FC_Application

Olivier LAVIALE 2004

COLLABORATORS

	TITLE : FC_Application	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Olivier LAVIALE 2004	January 13, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

FC_Application

1.1 Feelin : FC_Application

FC_Application

IDs: Static Super: FC_Object Include: <libraries/feelin.h>

This class is the master class for all Feelin applications. It serves as a kind of anchor for all input, either coming from the user or somewhere from the system, e.g. commodities or ARexx messages.

An application can have any number of sub windows, these windows are the children of the application. A FC_Family object is used to manage them, thus you can use FC_Family methods to add or remove objects.

Each Application uses a FC_DisplayContext object to manage its environnement. FC_Application objects recognise and handle FA_Pen_Xxx and FA_SchemeSpec attributes, allowing each application to have its own color scheme.

METHODS

FM_Application_Setup FM_Application_Cleanup

FM_Application_Run FM_Application_Shutdown

FM_Application_Awake FM_Application_Sleep

FM_Application_PushMethod FM_Application_Load

FM_Application_Save FM_Application_Resolve

FM_Application_ResolveInt FM_Application_OpenFont

FM_Application_AddSignalHandler FM_Application_RemSignalHandler

ATTRIBUTES

FA_Application FA_Application_Author

FA_Application_Base FA_Application_Copyright

FA_Application_Description FA_Application_Title

FA_Application_Version FA_Application_Sleep

FA_Application_Broker FA_Application_BrokerPri

FA_Application_BrokerHook FA_Application_BrokerPort

FA_Application_WindowPort FA_Application_Context

FA_Application_Signal FA_Application_UserSignal

FA_Application_OBJSpace

PREFERENCES

FP_Application_ColorScheme FP_Font_Tiny

FP_Font_Normal FP_Font_Big

FP_Font_Fixed

TYPES

FeelinEvent FeelinSignalHandler

1.2 FC_Application / FM_Application_AddSignalHandler

NAME

FM_Application_AddSignalHandler -- (00.00)

SYNOPSIS

F_Do(Obj,FM_Application_AddSignalHandler,struct FeelinSignalHandler *sh);

FUNCTION

Allow classes to react on signals of private message ports. A class can create messages ports and react on their signals on its own without interferring the main program. A game class could open the gameport.device and react on joystick messages, an html class could talk to network all on its own.

All this helps to further encapsulate your program into subclasses and make it a lot more easy to maintain.

To be abble to react on signals, you must fill out a FeelinSignalHandler structure (probably located in your local object data) and call FM_Application_AddSignalHandler with the structure as parameter. From now on, your class will receive the specified method whenever one of the given signals arrives.

Since we're talking here about method of FC_Application, it's clear that you cannot call it until you know about your application object. Good places for FM_Application_AddSignalHandler / FM_Application_RemSignalHandler are probably the FM_Setup / FM_Cleanup methods of your class.

INPUTS

sh - Pointer to an initialized struct FeelinSignalHandler .

RESULT

FM_Application_AddSignalHandler cannot fail, the result value of the method is currently undefined.

NOTE

You must match each FM_Application_AddSignalHandler with exatly one FM_Application_RemSignalHandler method. Do not add a FeelinSignalHandler which is currently in use, actually it is not dangerous but the method will complains about that.

TIMER

FC_Application implements a builtin timer. By using this one instead of creating your own I/O requests, you avoid the problem of having each instance of your object allocating a signal bit.

To make use of this timer, use the above described procedure of initializing and adding your FeelinSignalHandler structure, but set the FF_InputHandler_Timer in Flags. Furthermore, specify the number of secs and micros after which you want to receive your method in Secs and Micros. Note that Secs and Micros are in fact part of a union placed at the same memory location as Signals, do not use Signals when FF_InputHandler_Timer is set.

Besides this, using the timer is similiar to other input handlers. Removing with FM_Application_RemSignalHandler is not different at all.

1.3 FC_Application / FM_Application_Awake

NAME

FM_Application_Awake -- (06.00)

SYNOPSIS

[PRIVATE]

NOTE

It is safe to call this method from another task because it actually sends a message to the application to avoid "death circle".

SEE ALSO

FM_Application_Sleep FA_Application_Sleep

1.4 FC_Application / FM_Application_Cleanup

NAME

FM_Application_Cleanup -- (05.00) [For use within classes only]

SYNOPSIS

[PRIVATE]

SEE ALSO

FM_Application_Setup

1.5 FC_Application / FM_Application_Load

NAME

FM_Application_Load -- (05.00)

SYNOPSIS

F_Do(Obj,FM_Application_Load,STRPTR Name);

FUNCTION

FM_Application_Save , FM_Application_Load and FA_ID offer an easy way of saving and loading a programs configuration.

Each gadget with a non NULL FA_ID will get its contents saved during FM_Application_Save and restored during FM_Application_Loa
This makes it very easy to design a configuration window with "Save", "Use" and "Cancel" buttons to allow the user storing the settings. When the application starts, you would just have to call FM_Application_Load and the stored settings will be read and installed.

Not all classes are able to import and export their contents.

INPUTS

Name - Name of the file you wish to load the settings from. Usually you won't need to think of a real name but instead use one of the magic cookies FV_Application_ENV or FV_Application_ENVARC.

NOTE

Currently object's datas are automatically imported and exported.

SEE ALSO

FM_Application_Resolve

1.6 FC_Application / FM_Application_OpenFont

NAME

FM_Application_OpenFont -- (00.00)

SYNOPSIS

F_Do(Obj,FM_Application_OpenFont,FObject Requester,STRPTR Spec);

FUNCTION

Open a font.

INPUTS

Requester - Pointer to the object requesting the font. This pointer is used to retreive parent font if Spec equals FV_Font_Inherit or if the method is unable to open the font defined by Spec.

Spec - Pointer to a string defining the font to open. e.g. "Garnet/16". Spec may be FV_Font_Inherit in which case Requester will be used to retreive parent's font. Spec may also be a preference item such as "FP_Font_Big" or "FP_Font_String". If all attempt to open the font fail the font of the FC_Display object will be opened instead.

RESULT

Pointer to a struct TextFont. This fonction should never fail as there is multiple fallbacks.

NOTE

The font opened by this method must be closed with CloseFont().

SEE ALSO

FM_Application_Resolve

1.7 FC_Application / FM_Application_PushMethod

NAME

FM_Application_PushMethod -- (07.00)

SYNOPSIS

F_Do(Obj,FM_Application_PushMethod,FObject Target,ULONG Method,ULONG Count,...);

FUNCTION

Usually, you may not talk to an application from two tasks at the same time. This method provides some kind of solution for this problem.

This (and only this) method may be called from a second task. It takes another method as parameter and puts in onto a private stack of the application object. The pushed method is read within FM_Application_Run and executed in the context of the current task.

This method is also very usefull to avoid "death circles". Take the example of a bad iconify gadget. When you release it, it sets FA_Application_Sleep to TRUE to iconify the application. When the application iconifies itself it closes all windows opened, including the window where the iconify gadget is. When the window closes it disposes all of its gadgets... Then, when the iconify mecanism is done the code returns to its first place : the iconify gadget released. And everything crashes because : the object does no longer exists, the class of the object may not exists neither, and the code of the class may have been trashed. This is a BAD situation. In this kind of "death circle" FM_Application_PushMethod is the method to use.

INPUTS

Inputs are similar to FM_Notify method.

Target - Object on which to perform the pushed method.

Method - Method to invoke on the target object.

Count - Number of following arguments.

... - Arguments.

RESULT

TRUE if successful, FALSE otherwise.

NOTE

FM_Application_PushMethod has a limit of 15 arguments !

SEE ALSO

FM_Application_Run

1.8 FC_Application / FM_Application_RemSignalHandler

NAME

FM_Application_RemSignalHandler -- (00.00)

SYNOPSIS

F_Do(Obj,FM_Application_RemSignalHandler,struct FeelinSignalHandler *sh);

FUNCTION

Remove an input handler.

Your trigger method is no longer called after you have removed the FeelinSignalHandler . You can add/remove input handler nodes any time as long as you know about your FC_Application object.

INPUTS

sh - input handler structure you passed to FM_Application_AddSignalHandler previously.

RESULT

FM_Application_RemSignalHandler cannot fail, the result value of the method is currently undefined.

SEE ALSO

FM_Application_AddSignalHandler

1.9 FC_Application / FM_Application_Resolve

NAME

FM_Application_Resolve -- (04.00)

SYNOPSIS

F_Do(Obj,FM_Application_Resolve,STRPTR Item,APTR Default);

FUNCTION

Resolve a preference item.

Because Feelin uses Dynamic IDs, preference items don't have any ID number, but are plain strings. e.g. "FP_String_Cursor". Each application uses a FC_Preference object to manage preferences. This method is a simple interface to the FM_Preference_Resolve method.

EXAMPLE

F_METHOD(APTR,mNew) { struct LocalObjectData *LOD = F_LOD(Class,Obj); struct TagItem *Tags = Msg, item; ... LOD -> p_BlinkSpeed = "FP_String_BlinkSpeed"; LOD -> p_Cursor = "FP_String_Cursor"; ...

while (F_DynamicNTI(&Tags,&item,Class)) switch (item.ti_Tag) { ...

/* This allows overwriting preference settings */

case FA_String_BlinkSpeed: LOD -> p_BlinkSpeed = (STRPTR)(item.ti_Data); break; case FA_String_Cursor: LOD -> p_Cursor = (STRPTR)(item.ti_Data); break; ... } ... }

F_METHOD(ULONG,mSetup) { struct LocalObjectData *LOD = F_LOD(Class,Obj);

if (F_SUPERDO()) { ULONG data;

...

if (data = F_Do(_app(Obj),FM_Application_Resolve,LOD -> p_Cursor,DEF_STRING_CURSOR)) { if (LOD -> Cursor = F_NewObj(FC_ImageDisplay,FA_ImageDisplay_Spec,data,TAG_DONE)) { F_DoA(LOD -> Cursor,FM_ImageDisplay_Setup,Msg); } }

LOD -> BlinkSpeed = F_Do(_app(Obj), FM_Application_ResolveInt ,LOD -> p_BlinkSpeed,DEF_STRING_BLINKSPEED);

...

return TRUE; } return FALSE; }

SEE ALSO

FM_Application_ResolveInt

1.10 FC_Application / FM_Application_ResolveInt

NAME

FM_Application_ResolveInt -- (04.00)

SYNOPSIS

F_Do(Obj,FM_Application_ResolveInt,STRPTR Item,ULONG Default);

FUNCTION

Resolve a preference item.

Because Feelin uses Dynamic IDs, preference items don't have any ID number, but are plain string. e.g. "FP_String_BlinkSpeed". Each application uses a FC_Preference object to manage preferences. This method is a simple interface to the FM_Preference_ResolveInt method.

SEE ALSO

FM_Application_Resolve

1.11 FC_Application / FM_Application_Run

NAME

FM_Application_Run -- (06.00)

SYNOPSIS

F_Do(Obj,FM_Application_Run);

FUNCTION

Launch the application.

Once your application tree has been successfully created, all you need to do to see all your dreams come true is to launch the application using this method.

FM_Application_Run uses the application process to handle signals, system activity and window events... This method is the heart of your application. No window will open before a call to this method.

The method won't exit until the application receives a FM_Application_Shutdown method, thus the programmer MUST encapsulate eveything into custom classes. Feelin don't support a FM_Application_ReturnID because it's a VERY BAD idea.

EXAMPLE

c = AppObject, Child, w = WindowObject, FA_Window_Title, "Feelin : Test", FA_Window_Open, TRUE,

Child, HGroup, Child, SimpleButton("Save"), Child, SimpleButton("Use"), Child, SimpleButton("Cancel"), End, End, End;

if (c) { F_Do(w,FM_Notify,FA_Window_CloseRequest,TRUE, c,FM_Application_Shutdown,0); F_Do(c,FM_Application_Run); F_DisposeObj(c); }

SEE ALSO

FM_Application_Shutdown FA_Application_Sleep

1.12 FC_Application / FM_Application_Save

NAME

FM_Application_Save -- (05.00)

SYNOPSIS

F_Do(Obj,FM_Application_Save,STRPTR Name);

FUNCTION

See FM_Application_Load .

1.13 FC_Application / FM_Application_Setup

NAME

FM_Application_Setup -- (05.00) [For use within classes only]

SYNOPSIS

F_Do(Obj,FM_Application_Setup);

FUNCTION

Not yet documented.

SEE ALSO

FM_Application_Cleanup

1.14 FC_Application / FM_Application_Shutdown

NAME

FM_Application_Shutdown -- (06.00)

SYNOPSIS

F_Do(Obj,FM_Application_Shutdown);

FUNCTION

Returns from the FM_Application_Run method.

It is safe to call this method from another task because it actually sends a message to the application to avoid "death circle".

SEE ALSO

FM_Application_PushMethod FA_Application_Sleep

1.15 FC_Application / FM_Application_Sleep

NAME

FM_Application_Sleep -- (06.00)

SYNOPSIS

[PRIVATE]

NOTE

It is safe to call this method from another task because it actually sends a message to the application to avoid "death circle".

SEE ALSO

FM_Application_Awake FA_Application_Sleep

1.16 FC_Application / FA_Application

NAME

FA_Application -- (00.00) [.G.], APTR

FUNCTION

This attribute is for general purpose. It is currently handled by FC_Window and FC_Area objects, and may be used to obtain the application these objects belong to.

1.17 FC_Application / FA_Application_Author

NAME

FA_Application_Author -- (00.00) [I.G], STRPTR

FUNCTION

Name of the application's author.

SEE ALSO

FA_Application_Base FA_Application_Copyright

FA_Application_Description FA_Application_Title

FA_Application_Version

1.18 FC_Application / FA_Application_Base

NAME

FA_Application_Base -- (00.00) [I.G], STRPTR

FUNCTION

The basename for an application. This name is used for the builtin ARexx port and for some internal file management.

A basename must neither contain spaces nor any special characters such as ":/()#?*...".

When your program is a single task application (i.e. FA_Application_SingleTask is TRUE), the base name will be used without further modification. Otherwise, it gets a ".1", ".2", etc. appended, depending on how many applications are already running. If you need to know the name of your ARexx port, you can query the base name attribute after the application is created.

SEE ALSO

FA_Application_Author FA_Application_Copyright

FA_Application_Description FA_Application_Title

FA_Application_Version

1.19 FC_Application / FA_Application_Broker

NAME

FA_Application_Broker -- (04.30) [..G], STRPTR

FUNCTION

If you need to attach some additional commodities objects to your application (e.g. because you need lots of hotkeys), you can obtain a pointer to the application's Broker structure and add some commodities objects.

The broker is completely freed when the application is disposed, no need for you to free your objects yourself.

To receive input from your objects, you will also need to install a FA_Application_BrokerHook .

NOTE

You must be prepared to receive a NULL pointer. In this case, the commodities interface is not available.

SEE ALSO

FA_Application_BrokerHook

1.20 FC_Application / FA_Application_BrokerHook

NAME

FA_Application_BrokerHook -- (04.30) [ISG], struct Hook *

FUNCTION

You specify a pointer to struct Hook. The function will be called whenever a commodities message arrives (between Feelin's GetMsg() and ReplyMsg()).

You receive a pointer to the object in A2 and a pointer to commodities CxMsg message in A1.

NOTE

If the commodities interface is not available your hook will never be called.

SEE ALSO

FA_Application_Broker

1.21 FC_Application / FA_Application_BrokerPort

NAME

FA_Application_BrokerPort -- (04.30) [..G], struct MsgPort *

FUNCTION

Get a pointer to the object's commodities message port. If you want to add own Hotkeys to your application, you need a message port. Instead of creating your own, you should better use this one.

NOTE

You must be prepared to receive a NULL pointer. In this case, the commodities interface is not available.

SEE ALSO

FA_Application_BrokerHook

1.22 FC_Application / FA_Application_BrokerPri

NAME

FA_Application_BrokerPri -- (04.30) [I.G], LONG

FUNCTION

Adjust the priority of an application's broker.

SEE ALSO

FA_Application_BrokerHook

1.23 FC_Application / FA_Application_Context

NAME

FA_Application_Context -- (00.00) [..G], APTR

FUNCTION

Returns a pointer to the FC_DisplayContext object created by the application on creation time to manage its environement.

A pointer to this FC_DisplayContext object is also available in the FC_Render object shared by FC_Area objects.

1.24 FC_Application / FA_Application_Copyright

NAME

FA_Application_Copyright -- (00.00) [I.G], STRPTR

FUNCTION

A copyright string, containing the year and the company.

SEE ALSO

FA_Application_Author FA_Application_Base

FA_Application_Description FA_Application_Title

FA_Application_Version

1.25 FC_Application / FA_Application_Description

NAME

FA_Application_Description -- (00.00) [I.G], STRPTR

FUNCTION

Short description, about 40 characters. Shown e.g. in commodities exchange.

SEE ALSO

FA_Application_Author FA_Application_Base

FA_Application_Copyright FA_Application_Title

FA_Application_Version

1.26 FC_Application / FA_Application_OBJSpace

NAME

FA_Application_OBJSpace -- (00.00) [..G], APTR

FUNCTION

Returns a pointer to a FC_Dataspace object where objects with a FA_ID non NULL can load / save their data on FM_Application_Load / FM_Application_Save .

1.27 FC_Application / FA_Application_Signal

NAME

FA_Application_Signal -- (00.00) [ISG], ULONG

FUNCTION

Allow you to react on signals.

EXAMPLE

For example lets say you want to handle Ctrl-D signal:

c := AppObject, ... FA_Application_Signal, SIGBREAKF_CTRL_D, ...

F_Do(c,FM_Notify, FA_Application_Signal,SIGBREAKF_CTRL_D, FV_Notify_Self, FM_Application_Shutdown ,0);

F_Do(c, FM_Application_Run);

F_DisposeObj(c);

...

NOTE

Ctrl-C signal is automaticaly handled by the object as a shutdown.

SEE ALSO

FM_Application_AddSignalHandler

1.28 FC_Application / FA_Application_Sleep

NAME

FA_Application_Sleep -- (00.00) [ISG], BOOL

FUNCTION

This attribute can be used to put a whole application to sleep. All windows are closed and resources freed. Depending on user settings application can be iconified or menufied (available as an item in the worbench menu).

You wake up an application :

- by a double clik on its icon (if application has been iconified). - selection the menu item in the workbench menu. - sending the ARexx command "SHOW". - using the commodity interface.

1.29 FC_Application / FA_Application_Title

NAME

FA_Application_Title -- (00.00) [I.G], STRPTR

FUNCTION

This tag defines the title of an application. The title is e.g. shown in Commodities Exchange or in the Feelin preferences program.

An application title shall not contain any version information, just the pure title. Also, special characters such as ":/()#?*..." are not allowed.

You should use a quiet long and unique name for your applications. Naming it "Viewer" or "Browser" is not a wise choice.

The length of the name must not exceed 30 characters!

EXAMPLE

AppObject, FA_Application_Title, "MyCD", FA_Application_Version, "$VER: MyCD 1.00 (01.10.01)", FA_Application_Copyright, "© 2001 - 2004 by Olivier LAVIALE", FA_Application_Author, "Olivier LAVIALE <gofromiel@numericable.fr>", FA_Application_De "A simple and groovy CD Player.", FA_Application_Base, "MYCD", ...

SEE ALSO

FA_Application_Author FA_Application_Base

FA_Application_Copyright FA_Application_Description

FA_Application_Version

1.30 FC_Application / FA_Application_Version

NAME

FA_Application_Version -- (00.00) [I.G], STRPTR

FUNCTION

Define a version string for an application. This string shall follow standard version string conventions but must not contain a leading "0".

SEE ALSO

FA_Application_Author FA_Application_Base

FA_Application_Copyright FA_Application_Description

FA_Application_Title

1.31 FC_Application / FA_Application_WindowPort

NAME

FA_Application_WindowPort -- (00.00) [..G], MsgPort *

FUNCTION

Returns a pointer to the message port shared by each FC_Window object.

1.32 FC_Application / FP_Application_ColorScheme

NAME

FP_Application_ColorScheme -- (08.00), STRPTR

PREFERENCE

This preference item holds the color scheme specifications defined by the user.

SEE ALSO

FC_Preference FC_Display / FM_CreateColorScheme

1.33 FC_Application / FP_Font_Tiny

NAME

FP_Font_Tiny -- (08.00), STRPTR

PREFERENCE

This preference item holds the tiny font specifications defined by the user.

SEE ALSO

FC_Preference FM_Application_OpenFont

1.34 FC_Application / FP_Font_Normal

NAME

FP_Font_Normal -- (08.00), STRPTR

PREFERENCE

This preference item holds the normal font specifications defined by the user.

SEE ALSO

FC_Preference FM_Application_OpenFont

1.35 FC_Application / FP_Font_Big

NAME

FP_Font_Big -- (08.00), STRPTR

PREFERENCE

This preference item holds the big font specifications defined by the user.

SEE ALSO

FC_Preference FM_Application_OpenFont

1.36 FC_Application / FP_Font_Fixed

NAME

FP_Font_Fixed -- (08.00), STRPTR

PREFERENCE

This preference item holds the fixed font specifications defined by the user.

SEE ALSO

FC_Preference FM_Application_OpenFont

1.37 FC_Application / FeelinEvent

NAME

FeelinEvent -- (06.00)

STRUCT

struct FeelinEvent { struct IntuiMessage *IMsg;

ULONG Flags;

ULONG Class; UWORD Code; UWORD Qualifier; UBYTE Key; UBYTE DecodedChar; UWORD reserved;

WORD MouseX; WORD MouseY; ULONG Seconds; ULONG Micros;

FObject Window; };

FUNCTION

All FC_Window objects use a shared message port. This port is created and handled by the application. System events and intuition events are collected and dispatched within the FM_Application_Run method. When an event occurs in a window, a struct FeelinEvent is created and sent to the window.

This is not a simple copy of intuition message. It holds very nice and usefull things not handled by intuition. For example each IDCMP_RAWKEY event is decoded as an application's key or a single char, this behaviour makes IDCMP_VANILLAKEY obsolete and is far more interresting because you can know about keys up, down and repeat, which is not possible with IDCMP_VANILLAKEY.

FIELDS IMsg

Pointer to the original struct IntuiMessage. You will rarely use this message as struct FeelinEvent holds everything necessary plus a lot of special features.

Flags

FF_Event_KeyUp - Set on IDCMP_RAWKEY when key is up.

FF_Event_Repeat - Set on IDCMP_RAWKEY when key is held down, triggering repeat events.

FF_Event_DoubleClick - Set on IDCMP_MOUSEBUTTONS when mouse clicks produced a double click. Although defined here, this flag is in fact set by the FC_Window object.

Class, Code, Qualifier

These fields correspond to IntuiMessage fields. The Class bits correspond directly with the IDCMP Flags. The Code field is for special values e.g. SELECTDOWN. And the Qualifier field is a copy of the current InputEvent's Qualifier.

Key

User have the possibility to customise application keys (page up, next word, line end...). These keys are standard definitions such as "-repeat shift left" for 'next word'. The following keys are currently mapped :

Key (mapped) Default define _____

FV_KEY_NONE <none> FV_KEY_PRESS "return" FV_KEY_RELEASE, "upstroke return" FV_KEY_UP, "-repeat up" FV_KEY_DC "-repeat down" FV_KEY_STEPUP, "-repeat shift up" FV_KEY_STEPDOWN, "-repeat shift down" FV_KEY_TOP, "alt up" FV_KEY_BOTTOM, "alt down" FV_KEY_LEFT, "-repeat left" FV_KEY_RIGHT, "-repeat right" FV_KEY_STEPLEFT, "-repeat shift left" FV_KEY_STEPRIGHT, "-repeat shift right" FV_KEY_FIRST, "alt left" FV_KEY_LAST, "alt right" FV_KEY_CHARE "-repeat backspace" FV_KEY_CHARDEL, "-repeat del" FV_KEY_WORDBACK, "alt backspace" FV_KEY_WORDDEL, "alt del" FV_KEY_LINEBACK, "shift backspace" FV_KEY_LINEDEL, "shift del" FV_KEY_NEXTOBJ, "-repeat tab" FV_KEY_PREVOI "-repeat shift tab" FV_KEY_NOOBJ, "control return" FV_KEY_CLOSEWINDOW "esc"

DecodedChar

IDCMP_RAWKEY events that do not map to application's key are, whenever possible, decoded as a single character. Thus using IDCMP_VANILLAKEY is obsolete as raw keys are always decoded. More over using this feature instead of IDCMP_VANILLAKEY allows you to know about key up, down and repeat, which is not possible with IDCMP_VANILLAKEY.

MouseX, MouseY

When getting mouse movement reports, any event you get will have the mouse coordinates in these fields. The coordinates are relative to the upper-left corner of the window in which the event occured. If IDCMP_DELTAMOVE is set, these values will be deltas from the lastest reported position.

Seconds, Micros

The time values are copies of the current system clock time. Micros are in units of microseconds, Seconds in seconds.

Window

Pointer to the FC_Window object in which the event occured.

SEE ALSO

FM_HandleEvent

1.38 FC_Application / FeelinSignalHandler

NAME

FeelinSinalHandler -- (00.00)

STRUCT

struct FeelinSignalHandler { struct FeelinSignalHandler *Next; struct FeelinSignalHandler *Prev;

ULONG Flags; FObject Object; ULONG Method;

union { struct { ULONG Signals; ULONG Reserved; } fsh_sig;

struct { ULONG Secs; ULONG Micros; } fsh_timer; } fsh_union; };

FUNCTION

FM_Application_AddSignalHandler allows classes to react on signals of private message ports. A class can create messages ports and react on their signals on its own withou interferring the main progam.

To be able to react on signals, you must fill out a struct FeelinSignalHandler (probably located in your local object data) and call FM_Application_AddSignalHandler with the structure as parameter. From now on, your class will receive the specific method whenever one of the given signal arrives.

FIELDS Next, Prev

These fields are used by the FC_Application object to link handlers, you don't have to bother with them.

Flags

FF_SignalHandler_Timer - FC_Application implements a builtin timer. By using this one instead of creating your own IO requests, you avoid the problem of having each instance of your object allocating a signal bit.

To make use of this timer, use the above described procedure of initializing and adding your FeelinSignalHandler structure, but set the FF_InputHandler_Timer in Flags. Furthermore, specify the number of secs and micros after which you want to receive

your method in fsh_Secs and fsh_Micros. Note that fsh_Secs and fsh_Micros are in fact part of a union placed at the same memory location as Signals, do not use Signals when F_InputHandler_Timer is set.

Besides this, using the timer is similiar to other input handlers. Removing with FM_Application_RemSignalHandler is not different at all.

Object

Object on which the Method will be invoked if one of the signals arrives.

Method

Method invoked on the Object.

Signals

Signals you wish to be notified on. You can set more than one bit here. This field is part of an union, use the macro fsh_Signals instead (fsh_Signals equals "fsh_union.fsh_sig.Signals").

Secs, Micros

These fields are only valid when the flag FF_InputHandler_Timer is set. They are part of an union, use the macros fsh_Secs and fsh_Micros instead (fsh_Secs equals fsh_union.fsh_timer.Secs and fsh_Micros equals fsh_union.fsh_timer.Micros).