX must be kept in memory until you exit Interface, the main routine is never jumped by Interface, because the EXTOBFIX would otherwise be removed from memory.

### 7.2.2 The initialization

Interface searches in the EXTOBFIX for the string '06101964' or '06101965' (8 bytes “Magic”). Immediately behind it expects a pointer to the initialization routine of EXTOBFIX.

Interface then jumps to the initialization routine and gives it a value in the register D0. If this value is not 0 then the EXTOBFIX should initialize. If this value is 0, the EXTOBFIX should de-initialize. In the event of an error or after de-initialization, the EXTOBFIX must return the long value 0L in the register D0, and a pointer to a pointer array if the initialization is successful.

### 7.2.3 The pointer array

In this array are three pointers to routines of the EXTOBFIX program. The first pointer points to the `fix_objs()` routine, which normally adapts the G_USERDEF objects and provides pointers to the EXTOBFIX drawing routines. The second pointer points to the Dialog test routine. This routine replaces the complete interface test routine of the Dialog boxes. The third pointer points to the `alertbox` test routine. If the EXTOBFIX does not provide one of these routines, the pointer must have the value 0 (see EXTOBFIX for the Geiss routines).

### 7.2.4 The fix_objs routine

All coordinates of a tree have already been converted by Interface into the device-dependent format. In addition, a copy of all OBJECT structures, all TEDINFO structures and all `te_ptext` strings of the EDITABLE G_FTEXT and G_FBOXTEXT TEDINFO structures was created. This means that the EXTOBFIX test routine is allowed to modify this data as it will be restored by Interface again after the test is finished. Transfer parameters in the CPU registers:

- A0: Pointer to the first object of the tree.
- D0: If the word in D0 is not 0, then the tree is a Dialog box, otherwise a Pulldown Menu. The `fix_objs` routine must now search the entire tree for G_USERDEF objects and, if necessary, enter a pointer to a USERBLK structure in the `ob_spec` field. The USERBLK structure must contain in `ub_code` a pointer to its own EXTOBFIX drawing routine for this object. In the parameter `ub_parm` a copy of the old `ob_spec` value should be stored.

### 7.2.5 The routine for testing Dialogs

This routine must adapt all objects of the tree (see `fix_objs`), center the Dialog box, draw and execute. After the execution, the Dialog box must also be correctly removed again (either restore background, or send redraw message through `form_dial(FMD_FINISH)`).

Parameters:

- A0: Pointer to the first object of the tree
- D0: Number of the default edit object (integer) is always 0.
- D1: pointer to the name search routine of Interface (Long)
- A1: Pointer to the WINDOW structure of the tree window. This pointer must be passed to the interface name lookup routine.

If you want to determine the name of an object in this routine, then you must call the interface name search routine. This routine needs the following transfer parameters:

- A0: Pointer to the WINDOW structure of the tree window.
- D0: object number.
- A1: Pointer to a text buffer of at least 18 characters.

The routine writes the name of the object with 0 terminated in the text buffer. If no name could be found, the text buffer stores a zero byte as the first character.

### 7.2.6 The routine for testing Alerts

This routine should draw and execute an Alert box.

Parameters:

- D0: always 1 (integer). Should be the default exit button of the Alert box.
- A0: Pointer to the alert string. The number of the Alert icon can take values from 0-Z (36 different characters).

### 7.2.7 A separate help page

Interface supports a help page in every EXTOBFIX file. This help page can be displayed in Interface in any object Dialog. The pointer to the tree of the help page must be defined last in the function array (see new MyDial EXTOBFIX file). EXTOBFIX files with help page get the Magic '06101965' instead of '06101964'.

The tree is adapted by Interface to the extended MyDial object types. The first two objects (FlyDial corner and heading) must always have the object numbers 1 and 2. All other objects are freely definable, the number of objects and the size of the Dialog is also not specified.

If an EXTOBFIX file with help page has been loaded, Interface will always display this page when you press Help. Otherwise, a default MyDial help page (nice word construction ...) will be displayed.

# 8 Interface “Remote Controlled”

Interface was equipped with an accessory interface. This allows accessories to access all internal data from Interface and provide their own functions. Other functions, such as Extended drawing functions in the icon editor can also be realized via the Accessory pipeline. Since Interface is based on routines of the book 'From beginner to GEM professional' (2nd edition) from the Hüthig Verlag, reading this book is a prerequisite for anyone who wants to write a program that accesses via the Accessory interface routines of Interface. Since some changes were made in the second edition, the first edition of the book is not enough.

A detailed explanation of all routines would go beyond the scope of this manual. So far, to our knowledge, no one has bothered to extend Interface on this path. If you require explanations about the Accessory interface, please contact **nol Software** in writing or via the MausNet via modem directly to the author of Interface: Olaf Meisiek @ FL

# 9 A resource program as an example

In this section, a small program in C will show you how to load a resource file and display a Dialog box from it. The used resource ('DEMO') has already been described several times in the manual.

```
/*****************************************************************/
/*                                                               */
/* Modul: DEMO.C                                                 */
/* Datum: 23.02.93                                               */
/*                                                               */
/*****************************************************************/
#include <string.h>
#include <stdio.h>
#include <portab.h>
#include <vdi.h>
#include <aes.h>
#include "demo.h"

/* VARIABLES */
WORD gl_apid;           /* Application number */
OBJECT* planet;         /* Pointer to the Tree of the Planet Dialog Box */

/* FUNCTIONS */
BOOLEAN init_global(VOID); BOOLEAN term global(VOID);

main()
{
    WORD x, y, w, h;
    WORD button;
    BYTE alerttxt[400], ort[10];
    /* Register application and load resource */
    if (init_global() == FALSE)
        return (0);                         /* Error, terminate program immediately */
    form_center(planet, &x, &y, &w, &h);    /* Center Dialog */
    /* Reserve Screen Memory  */
    form_dial(FMD_START, 0, 0, 0, 0, x, y, w, h);
    objc_draw(planet, 0, 8, x, y, w, h);    /* Draw Dialog box */
    button - form_do(planet, 0) & 0x7fff;   /* Execute Dialog */
    if (button == PLOK)                     /* evaluate Dialog box with OK */
    {
        if (planet[PLVENUS].ob_state & SELECTED)
            strcpy(ort, "Venus");
        if (planet[PLERDE].ob_state & SELECTED)
            strcpy(ort, "Erde");
        if (planet[PLMARS].ob_state & SELECTED)
            strcpy(ort, "Mars");
        sprintf(alerttxt,
            "[2][Sie kommen wirklich|vom Planeten %s?][ Nein | Ja]", ort);
        if (form_alert(2, alerttxt) == 2)
            form_alert(1, "[1][ |Faszinierend!][ Bye ]");
        else
            form_alert(1, "[3]( |Schade...][ Ende ]");
    }
    /* Share screen memory, create redraw */
    form_dial(FMD_FINISH, 0, 0, 0, 0, x, y, w, h);
    term_global();          /* Finish application */
    return (0);             /* Exit program */
}

/*****************************************************************/
/* Initialisieren des Moduls                                     */
/*****************************************************************/
BOOLEAN init_global()
{
    WORD i;
    i = appl_init(); /* Get application number */
    gl_apid = i;
    if (gl_apid < 0)
        return (FALSE);
    graf_mouse(BUSY_BEE, NULL); /* Show busy bee */
    if (!rsrc_load("DEMO.RSC"))
    {
        graf_mouse(ARROW, NULL); /* Show arrow */
        form_alert(1, "[3][Resource-File|DEMO.RSC?][ EXIT ]");
        return (FALSE);
```

```
    } /* if */
    graf_mouse(ARROW, NULL); /* Show arrow */
    rsrc_gaddr(R_TREE, PLANET, &planet); /* Dialog box's  addr */
    return (TRUE);
} /* init global */

/*************************************************************/
/* Terminieren des Moduls                                    */
/*************************************************************/
BOOLEAN term_global()
{
    if (gl_apid >= 0)
    {
        rsrc_free(); /* Release resources */
        appl_exit(); /* Finish application*/
    } /* if */
    return (TRUE); /* term_global */
} /* term_global */
```

# 10 The MyDial Library

*MyDials have evolved since the Interface manual was written, therefore this chapter uses the MyDials information provided on the Interface 2.33 diskettes (MYDIAL.TXT in folder MYDIAL_C).*[5]

## Foreword

This is now my second attempt to document the MyDials. The first time I had little time and even less desire, and that affected then also on the first "guidance", which actually did not deserve her name. I hope that this time I will be able to do it better. Since I have programmed most of the MyDials myself, there is of course a danger that I won't explain some things, because they just seem natural to me.

If anything should be unclear or you have problems with the library, then you can reach me any time over the company Team Computer. If you have a modem, you can send your questions and suggestions directly to me via your mailbox. My address is "Olaf_Meisiek@fl.maus.de".

## 10.1 Introduction

### 10.1.1 What are the MyDials?

The MyDials are a collection of routines, which allow 'Flying Dialogs' that can be operated via keyboard in a simple way. They offer real Radiobuttons (small round buttons), Checkboxes, Popup Menus, comfortable input routines in edit fields, moveable Dialogs, keyboard buttons, support for HELP and UNDO keys, special character input and much more. The MyDials are not an integral part of the Interface program package, but an addition, for whose completeness and usability TEAM assumes no liability.

### 10.1.2 Using MyDials in your own programs

The prerequisite for using MyDials in your own programs is that you have an original version of Interface. The use of MyDials without an original version of Interface is not permitted.

### 10.1.3 Who can use the MyDials?

Anyone who owns Turbo C, Pure C or a linker that can link Turbo C or Pure C libraries. Unfortunately, the Pure C libraries are not Digital Research compatible, i.e. with a few exceptions you can only use the MyDials in combination with Pure C or Turbo C.

The use of MyDials under GFA and Omikron Basic is now also possible. The diskette contains the necessary routines. Since I'm not familiar with Basic, I can't answer specific questions about integration myself. Questions should therefore be sent to Shift so that they can be passed on to the author of the respective integration.

For Modula 2 there is a program that can link Pure C libraries to Modula 2 programs.

### 10.1.4 MyDial files and their meaning

#### 10.1.4.1 NKCC folder:

NKCC is a library by Harald Siegmund that is used by the MyDials. This library makes it possible that every key combination can be recognized and evaluated correctly ***independent of country***. `NKCC_TOS.O` must be linked with the MyDials. A manual for NKCC can be found in the file `NKCC.DOC`.

Another warning: For the sake of completeness the folder also contains the `NKCC.O` Library. This library should not be used under any circumstances, because this library does not run correctly under MultiTOS or Mag!X. This warning only applies to the version supplied on the Interface disk. It is quite possible that there is a newer version by Harald Siegmund (the author of NKCC) which no longer causes these problems. The MyDials only need `NKCC_TOS.O` in any case.

#### 10.1.4.2 MYDIALG.LIB:

This file contains all MyDial routines. It also contains some routines from the module GLOBAL from the book "From Beginner to GEM-Profi", which need the MyDials.

#### 10.1.4.3 MYDIAL.LIB:

This file no longer exists. Please use MYDIALG.LIB now. So there should be no more naming conflicts with routines from the book "Vom Anfänger zum GEM-Profi".

[5] The rest of this chapter is taken directly from the MYDIAL.TXT file on the diskette. It is supposed to contain the latest information about MyDial (DrCoolZic).

#### 10.1.4.4 MYDIAL.H

The MYDIAL.H File contains declaration for all the MyDial routines.

#### 10.1.4.5 PORTAB.H:

Please include this header file **before** the MyDial header file. PORTAB.H contains all definitions important for MyDials. More details about this file can be found in the book “Vom Anfänger zum GEM-Professional”. If you already own PORTAB.H, you don't need to replace it with this version.

### 10.1.5 How do MyDials work?

So that the MyDials can move Dialogs and edit keyboard objects, they have to do some preparatory work. The MyDials check the object type and provide their own routines for special types of objects. As a rule, these objects are so-called “extended” object types.

With Interface you can easily create such an object. If you edit an object in Interface, you can see the field “Extended type” in the upper right corner of the Dialog. There you simply enter a number to tell the MyDials later that this object should be treated specially.

For each type of object there is a different number that you can enter. Sometimes you have to use a certain “normal” object type in addition to the number, or set certain flags. In the interface resource you will find many examples of such extended objects.

### 10.1.6 What extended object types are there?

I will now enumerate all the extended object types supported by MyDials to list: the types, the extended type, the normal object type (if necessary), the required object flags and states, and the flags that must not be set under any circumstances are specified. All flags that do not fall under either one or the other category may be set additionally if required.

#### 10.1.6.1 Displacement Object

Extended type: 17
Flags set: TOUCHEXIT
Note: If you click and drag any object with the extended type 17 with the mouse, you can move the whole Dialog.

#### 10.1.6.2 FlyDial Corner

Normal type: IBOX
Extended type: 17
Legal flags: OUTLINED, CROSSED
Note: This object corresponds to the Move object in the function. However, it is drawn by the MyDials as a “dog's ear”.

#### 10.1.6.3 Radiobutton

Normal type: BUTTON or STRING
Extended Type: 18
Flags set: RBUTTON (Radiobutton)
Note: A radio button of this type is a round button with the text next to it. Since the button is relatively small, you can also select it by clicking on the text next to it.

#### 10.1.6.4 Checkbox

Normal type: BUTTON
Extended Type: 18
Flags set: -
Unlawful flags: RBUTTON, EXIT, SHADOWED
Note: A Checkbox looks almost like a Radiobutton. Instead of the round button a square box is drawn. If the SELECTED flag is set, a checkbox is represented by a cross. If the CHECKED flag is set, but SELECTED is not set, the checkbox is filled with a dot pattern. From these 3 states you can easily program a “TriState” button, which is used e.g. by Interface to set flags in several objects simultaneously (Menu item “Set flags...”).

#### 10.1.6.5 Exit button

Normal type: BUTTON
Extended Type: 18
Flags set: EXIT
Non-imposed flags: RBUTTON
Note: An exit button corresponds to a normal button. The one big difference is that it can be operated by the keyboard. If you want to use the flag TOUCHEXIT for this button, you also have to set EXIT, because otherwise the button would be drawn as a Checkbox.

#### 10.1.6.6 String

Normal type: STRING, TEXT, BOXTEXT
Extended Type: 18
Non-imposed flags: RBUTTON, EXIT
Note: This is a keyboard text. You could use it e.g. for the text in front of a popup button to trigger the popup. The object BOXTEXT could also be used directly for the popup button.

#### 10.1.6.7 Undobutton

Extended Type: 18
Flags set: `ob_flags` 11
Note: If `ob_flags` 11 was set in any extended object of type 18, this object can also be selected using the UNDO key. Interface uses this e.g. for all “Cancel” buttons.

#### 10.1.6.8 Underlined

Normal type: STRING or BUTTON
Extended Type: 19
Note: Such an object is represented by the MyDials as an underlined string. Interface uses this object type for all Dialog headings. The underscore has the length of the object width.

#### 10.1.6.9 Titlebox

Normal type: BUTTON
Extended type: 20
Note: A title box is drawn as a large frame with the text of the object in the upper left corner as a heading. Box titles are used by Interface, for example, in the Resource Info Dialog.

#### 10.1.6.10 Help Button

Extended Type: 21
Note: All objects of the extended type 21 can be identified by the HELP-Select key. The type and flags of the object do not matter, but it has become common practice to display a help button as box text with the flags EXIT, OUTLINED and SHADOWED. The text should be lowercase and centered. Help buttons are now available in Interface in every object Dialog box.

#### 10.1.6.11 Circle button

Normal type: BOXCHAR
Extended type: 22
Note: A circle button is usually always used together with a Popup. The button is displayed as a small box with a circular arrow inside. With this object you can quickly call up the next popup entry without having to select the popup first. Circlebuttons are used by Interface e.g. in the BoxText Dialog box.

#### 10.1.6.12 Popup object

Extended Type: 23
Note: All selectable objects in a popup should contain the extended Type 23 so that they are drawn correctly in 3D mode.

#### 10.1.6.13 Keyboard operable Popup string

Normal type: STRING
Extended type: 24
Note: See “Popup Object”.

#### 10.1.6.14 3D slider objects

Extended type: 25
Note: All objects of a slider should have type 25 so that they are drawn correctly in 3D mode.

#### 10.1.6.15 Scrollable input fields

Normal type: FTEXT or FBOXTEXT
Extended Type: 26
Note: A scrollable input field can be up to 255 characters long. The field behaves like a normal input field. In contrast to multiline input fields, the administration is completely taken over by the MyDials.

#### 10.1.6.16 Undo button

Extended type: 31
Note: An Undo button corresponds to the Help button (type 21). Of course the UNDO key is used to select the object.

#### 10.1.6.17 Multiline input fields

Normal type: FTEXT or FBOXTEXT
Extended type: 0 (zero)
Flags set: EDITABLE, `ob_flags` 10
Note: This object type already existed in the interface 1.0 MyDials, but it was not documented by me, because it is a bit more complicated to handle. Multi-line input fields are used by Interface e.g. in the FreeString Dialog box. Several successive input fields behave as if they were a single field.

## 10.2 Programming with the MyDials

### 10.2.1 Program Flow chart

At program start:

- - Initialization of NKCC
- - Initialization of the MyDials
- - Initialization of the program resource with the MyDials

Arbitrary use of the MyDial routines

At the end of the program:

- Deinitialization of MyDials
- Deinitialization of NKCC

### 10.2.2 An example program

The following sample program should give you an impression of how the MyDials are integrated into a main program. NKCC and the MyDials are initialized. The MyDials open internally a workstation, so that they do not change the parameters of the main program. In the old MyDial version NKCC had to pass a VDI workstation handle. This is no longer necessary, since you should only use the `NKCC_TOS.LIB`, which does not contain VDI and AES functions.

```
#include <portab.h>
#include <aes.h>
#include <vdi.h>

#include <nkcc.h>
#include <mglobal.h>
#include <mydial.h>

OBBLK usr_block[100];

main()
{
    /* NKCC and the MyDials are initialized */
    nkc_init(NKI_NO200HZ, 0); /* Initialize NKCC */
    nkc_set(0);

    if (dial_init(Malloc, Mfree, NULL, NULL, NULL, 0, usr_block, 100) == TRUE)
    {
        /* The main program follows */

        dial_alert(NULL, "[0][Initialization|has worked!][[OK]", 1, 1, ALI_LEFT);
```

```
        /* That end the main program... */

        dial_exit(); /* deinitialize MyDials */
    }

    nkc_exit(); /* deinitialize NKCC */

    return (0);
}
```

The Pure C project file could look like this for the short program:

```
                ; MYDEMO.PRJ
                ; ----------
mydemo.prg      ; Name
=               ; list of modules follows...
pcstart.o       ; startup code
mydemo          ; Hauptprogramm
mydialg.lib     ; Mydial lib
pcstdlib.lib    ; standard lib
pctoslib.lib    ; TOS lib
pcgemlib.lib    ; AES and VDI lib
nkcc_tos.o      ; NKCC lib
```

## 10.2.3 Description of the individual MyDial routines

### 10.2.3.1 9.2.3.1 Initialization routines

#### 10.2.3.1.1 dial_init

```
GLOBAL BOOLEAN dial_init(VOID *alc, VOID *fr, WORD *mnum,
                         MFORM **mform, BOOLEAN *gs, BOOLEAN do3D,
                         OBBLK *block, WORD blocklen);
```

- `alc` = pointer to a memory reservation routine (e.g. Malloc)
- `fr` = Pointer to a routine that releases the reserved memory (e.g. Mfree).
- `mnum` = Pointer to an integer variable containing the last mouse type used (e.g. ARROW). The value is needed so that the MyDials can always restore the correct mouse shape. The mouse types are passed to the AES command "`graf_mouse`". A list of all possible mouse shapes can be found e.g. in the professional book in the explanation of the AES command "`graf_mouse`".
- `mform` = Pointer to a pointer pointing to a self-defined mouse shape. This pointer is only needed if *`mnum` contains the value 255 (USER_DEF).
- `gs` = If this variable contains the value TRUE, the MyDials will draw Grow/Shrinkboxes, otherwise not.
- `do3d` = If TRUE, all objects for which the DRAW3D algorithm was set are drawn in resolutions of at least 16 colors with a 3D effect.
- `block` = A pointer to a memory area that MyDials can use for `userdefs`. To reserve space for 1000 `userdefs`, you should use a global variable defined with "OBBLK `usr_block[1000];`".
- `blocklen` = number of `userblock` structures that can be placed in "block".

You don't need to specify all values. If your program should not contain these variables, you can also pass NULL pointers. The shortest possible "`dial_init`" call looks like this:

```
ret = dial_init (Malloc, Mfree, NULL, NULL, NULL, 0, usr_block, 100);
```

If "`dial_init`" returns the value FALSE, the initialization did not work. The program should then be terminated with an error message. The initialization can only fail if there is not enough memory.

#### 10.2.3.1.2 dial_exit

```
GLOBAL BOOLEAN dial_exit (VOID);
```

`dial_exit` deinitializes the MyDials. If the MyDials could be initialized at the program start, `dial_exit` must be called at the program end, so that the MyDials can close e.g. their own workstation.

#### 10.2.3.1.3 dial_fix

```
GLOBAL VOID dial_fix (OBJECT *tree, BOOLEAN is_Dialog);
```

- tree = address of the tree
- is_Dialog = If TRUE, then it's a Dialog box, if FALSE, then it's a Menu tree

dial_fix must be called once at the program start of the main program after the initialization of the MyDials for each tree of the main program. The MyDials initialize the extended object types in dial_fix and enter their drawing routines in them. Furthermore, dial_fix adapts the values ob_width and ob_height of the OBJECT structure of all icons and images to the valid resolution, because the AES makes errors at rsrc_load.

For each user-defined object the MyDials create the structure OBBLK (definition in MYDIAL.H). This structure contains the USER-BLK-structure and the former object type, so that the old object type can also be experienced after the adaptation.

### 10.2.3.2 The Dialog routines

If you want to use the Dialog routines, you must reserve the screen area beforehand and release it again after the Dialog has disappeared. It looks like this:

```
wind_update (BEG_UPDATE); /* GEM Log in screen output */
/* Dialog editing */
wind_update (END_UPDATE); /* Screen for other processes */
/* release again */
```

Most programmers call wind_update directly after the main evnt_multi, and can save themselves a second call before the Dialog routines of course. If you forget to use wind_update, all mouse clicks will be passed on to other applications and select e.g. objects on the desktop, even though you had only clicked on the shift corner of the Dialog.

#### 10.2.3.2.1 HndlDial

```
GLOBAL WORD HndlDial (OBJECT *tree, WORD def, BOOLEAN grow_shrink,
                      RECT *size, BOOLEAN *ok);
```

- tree = address of the Dialog tree
- def = Default edit object, usually ROOT (ROOT = 0)
- grow_shrink = Draw on TRUE Grow/Shrinkbox, do not draw on FALSE
- size = starting position of the Grow/Shrinkbox, at NULL the center of the screen is taken
- ok = With TRUE the background could be restored. With FALSE the MyDials execute another form_dial (FMD_FINISH, ...), so that a Redraw-Message is sent. If you nest two Dialog boxes (i.e. call a second one from one box), you must redraw the first box if ok == FALSE because the background of the second box could not be restored due to lack of memory.
- Return value: Index of the object that left the Dialog.

HndlDial executes a complete Dialog processing. The Dialog is drawn, executed and removed from the screen again. Internally HndlDial uses the routines open_dial, dial_draw, dial_do, close_dial (in this order).

#### 10.2.3.2.2 open_dial

```
GLOBAL BOOLEAN open_dial (OBJECT *tree, BOOLEAN grow, RECT *size,
                          DIALINFO *dialinfo);
```

- tree = address of the Dialog tree
- grow = Draw with TRUE Grow/Shrinkbox, do not draw with FALSE
- size = starting position of the Grow/Shrinkbox, at NULL the center of the screen is taken
- dialinfo = open_dial remembers in the DIALINFO structure the size of the Dialog box including all attributes and the memory area in which the background of the Dialog box was saved. The DIALINFO structure is needed by close_dial to restore the screen. DIALINFO is declared in MYDIAL.H.
- Return value = If TRUE, the background of the Dialog box could not be saved. With FALSE the MyDials execute a form_dial (FMD_FINISH,...) in the function close_dial to send a redraw message. If you nest two Dialog boxes (i.e. call a second one from one box), you must redraw the first box if ok == FALSE after close_dial because the background of the second box could not be restored due to lack of memory.

open_dial centers the Dialog, draws a Grow/Shrinkbox and saves the desktop area over which the Dialog should be drawn. In addition, the DIALINFO structure is initialized with the corresponding values. open_dial does not draw the Dialog. You must then do this yourself with dial_draw or objc_draw.

The Dialog is only centered by open_dial if both the X and Y position of the Dialog is 0 (ob_x and ob_y = 0). This means that the Dialog appears at the last position to which it was moved in all subsequent calls. If you want to force a new centering, you only have to reset the X- and Y-position of the ROOT object to zero.

#### 10.2.3.2.3 close_dial

```
GLOBAL BOOLEAN close_dial (BOOLEAN shrink, RECT *size,
                           DIALINFO *dialinfo);
```

- shrink = Draw with TRUE Grow/Shrinkbox, do not draw with FALSE
- size = end position of the Grow/Shrinkbox, at NULL the center of the screen is taken
- dialinfo = This structure was initialized by open_dial. close_dial finds the size of the Dialog box and the address of the background buffer.
- Return value = close_dial always returns TRUE.

close_dial draws a Grow/Shrinkbox and restores the screen. If the wallpaper in open_dial could not be saved due to lack of memory, close_dial sends a redraw message (FMD_FINISH) to the AES.

#### 10.2.3.2.4 dial_center

```
GLOBAL VOID dial_center (OBJECT *tree);
```

- tree = address of the Dialog tree

dial_center initializes the Dialog box in such a way that it is drawn exactly in the middle of the screen. Since dial_center is executed internally by HndlDial or open_dial, you don't usually need this function (unless you do not use HndlDial/open_dial).

#### 10.2.3.2.5 dial_start

```
GLOBAL BOOLEAN dial_start (OBJECT *tree, DIALINFO *dialinfo);
```

- tree = address of the Dialog tree
- dialinfo = dial_start remembers in the DIALINFO structure the Dialog box size including all attributes and the memory area in which the Dialog box background was saved. The DIALINFO structure is needed by close_dial to restore the screen.

DIALINFO is declared in MYDIAL.H.

- Return value = If TRUE, the background of the Dialog box could not be saved. With FALSE the MyDials still execute a form_dial (FMD_FINISH,...) in close_dial to send a redraw message. If you nest two Dialog boxes (i.e. call a second one from one box), you must redraw the first box if ok == FALSE after close_dial because the background of the second box could not be restored due to lack of memory.

dial_start initializes the DIALINFO structure and buffers the screen area over which the Dialog box is to be drawn. Before dial_start you have to call dial_center or a custom function to save the character position in the Dialog tree. dial_start is called by open_dial. If you use HndlDial or open_dial, you don't have to use dial_start.

#### 10.2.3.2.6 dial_draw

```
GLOBAL VOID dial_draw (DIALINFO *di);
```

- di = DIALINFO structure initialized by open_dial or dial_start.

dial_draw draws the Dialog. dial_draw executes only one objc_draw internally, whereby objc_draw passes the values of the DIALINFO structure. You can also use objc_draw instead of dial_draw, but then you have to pass more parameters.

#### 10.2.3.2.7 dial_do

```
GLOBAL WORD dial_do (DIALINFO *di, WORD edit_obj);
```

- di = DIALINFO structure initialized by open_dial or dial_start.
- edit_obj = Default edit object, usually ROOT (ROOT = 0)
- Return value = index of the object that left the Dialog.

dial_do corresponds to form_dial as far as possible, but is much more comfortable. dial_do automatically calls the move function if the FlyDial corner has been clicked and evaluates all shortcuts. You can position the cursor with the mouse, fill in multiline input fields and much more. All special keyboard functions are described in the interface manual in chapter 1.7.

dial_do can also enter FBoxText objects with the SMALL font. form_do would at best display pixel garbage on the screen.

#### 10.2.3.2.8 dial_move

```
GLOBAL VOID dial_move (DIALINFO *di, WORD x, WORD y, WORD w, WORD h);
```

- di = DIALINFO structure initialized by open_dial or dial_start.
- x,y,w,h = rectangle within which the Dialog is moved. Normally you should specify the desktop values without a Menu bar.

dial_move moves a Dialog within a bounding rectangle. When the mouse button is released, dial_move is exited. dial_move is called automatically by dial_do. Therefore, you only need this function if you have to edit the Dialog completely by yourself without dial_do. In this case you should call dial_move when the user has clicked on an object with the extended object type 17.

#### 10.2.3.2.9 dial_end

```
GLOBAL VOID dial_end (DIALINFO *di);
```

- di = DIALINFO structure initialized by open_dial or dial_start.

dial_end restores the background of the Dialog box and releases the memory. If the background could not be buffered, dial_end sends a redraw message (FMD_FINISH) to the AES. Since dial_end is called internally by dial_close, you don't usually have to use this function.

### 10.2.3.3 Connecting an own keyboard routine in dial_do

You can add your own keyboard routine to dial_do. dial_do then starts this keyboard routine for each key function. Only if it reports that it could not evaluate the key, the key will be evaluated by dial_do afterwards.

#### 10.2.3.3.1 get_Keybd

```
GLOBAL FORMKEYFUNC get_Keybd (VOID);
```

- Return value = The address of the currently latched routine. With NULL no routine is latched.

get_Keybd returns the address of the currently latched routine. This function is used, for example, if you call a second routine from a Dialog box with a latched routine, into which a separate keyboard routine is also to be latched. In this case you have to remember the address of the old function and enter the new one. If the second Dialog box has been left, the keyboard routine of the first Dialog box must be restored.

FORMKEYFUNC is declared in MYDIAL.H and is described in the explanation of set_Keybd.

#### 10.2.3.3.2 set_Keybd

```
GLOBAL VOID set_Keybd (FORMKEYFUNC fun);
```

- fun = A pointer to a function of its own that is to evaluate the keyboard events. Several parameters are passed over to this routine, which correspond in general to the parameters of form_keybd.

With set_Keybd you can add your own keyboard routine to dial_do, which evaluates the keys by itself. To explain the process a bit better, here is an excerpt from the interface Fileselektor-Source. The file selector inserts the routine "fsel_keybd" into dial_do to evaluate all special keys.

The individual parameters of the FORMKEYFUNC structure have the following meanings (see also fsel_keybd-Source):

- tree = address of the Dialog tree
- edit_obj = object number of the current EDIT object
- next_obj = Is currently always 0 and is only passed to form_keybd for compatibility reasons.
- kr = Contains the ASCII and scan code of the second last evnt_multi parameter (pkr). Needed by NKCC to generate a normalized keyboard code.
- ks = Contains the state of the special keys and corresponds to the third last parameter of evnt_multi (pks).
- onext_obj = Contains the next edit object (the object on which the cursor is to be positioned). Could be changed to place the cursor on a specific object.
- okr = Corresponds to kr. With this parameter you can return another key to the MyDials. The MyDials then evaluate this new key.

  *Important: The routine latched with set_Keybd is automatically reset to NULL in dial_end by the MyDials as soon as the Dialog has been left. So you have to latch the routine again at the next Dialog call.*

```
/*****************************************************************
keyboard handler
*****************************************************************/

LOCAL WORD fsel_keybd(OBJECT* tree, WORD edit_obj, WORD next_obj,
    WORD kr, WORD ks, WORD* onext_obj, UWORD* okr)

{
    WORD selobj, nkc_key, clicks = 1, ascii;

    nkc_key = normkey(ks, kr); /* Convert key code to NKCC code */

    /* some unimportant lines have been deleted
       ...
     */

    { ascii = nkc_key & 0x00FF;

    switch (nkc_key)
    {
    case NKF_FUNC | NKF_CTRL | NK_ESC: /* ESCAPE */
        selobj = SELHEAD;
        break;
    case NKF_FUNC | NKF_CTRL | SP: /* SPACE */
        selobj = SELHEAD;
        break;
    case NKF_FUNC | NKF_CTRL | NK_BS: /* BACKSPACE */
        selobj = SELCLOSE;
        break;
    case NKF_FUNC | NKF_CTRL | NK_CLRHOME: /* CLR/HOME */
        clicks = 2;
        selobj = USELECT;
        break;
    case NKF_FUNC | NKF_CTRL | NKF_SHIFT | NK_CLRHOME: /*SHIFT-CLR/HOME*/
        clicks = 2;
        selobj = DSELECT;
        break;
    case NKF_FUNC | NKF_CTRL | NK_UP: /* Cursor up */
        selobj = USELECT;
        break;
    case NKF_FUNC | NKF_CTRL | NK_DOWN: /* Cursor down */
        selobj = DSELECT;
        break;
    case NKF_FUNC | NKF_CTRL | NKF_SHIFT | NK_UP: /* SHIFT Cursor up */
        selobj = BSELECT;
        break;
    case NKF_FUNC | NKF_CTRL | NKF_SHIFT | NK_DOWN: /*SHIFT Cursor down*/
        selobj = SSELECT;
        break;
    case NKF_FUNC | NKF_CTRL | NKF_SHIFT | NK_LEFT: /* Cursor left */
        selobj = INFOL;
        break;
    case NKF_FUNC | NKF_CTRL | NKF_SHIFT | NK_RIGHT:/* Cursor right */
        selobj = INFOR;
        break;
    }
    if (selobj != NIL && !(tree[selobj].ob_flags & HIDETREE))
    {
        *Okr = 0;
        return (form_button(tree, selobj, clicks, onext_obj));
    }
    }
    return (NIL);
}
```

The file selector evaluates the keys. If a key combination was recognized, the file selector executes a `form_button` on the corresponding object and returns the return value of `form_button`.

Either you evaluate the key yourself and return a value that would also be returned by `form_button`, or you return NIL so that the key is evaluated by the MyDials afterwards. You can also change the return value by returning the new ASCII value in `okr`. The MyDials would then enter the key whose value was returned in okr into the edit field.

An example:

```
*okr = 127;
return (NIL);
```

At the source you can also see how easy it is to query key combinations with NKCC without needing a scancode table. Since NKCC always returns correct values even on foreign keyboards, I don't want to do without it anymore.

#### 10.2.3.3.3 Extending dial_do

You can add your own events to the dial_do of the MyDials and/or change the event behavior of the routine completely. There is also a function to add 2 own routines to dial_do:

```
set_MyEvnt (init_func, evnt_func)
```

- init_func is called right at the beginning of dial_do. A pointer to a structure is passed to the routine, in which all parameters of evnt_multi were entered (definition see mydial.h). The structure has already been initialized with the default values of the dial_do routine. init_func could, for example, also enter a timer event in the first entry of the structure.
- evnt_func is called by the MyDials directly after the evnt_multi of the dial_do routine. The first variable passed contains a pointer to the return value of evnt_multi, which indicates the occurred events (which can also be changed). The second variable contains a pointer to all evnt_multi parameters. In this routine one could evaluate the additionally entered events or also change or delete the values determined by evnt_multi.

With get_MyEvnt you can query the set addresses again.

### 10.2.3.4 Connecting a custom adjustment routine in dial_fix

You can add your own routine to dial_fix, which is called whenever MyDials do not know an extended object type. This makes it easier to adapt further extended object types without having to search the entire object tree for these objects in a separate routine after dial_fix.

#### 10.2.3.4.1 set_fixobj

```
GLOBAL VOID set_fixobj (FIXOBJFUNC fun);
```

- fun = A pointer to a function of its own that is to adapt extended object types that have not been recognized.

The individual parameters of the FIXOBJFUNC structure have the following meanings:

- tree = address of the Dialog tree
- obj = number of the object to be adjusted

#### 10.2.3.4.2 get_fixobj

```
GLOBAL FIXOBJFUNC get_fixobj (VOID);
```

- Return value = A pointer to the currently entered function. If no function has been entered yet, NULL is returned.

#### 10.2.3.4.3 add_ublock

```
GLOBAL USERBLK *add_ublock (WORD cdecl (*code)(PARMBLK *parmblock),
                            LONG obspec, WORD type);
```

- code = A pointer to the character function to be entered in the user block.
- obspec = The old obspec value to be saved in the ob_parm parameter of the user block structure.
- type = The old object type that is saved directly after the user block structure (see also OBBLK structure, de clarified in MYDIAL.H).
- Return value = A pointer to the newly created USERBLK structure. If there was not enough memory, NULL is returned.

add_ublock is usually only called by the MyDials in the dial_fix routine to create a USERBLK structure. If you have included your own function in dial_fix, you can use this function to create a USERBLK structure for your own extended objects.

If the USERBLK structure could be created, you must enter the pointer returned by add_ublock in the ob_spec parameter of the object.

An example from the MyDials:

```
switch (xtype)
{
case TITLELINE:
    if (type == G_BUTTON)
    {
        if ((userblk = add_ublock(draw_titleline, (LONG)ob->ob_spec,
            ob->ob_type)) != NULL)
        {
            ob->ob_height++;
            ob->ob_type = (ob->ob_type & 0xff00) | G_USERDEF;
            ob->ob_spec = (LONG)userblk;
        }
    }
    break;
}
```

### 10.2.3.5 Multiline Input Fields

There are several Dialog boxes in Interface that use multiline input fields (e.g. Free String). These fields are mostly managed by the MyDials, but it is not easy to use them, because the main program has to do some things. That's why I didn't document this function in the first MyDial “manual”. I would like to make up for this omission now.

All connected input fields, which are to be combined to a large field, must have the same parent. If you want to use several multiline input objects in a Dialog box, you have to place them in different parents. In principle, this works in the same way as with radio buttons.

You have to provide the required memory for the complete string when starting the program. In Interface it works as follows:

```
GLOBAL VOID fix_objs(tree)
OBJECT* tree;
{
    WORD obj;
    OBJECT* ob;
    UWORD type, xtype;
    WORD par, new, len;
    BYTE* txt;

    if (tree != NULL)
    {
        obj = NIL;

        do
        {
            whether = &tree[++obj]; /* Get address of object */
            type = ob->ob_type & 0xFF; /* Object type */
            xtype = ob->ob_type >> 8; /* extended object type */


            switch (type)
            {
            case G_FTEXT:
            case G_FBOXTEXT:
                /* Calculate the length of a multiline input object */
                if (ob->ob_flags & FLAGS10)
                {
                    len = 0;
                    par = parent(tree, obj);
                    if (par == NIL) par = ROOT;
                    new = tree[par].ob_head;
                    while (new != NIL && new != par)
                    {
                        switch (tree[new].ob_type & 0xff)
                        {
                        case G_FTEXT:
                        case G_FBOXTEXT:
                            if (tree[new].ob_flags & FLAGS10)
                                len += ((TEDINFO*)tree[new].ob_spec)->te_txtlen - 1;
                            break;
```

```
                    }
                    new = tree[new].ob_next;
                }

                /* Allocate memory for this length, and the individual */
                /* Combine fields into one field */
                if (len)
                {
                    if ((txt = mem_alloc(len + 1)) == NULL)
                        dial_alert(NULL, alertmsg[NOMEMORY], 1, 1, ALI_LEFT);
                    else
                    {
                        memset(txt, 0, len + 1);
                        len = 0;
                        new = tree[par].ob_head;
                        while (new != NIL && new != par)
                        {
                            switch (tree[new].ob_type & 0xff)
                            {
                            case G_FTEXT:
                            case G_FBOXTEXT:
                                if (tree[new].ob_flags & FLAGS10)
                                {
                                    ((TEDINFO*)tree[new].ob_spec)->te_ptext = &txt[len];
                                    len += ((TEDINFO*)tree[new].ob_spec)->te_txtlen - 1;
                                    undo_flags(tree, new, FLAGS10);
                                    /* set whether_flags 15 if the edit object */
                                    /* could be adjusted successfully */
                                    do_flags(tree, new, FLAGS15);
                                }
                                break;
                            }
                            new = tree[new].ob_next;
                        }
                    }
                }
            }
            break;
        }
        while (!(ob->ob_flags & LASTOB));
    } /* if */
} /* fix_objs */
```

In the routine that calls and evaluates the Dialog, the following must be defined:

```
LOCAL BOOLEAN edit_alert(window, obj)
WINDOWP window;
WORD obj;

{
    /* unimportant deleted! */

    text = (BYTE*)(kf->tree);
    icon = al_token((BYTE*)kf->tree, al_str, &txtnum, &butnum);

    /* Initialize Dialog boxes, filling string remainder with 0 */
    for (i = 0; i < 5; i++)
    {
        if (!strcmp(al_str[i], " "))
            memset((((TEDINFO*)alerttree[ALZ1 + i].ob_spec)->te_ptext, 0,
                    strlen(((TEDINFO*)alerttree[ALZ1 + i].ob_spec)->te_pvalid));
        else
                strncpy((((TEDINFO*)alerttree[ALZ1 + i].ob_spec)->te_ptext, al_str[i],
                    strlen((((TEDINFO*)alerttree[ALZ1 + i].ob_spec)->te_pvalid)));
    }

    /* Dialog processing */
    /* ... */
    /* Evaluation of the Dialog content: */

    if (exit_obj == ALOK)
    {
        for (i = 0; i < 5; i++)
        {
            strncat(alstrg, ((TEDINFO*)alerttree[ALZ1 + i].ob_spec)->te_ptext,
```

```
                strlen((((TEDINFO*)alerttree[ALZ1 + i].ob_spec)->te_pvalid)));
            strcat(alstrg, "|");
        }
        /* and so on... */
    }

    return (ret);
}
```

To cut a long story short: Internally, the individual input fields are stored in memory as a single large string. So that the individual fields can still be separated from each other, the unused part must be filled with zero bytes during initialization. Otherwise, old letter rubbish could still appear during processing.

During the evaluation one must pay attention to the fact that one works with strncpy. If lines 1 and 2 are connected (i.e. are no longer separated by null bytes), you would get lines 1 and 2 together immediately with a strcpy. If you process the whole thing in a for...next loop, the next strcpy command would return line 2 again!

#### 10.2.3.5.1 find_next

```
GLOBAL WORD find_next (OBJECT *tree, WORD obj);
```

- `tree` = address of the Dialog tree
- `obj` = Start object
- Return value = object index of the next input line

`find_next` searches from "`obj`" the next line of a multiline input object. `find_next` is required internally by the MyDials. Interface needs this routine for the window Dialogs to find the next input object.

#### 10.2.3.5.2 find_prev

```
GLOBAL WORD find_prev (OBJECT *tree, WORD obj);
```

- `tree` = address of the Dialog tree
- `obj` = Start object
- return value = object index of the previous input line

`find_prev` searches from "`obj`" the previous line of a multiline input object. `find_prev` is required internally by the MyDials. Interface needs this routine for the window Dialogs to find the previous input object.

#### 10.2.3.5.3 Scrolling input fields

For this purpose, the edit field must have type 26 (LONGINPUT) and may only consist of one long input field. Such edit fields can, of course, be reduced in size at will. In Interface you can now edit input fields with a length of up to 255 characters.

### 10.2.3.6 2.3.6 Popup-Routine

#### 10.2.3.6.1 popup_Menu

```
GLOBAL WORD popup_Menu (OBJECT *tree, WORD obj, WORD x, WORD y,
                        WORD center_obj, BOOLEAN relative,
                        WORD bmsk, BOOLEAN *ok);
```

- `tree` = Address of the object tree in which the popup Menu is located.
- `obj` = If the Dialog tree consists only of the popup, obj = ROOT. If there are several popups in the Dialog tree, `obj` specifies the parent of the individual object entries (see also Interface Resource, POPUPS Tree).
- `x, y` = Position of the upper left popup corner (or of the object trum if center_obj != NIL)
- `center_obj` = Contains the index of the popup entry to be centered.
- `relative` = TRUE: Popup is drawn at the mouse position, x and y should be 0.
- `bmsk` = Specifies the mouse button used to exit the popup (1 = left, 2 = right, 3 = left or right).
- `ok` = TRUE, if background could be saved, otherwise FALSE. If a null pointer is passed, ok is not set.
- Return value = The selected entry, or NIL if no entry was selected.

`popup_Menu` draws a popup and performs the complete processing. The popup may contain any objects. Each selectable object is inverted if the mouse is over it or if it is moved with the cursor keys. You can also place keyboard strings in the popup and select them with the shortcut.

If ok contains FALSE, the calling routine must redraw the Dialog at the popup position, since there is not enough memory to save the popup background. In this case, the popups send a redraw message to the AES.

#### 10.2.3.6.2 popup_select

```
GLOBAL WORD popup_select (WORD wh, OBJECT *dialtree, WORD btn,
                          OBJECT *poptree, WORD obj, BOOLEAN docheck,
                          WORD docycle, BOOLEAN *ok));
```

- wh = window handle for the window Dialog containing the calling popup button. For normal Dialogs you specify NIL (-1).
- dialtree = Dialog tree
- btn = Index of the object that calls the popup (allowed types: BoxText, Button, String, Userdef-Boxtext or Userdef-Button)
- poptree = Address of the object tree in which the popup Menu is located.
- obj = index of the popup. The individual entries must consist of strings or keyboard-operated strings (Userdef 19).
- docheck = Check active popup entry (with button text).
- docycle = DO_POPUP: display popup; DO_CYCLE, DO_POPNEXT: display next entry; DO_POPPREV: display previous entry
- ok = TRUE, if background could be saved, otherwise FALSE. If a null pointer is passed, ok is not set.
- Return value = The selected entry, or NIL if no entry was selected.

popup_select manages a popup button with a cycle button. The popup button always displays the text last selected in the popup Menu. If a cycle object was clicked, you can pass DO_CYCLE to popup_select so that the routine enters the next text of the popup into the popup button.

If you call popup_select with DO_POPUP, a popup is displayed and processed. The popup is always positioned (if possible) so that the last selected entry is exactly above the popup button.

### 10.2.3.7 Alertboxes

#### 10.2.3.7.1 do_alert

```
GLOBAL WORD do_alert (WORD defbut, CONST BYTE *txt);
```

do_alert corresponds to form_alert in the parameters. The only difference is that you have not only 3 icons, but 16 icons available. The number of the desired icon is passed hexadecimal, i.e. the values from 0-F are allowed. The individual icons can be viewed in Interface.

Of course, the MyDial alerts have buttons that can be operated from the keyboard. Before the character, which is to be used as shortcut, one simply enters an angular bracket. A valid alert with a shortcut on the “O” of “OK” could look like this:

```
do_alert (1, "[5][Testalert][[OK]");
```

If you want to center a text line of the alert or output it right justified, simply enter a special character at the first position of the string. These special characters are defined in MYDIAL.H as ALCENTER (\001) and ALRIGHT (\002).

An example alert:

```
do_alert (1, "[6][left-aligned|\001centered|\002right-aligned][[OK]");
```

#### 10.2.3.7.2 dial_alert

```
GLOBAL WORD dial_alert (OBJECT *alicon, CONST BYTE *string,
                        WORD defbut, WORD undobut, WORD align);
```

- alicon = pointer to an own object to be shown instead of the icon. With NULL the icon specification is used in the string.
- string = Alert string as in form_alert
- defbut = Default exit button
- undobut = Default Undobutton; selected by pressing the UNDO key
- align = Position the alert text: ALI_LEFT, ALI_CENTER, ALI_RIGHT center the complete alert text left justified, centered or right justified. The 3 constants are defined in VDI.H. Individual lines can be positioned differently by prefixing them with AL CENTER or ALRIGHT.
- Return value = dial_alert returns the same value as form_alert to

dial_alert is called by do_alert. With dial_alert you can display your own object in the alert and define an undo button. If the text of a button starts with a dot, it is used as default button. If it starts with a colon, it is the default undo button. In these two cases, the values of defbut and undobut are ignored.

#### 10.2.3.7.3 get_icon

```
GLOBAL BITBLK *get_icon (WORD icon);
```

- icon = digit between 0 - 15
- Return value = Pointer to the BITBLK structure of the corresponding Al ertbox icon of the MyDials.

#### 10.2.3.7.4 al_token

```
GLOBAL WORD al_token (CONST BYTE *f_alert, BYTE str[8][50],
                      WORD *txtnum, WORD *butnum);
```

- f_alert = pointer to the alert string
- str = pointer to a buffer on which al_token should split the strings
- txtnum = Returns the number of text lines of the alert.
- butnum = Returns the number of alert buttons.
- Return value = number of the alert icon (e.g. for get_icon)

al_token splits an alert string into individual strings and writes them to "str". The first 5 strings contain the normal text; the remaining 3 strings contain the button texts.

#### 10.2.3.7.5 alert_data (for alert boxes in windows)

```
GLOBAL WORD alert_data (OBJECT *alicon, CONST BYTE *string, WORD defbut,
                        WORD undobut, OBJECT **tree, WORD *firstbut,
                        BYTE alert_texte[][MAXASLENGTH], RECT *size,
                        WORD al_dat[]);
```

- alicon = pointer to an own object to be shown instead of the icon. With NULL the icon specification is used in the string.
- string = Alert string as in form_alert
- defbut = Default exit button
- undobut = Default Undobutton; selected by pressing the UNDO key
- tree = If not equal to NULL, this variable contains a pointer to the alert tree after the function has been called.
- firstbut = If not equal to NULL, this variable contains the object number of the first button after the function has been called.
- alert_texte = If not equal to NULL, then this array contains all the individual texts of the alert after the alert has been called.
- size = Contains the dimensions of the alert box after the function has been called.
- al_dat = If NULL is not equal, this array contains 8 WORDs with data from the alert box that must be passed to the alert_draw function.
- align = Position the alert text: ALI_LEFT, ALI_CENTER, ALI_RIGHT center the complete alert text left justified, centered or right justified. The 3 constants are defined in VDI.H. Individual lines can be positioned differently by prefixing them with AL CENTER or ALRIGHT.

This function calculates all the data in an alert box (including the size and position of all elements). The results must be passed to alert_draw so that the alert box is drawn within a window.

In principle, the procedure for alert boxes in windows is as follows:

1. calculate the Alertbox with alert_data.
2. open a window the size of the alert box.
3. call alert_draw with the data determined in alert_data for each clipping rectangle for redraws of the window. In size.x and size.y you pass the position of the alert box.
4. process the window Dialog like a normal window Dialog (only difference: call alert_draw instead of objc_draw for redraw).
5. close the window and call alert_close.
6. the selected button number is determined by subtracting the object number of the exit button "firstbut" determined with alert_data from the object number of the exit button and adding 1 to it.

If there is no more window handle for an alert, you should call dial_alert instead.

#### 10.2.3.7.6 alert_draw (for alert boxes in windows)

```
GLOBAL VOID alert_draw (OBJECT *alicon, BYTE alert_texte[][MAXASLENGTH]),
                        RECT *size, WORD al_dat[], WORD align,
                        RECT *clip);
```

- alicon = pointer to an own object to be shown instead of the icon. With NULL the icon specification is used in the string.
- alert_texts = This array contains all individual texts of the alert.
- size = Contains the dimensions of the alert box.
- al_dat = Contains 8 WORDs with the data of the alert box.
- align = Position the alert text: ALI_LEFT, ALI_CENTER, ALI_RIGHT center the complete alert text left justified, centered or right justified. The 3 constants are defined in VDI.H. Individual lines can be positioned differently by prefixing them with AL CENTER or ALRIGHT.
- clip = Pointer to a clipping rectangle to be taken into account when drawing.

alert_draw draws an alert box within the given clipping rectangle.

#### 10.2.3.7.7 alert_close (for alert boxes in windows)

```
GLOBAL VOID alert_close (WORD al_dat[]);
```

- al_dat = Contains 8 WORDs with the data of the alert box.

If the alert box has been closed, this function must be called so that the Dialog is initialized again and is available for other alerts.

### 10.2.3.8 Various useful MyDials help routines

#### 10.2.3.8.1 getcookie

```
GLOBAL BOOLEAN getcookie (LONG cookie, LONG *p_value);
```

- cookie = value of the cookie to be searched for
- p_value = Pointer to the info structure of the cookie.
- Return value = If TRUE the cookie was found, otherwise FALSE.

getcookie searches for a cookie. The MyDials search for the “VSCR” cookie in order to position Dialogs also under bigscreen and other virtual screen managers always in the visible part.

#### 10.2.3.8.2 ask_vscr

```
GLOBAL BOOLEAN ask_vscr (VOID);
```

- Return value = If there is a VSCR cookie and the screen cut is smaller than the entire screen, TRUE is returned.

The MyDials use this function to ask whether a program such as Bigscreen is installed that simulates a large screen, but always displays only a section of it on the monitor. If this is the case, the MyDials always center the Dialogs on this visible part, so that you don't have to search the whole area for the Dialog.

Some programs also install a VSCR cookie although the whole screen is displayed (e.g. NVDI for CrazyDots). Therefore, this function also checks if the visible part of the screen matches the whole screen and returns FALSE in this case.

#### 10.2.3.8.3 parent

```
GLOBAL WORD parent (OBJECT *tree, WORD obj);
```

- tree = address of the Dialog tree
- obj = address of the object
- Return value = parent returns the parent of an object. If there is no parent because obj = ROOT, parent returns NIL.

#### 10.2.3.8.4 get_obspec

```
GLOBAL LONG get_obspec (OBJECT *tree, WORD obj);
```

- tree = address of the Dialog tree
- obj = address of the object
- Return value = ob_spec value of the object.

get_obspec also returns the correct ob_spec value for USERDEF objects. The ob_spec value is overwritten when the object is initialized by a pointer to a USERBLK structure. In this USERBLK structure, the old ob_spec value is stored again. get_obspec returns the original ob_spec value from the USERBLK structure for USERDEF objects.

#### 10.2.3.8.5 set_obspec

```
GLOBAL VOID set_obspec (OBJECT *tree, WORD obj, LONG obspec);
```

- tree = address of the Dialog tree
- obj = address of the object
- obspec = New ob_spec value

set_obspec replaces the old ob_spec value of an object with a new value. For USERDEF objects, the new value is entered in the USERBLK Structure.

#### 10.2.3.8.6 normkey

```
GLOBAL UWORD normkey (WORD ks, WORD kr);
```

- ks = The state of the special keys. ks corresponds to the third last parameter of evnt_multi (pks).
- kr = the ASCII and scan code of the second last evnt_multi parameter (pkr). Required by NKCC to create a normalized keyboard code.

With normkey you can convert the evnt_multi key code into the normalized NKCC key code. This makes it much easier and safer to query all special keys and key combinations, since NKCC provides values that are independent of the keyboard. See also NKCC.H and the NKCC instructions on this diskette.

#### 10.2.3.8.7 9.2.3.8.7 get_idx

```
GLOBAL WORD get_idx (OBJECT *tree, WORD obj, WORD cpos);
```

- tree = address of the Dialog tree
- obj = address of the object
- cpos = cursor position within an edit object
- Return value = The absolute cursor position. If the edit object looks like “Date: __/__/____”, and the cursor is at the first position (0), the absolute position is 7, since 7 letters of the mask appear before the first cursor position.

get_idx is used internally by the MyDials for the MyDial function obj_edit. It is also used by Interface for the window Dialogs.

#### 10.2.3.8.8 form_Keybd

```
GLOBAL WORD form_Keybd (OBJECT *tree, WORD edit_obj, WORD next_obj,
                        WORD kr, WORD ks, WORD *onext_obj, UWORD *okr);
```

- tree = address of the Dialog tree
- edit_obj = object number of the current EDIT object
- next_obj = Is currently always 0 and is only passed to form_keybd for compatibility reasons.
- kr = Contains the ASCII and scan code of the second last evnt_multi parameter (pkr). Needed by NKCC to generate a normalized keyboard code.
- ks = Contains the state of the special keys and corresponds to the third last parameter of evnt_multi (pks).
- onext_obj = Contains the next edit object after the return (the object on which the cursor should be positioned).
- okr = Corresponds to kr.
- Return value = see form_keybd

form_Keybd corresponds to form_keybd. It only calls the keyboard routine before the shortcut evaluation and before the execution of form_keybd (see set_Keybd). form_Keybd is used internally by the MyDials and by Interface for the window Dialogs.

#### 10.2.3.8.9 shortcut

```
GLOBAL WORD shortcut (OBJECT *tree, WORD startobj, WORD ks, WORD kr);
```

- tree = address of the Dialog tree
- startobj = start object, whose children are checked for shortcuts who should use them
- ks = Contains the state of the special keys and corresponds to the third last parameter of evnt_multi (pks).
- kr = Contains the ASCII and scan code of the second last evnt_multi parameter (pkr). Needed by NKCC to generate a normalized keyboard code.
- Return value = object index of the object containing the shortcut. If the shortcut was not found, NIL is returned.

shortcut is used internally by the MyDials and by Interface for the window Dialogs.

#### 10.2.3.8.10 obj_edit

```
GLOBAL WORD obj_edit (OBJECT *ob_edtree, WORD ob_edobject, WORD kstate,
                      WORD ob_edchar, WORD *ob_edidx, WORD ob_edkind,
                      WORD mode, WORD *next_obj, WORD winhndl));
```

- ob_edtree = address of the Dialog tree
- ob_edobject = object number of the current EDIT object
- kstate = Contains the state of the special keys and corresponds to the third last parameter of evnt_multi (pks).
- ob_edchar = Contains the ASCII and scan code of the second last evnt_multi parameter (pkr). Required by NKCC to generate a normalized keyboard code.
- ob_edidx = cursor position within the object
- ob_edkind = type of edit function: ED_INIT: initialize obj_edit, switch on cursor, ED_CHAR: process characters, ED_END : switch off cursor again
- mode = TRUE: obj_edit does not display any screen output and only inserts the character into the edit string. This prevents a hectic flickering when inserting multiple characters at once (e.g. insert clipboard string into edit field). TRUE is only used internally by obj_edit to recursively call itself. FALSE: Screen output allowed (default mode)
- next_obj = Contains the next Edit object. May be changed by obj_edit.
- winhndl = winhndl is a window handle. If winhndl is not equal to NIL, the cursor is drawn over the rectangle list of the window to which winhndl belongs. Used by Interface for the window Dialogs.

obj_edit is used by MyDials internally and by Interface for the window Dialogs. All the special keyboard functions of the MyDials have been realized by obj_edit itself.

#### 10.2.3.8.11 scrap_clear

```
GLOBAL WORD scrap_clear (VOID);
```

scrap_clear deletes the GEM clipboard.

All MyDial clipboard routines can only work if any program has already set the clipboard path with scrp_write. So the main program should set the clipboard path before using MyDial (if necessary) and create a clipboard folder if necessary.

#### 10.2.3.8.12 ascii_head

```
GLOBAL VOID ascii_head (BYTE *header);
```

- header = A pointer to the new heading of the MyDial ASCII table.

Since you could change the MyDial resource with a floppy monitor at best, there is this call, so that you can change the only German text of the MyDials with it. ascii_head only changes the pointer of the text, i.e. the transferred text must remain at the old place and must not be overwritten.

#### 10.2.3.8.13 my_menu_key

```
GLOBAL BOOLEAN my_menu_key (OBJECT *menu, MKINFO *mk, WORD *title,
                            WORD *item);
```

- menu = Menu tree
- mk = structure describing the pressed key
- title = Contains as result the Menu title of “item”.
- item = Contains the Menu item for which the shortcut matches as result.
- Return value = TRUE, if the shortcut was found, otherwise FALSE

my_menu_key searches a Menu tree for a shortcut. In mk, the key for which the function is to search must be described. The MKINFO structure is defined in MGLOBAL.H.

The MKINFO structure must be filled with the evnt_multi values as follows:

```
/* directly after the main event_multi: */
mk.scan_code = mk.kreturn;
mk.kreturn = normkey(mk.kstate, mk.kreturn);
if (mk.kreturn & NKF_SHIFT) /* set both SHIFT keys */
mk.kreturn |= NKF_SHIFT;
mk.kreturn &= ~NKF_CAPS; /* Delete CAPSLOCK */
mk.shift = (mk.kstate & (K_RSHIFT | K_LSHIFT)) != 0;
mk.ctrl = (mk.kstate & K_CTRL) != 0;
mk.alt = (mk.kstate & K_ALT) != 0;
```

mk.kstate and mk.kreturn contain the return values kstate and kreturn of evnt_multi before these lines are executed.

If you use the goat libraries, you can replace the goat routine `is_menu_key` with `my_menu_key`. `my_menu_key` also recognizes combined shortcuts like Shift-Control-U.

#### 10.2.3.8.14 get_sysfnt

```
GLOBAL BOOLEAN get_sysfnt (WORD vdi_handle, WORD *font_id, WORD *gl_point,
                           WORD *gl_wchar, WORD *gl_hchar);
```

- `vdi_handle` = current workstation
- `font_id` = get the system font ID (see `vst_font`)
- `gl_point` = get the point size of the system font (see `vst_point`)
- `gl_wchar` = get the width of the system font
- `gl_hchar` = get the height of the system font (s. `vst_height`)

`get_sysfnt` is used by the MyDials at opening of the workstation to get the current system font and character size. The function also returns correct results under MultiTOS.

In addition, the function sets the determined system font for the current workstation.

#### 10.2.3.8.15 form_Keybd

`form_Keybd` corresponds in principle to the AES `form_keybd`. `MyDialform_Keybd` internally first calls the keyboard routine latched with `set_Keybd` before it passes the key to `form_keybd`.

#### 10.2.3.8.16 form_Button

`form_button` evaluates a window handle in addition to the AES `form_button` to redraw an object via the window rectangle list. If NIL is passed as handle, the object is drawn without clipping with `objc_draw`.

#### 10.2.3.8.17 objc_setobspec

This function copies a string into any object.

#### 10.2.3.8.18 objc_getobspec

This function copies the text of an object into a string. The File Selection of Interface

### 10.2.4 An Example Program for Dialog Editing

`HndlDial` works through a Dialog completely, so I spare myself an example. If you have to work through a Dialog with `open_dial` and `close_dial` to evaluate `touchexit` objects and redraw parts of the Dialog, you can proceed as follows:

```
LOCAL VOID show_form(tree, editobj)
OBJECT* tree;
WORD editobj;

{ WORD but, between;
DIALINFO di;

open_dial(tree, TRUE, NULL, &di);

dial_draw(&di);

/* As long as Touchexit objects were clicked, the Dialog */
/* will not be abandoned.                                */

th
{
    but = dial_do(&di, editobj) & 0x7fff;
    tree[but].ob_state &= ~SELECTED;
    objc_draw(tree, ROOT, MAX_DEPTH, di.x, di.y, di.w, di.h);
while (tree[but].ob_flags & TOUCHEXIT);

close_dial(FALSE, NULL, &di);
}
```

## 10.3 Placing Dialogs in Windows

It's not very hard to put a Dialog in a window, it's just a lot of work :-) In principle it works like this:

1. you open a window with the size of the Dialog.
2. draw the Dialog into the window with the clipping of the window rectangle list.
3. the window `form_do` (which you still have to program) is jumped to if...
   - the Dialog is located in the uppermost window and your program receives a keyboard event message.
   - A button event has been reported to you and the window Dialog is located under the mouse.
4. If an exit or touch exit object was clicked, a routine is started which evaluates the object and reports back whether the Dialog should be closed after the evaluation.

To make this work, you have to know how the `form_do` (or better `dial_do`) command works internally. Because you have to process the events in the same way, so that there are no differences in the handling of the Dialog. Therefore, the MyDial-`dial_do` routine is located in the C source on this diskette. The file is called `DIAL_DO.C`.

All routines called by `form_xdo` are accessible in MYDIAL.H from the outside. You only have to pass the event to the routine and evaluate it instead of the `form_xdoevnt_multi`. It's a bit more complicated, but with these routines anyone who knows something about windows should be able to put their Dialogs into windows.

It's a bit easier for everyone who uses the goat routines from the book “Vom Anfänger zum GEM-Profi” 2nd edition. In the 2nd edition window Dialogs are already used and you only need to adapt the routines “`click_window`” and “`key_window`” to the MyDials in the WINDOW.C module.

In Interface you can switch between window Dialogs and normal Dialogs at any time, because Interface simply copies the Dialog heading into the window title and makes the window so small that the FlyDial corner and the Dialog heading are not displayed.

As already mentioned above, I am happy to help anyone who has questions about this topic.

By the way, `form_xdo` is based on the original Digital Research `form_do` source. I only adapted it “slightly” to the MyDials.

## 10.4 3D Dialogs

When programming the 3D effects for the Mydials, care was taken to remain as compatible as possible with the previous Mydials, on the one hand to continue to allow the previous monochrome display, and on the other hand to make the switch to 3D as easy as possible.

Almost all 3D effects can be achieved simply by setting the DRAW3D status of the object in question. To simplify this for entire object trees, a useful feature has been added to the current interface: if only the root object is selected from an object tree and then the function 'Set Edit Flags' is called, the DRAW3D flag (and only this one) can be set or deleted for ALL objects of this tree.

To switch on the 3D effects, only one additional parameter has to be passed to 'dial_init', which specifies whether the 3D representation should be used or not (TRUE/FALSE). Unfortunately, it is only possible to make this setting during program initialization, since the resource objects for the 3D mode must be converted into USERDEF objects and their size changed. However, these adjustments could only be reversed with great effort at program runtime, which is why they were omitted.

If the 3D representation is to be made switchable, the responsible variable must first be saved in a configuration file in order to be evaluated at the next program start. This procedure is also used in Interface.

### 10.4.1.1 Setting possibilities of the 3D-Mydials

There are two Mydial functions that can be used to change parameters used by the Mydials. With the function

```
VOID dial_3Dcolors (WORD back, WORD light, WORD dark);
```

the colors in which the 3D Dialogs are drawn can be changed. Normally, the background color (back) is VDI color 8, which should usually be defined as light gray. light' corresponds to VDI color 0 (white) and 'dark' is preset as color index 9 (dark gray). If you change the color, you have to consider that the radio buttons Mydial-intern are available as color icons and are not adapted to other colors at least at this time. Therefore, the possibility of color change should only be used if changed colors are absolutely necessary. It makes more sense to reset the color palette to the default system colors before displaying Dialogs.

You can use the

```
VOID dial_3Droot (BOOLEAN border, BOOLEAN perimt);
```

the appearance of 3D root objects can be changed. This option is intended primarily for the display of 3D window Dialogs. For these, the 3D outlined frame can be switched off (border = FALSE) and the black border can be omitted (`perimt` = FALSE), as this results in a more pleasant display of window Dialogs.

Since the switching of these parameters is immediately globally effective for all Mydial functions, it must be ensured that the correct parameters are always set for the respective Dialog type, while normal and window Dialogs can be displayed at the same time. The easiest way to do this is to use a central editing routine for all non-window Dialogs, in which the 3D parameters are set accordingly at the beginning and reset again for window Dialogs at the end. Since standard Dialogs always lock all other screen output, there can be no redraws of possibly visible window Dialogs during their processing.

#### 10.4.1.2 Adaptation of Resources to Special 3D Effects

As mentioned above, most 3D effects can be achieved by simply setting the DRAW3D flag. In some cases, however, this is not sufficient to achieve a reasonable display in both monochrome and 3D color mode with the previous resource design.

These cases include text input objects (G_FTEXT and G_FBOXTEXT) that are directly below or above other objects (i.e. in character height spacing). In the 3D display, these fields are forced to be slightly larger in all directions so that they overlap the neighboring objects. This can be avoided by redesigning the resource so that these edit objects have more space available.

If you don't want to have a whole character height distance to other objects, you can make edit fields (i.e. your (G_FTEXT invisible) frame) e.g. two lines high, so that the field content is centered in the middle of the two lines. Please do not position pixel by pixel, otherwise the compatibility to smaller resolutions will be lost!

A special case are several edit fields, which are directly below each other, possibly even multi-line input fields, as for example partially in interface. With these, the 3D frames would overlap or they would be too far apart with the above method.

With such input fields the DRAW3D status should be deleted and instead an IBOX should be placed as parent object under all fields. This must have the OUTLINED status (and of course DRAW3D) and be set to 'No frame' and frame color 0 (white). The latter ensures that this IBOX is not drawn in monochrome mode. An example can be found in the IMAGE Dialog of Interface.

Another special case are popup Menus. For a selected popup entry to be highlighted with white, it must have the extended object type 23 (0x17) and be of the type STRING, TEXT or BOXTEXT. In addition, it must of course have the DRAW3D flag. So just add DRAW3D and Extended Type 23 to all popup entries. The popup frame can remain as it was except for DRAW3D.

Finally, there is another special case, which can also be intercepted with an extended object type. It is about slider elements, as they are needed in list boxes (e.g. in the file selector). These are normally created as BOX or BOXCHAR objects (sliders or arrow buttons) with a frame of “1 pixel outside”. In the 3D view, however, the frame then appears too thin. Therefore, in addition to the DRAW3D flag, the Extended Type 25 (0x19) can be entered for these objects, whereupon the mydials increase the frame thickness in 3D by one pixel. In addition, the pattern and color of a BOX object without a DRAW3D flag (but with XType 25) are changed. This also allows the slider backgrounds to be adapted to the color display.

Another problem with slider elements in the above Dialog is that they overlap when positioned in the drawing grid due to the outer frame. In monochrome this was hardly noticeable up to now, at most there was a slight flickering of the redraw. In the 3D representation, however, the 3D effect is “sabotaged” by the overlap. For this reason, the Mydials offer a new function call, with which the object positions can be corrected. This function can be called independently of the 3D mode, since it also eliminates the mentioned redraw problems in monochrome.

The function is called `'dial_slinit'` and gets the address of an object tree as well as the resource indices of all slider objects. It can also be called for single horizontal or vertical sliders - for the non-existing elements it simply passes NIL (-1). The title bar or the closer can also be omitted in this way.

In the selected state, 3D buttons are normally drawn in a “pressed” state. If buttons in a Dialog are not used as exit buttons, the pressed state may differ too little from the normal state. If the button in the selected state is also to be drawn dark with inverted text, `ob_flags` 15 must be set. An example of this is the “S” button in the icon editor.

*IMPORTANT: So that the objects can be drawn in 3D mode, they are converted to `Userdefs`. Therefore, `ob_spec` no longer contains the old value but a pointer to a USERBLKS structure. To get*

*the old value back, you always have to use the function "`get_obspec`" if you want to access `ob_spec`! This function always returns the correct value, regardless of whether the object was converted into a Userdef or not. If you do without `get_obspec`, it will inevitably crash in 3D mode.*

# 11 Document revision history

- English V1.0 – December 2019: First official release
- English V0.2 – December 2019: Added the MyDial chapter **based on The MyDial.txt** file provided with the Interface 2.33 distribution diskettes (this file is supposed to contain latest information about MyDial). Rearranged the chapters (MyDial chapter pushed to end of document). Fixed many initial translation mistakes. Added hyperlinks to document.
- English V0.1 – November 2019: The formatted German document has been translated by DrCoolZic. Version 0.1 did not contain the MyDials chapter and was produced for first review.
- Formatted German manual - November 2019: The PDF has been converted to text (OCR) by DrCoolZic and formatted with Microsoft Word, to ease of conversion.
- Scan Original - November 2019: The document has been scanned in PDF by Dhelber in 2019.
- Original German Manual - 1994: The original document in German was written in by Olaf Meisiek in 1994.