FC_Object

Olivier LAVIALE 2004

COLLABORATORS

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

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

FC_Object

1.1 Feelin : FC_Object

FC_Object -- 03.00

IDs: Static Super: NONE Include: <libraries/feelin.h>

This class is superclass of most Feelin classes. It contains all necessary methods to create and delete objects, set or get attributes, add and remove notifications on attributes, plus some usefull methods such as FM_CallHook .

METHODS

FM_New FM_Dispose

FM_Set FM_Get

FM_Notify

FM_UnNotify

FM_CallHook FM_CallHookEntry

FM_WriteLong

FM_WriteString FM_MultiSet

FM_SetAsString

virtuals

FM_Import FM_Export

FM_Connect

FM_Disconnect

FM_AddMember FM_RemMember

ATTRIBUTES

FA_Class FA_Revision

FA_Version

FA_UserData FA_NoNotify

virtuals

FA_ID FA_Parent

FA_Child

TYPES

FObject

MACROS

_class _classname

1.2 FC_Object / FM_AddMember

NAME

FM_AddMember -- (03.00)

SYNOPSIS

F_DoA(Obj,FM_AddMember,FObject,FS_AddMember);

F_Do(Obj,FM_AddMember,FObject Child,ULONG Position,FObject Previous);

FUNCTION

This method should be used to add objects to a family.

Although defined here this method is not implemented in FC_Object, it's up to the subclass to implement or not the method.

INPUTS Child (FObject)

The object to be added. The object will be sent a FM_Connect method. The object is connected only if it the method has been replied TRUE.

Position (ULONG)

FV_AddMember_Head

Add an object as first object to the family. The 'Previous' field is not used and may be omitted.

FV_AddMember_Tail

Add an object as last object to the family. The 'Preivous' field is not used and may be omitted.

FV_AddMember_Insert

Add an object after another object to the family. The 'Previous' field is used and must be another object of the family.

Previous FObject

This parameter is only used when inserting object (with 'Position' FV_AddMember_Insert). The object must be a member of the family. If NULL, the object is added as first object to the family.

NOTE

Subclasses should always have the same implementation. On the other side, the way objects are linked is totaly free, there is no header restriction. For example FC_Group use the fields 'Next' and 'Prev' available in the FAreaData structure of FC_Area objects to link them with each other, FC_Family creates abstract nodes that may contain any kind of objects...

EXAMPLE

FObject gr = GroupObject, End; FObject o1 = AreaObject, FA_Back,"c:FF0000", End; FObject o2 = AreaObject, FA_Back,"c:00FF00", End; FObject o3 = AreaObject, FA_Back,"c:0000FF", End;

F_Do(gr,FM_AddMember,o3,FM_AddMember_Tail); F_Do(gr,FM_AddMember,o1,FM_AddMember_Head); F_Do(gr,FM_AddMem

SEE ALSO

FM_RemMember

1.3 FC_Object / FM_CallHook

NAME

FM_CallHook -- (00.00)

SYNOPSIS

F_Do(Obj,FM_CallHook,struct Hook *,...);

FUNCTION

Call a standard Amiga callback hook, defined by a Hook structure. Together with FM_Notify , you can easily bind hooks to buttons, your hook will be called when the button is pressed.

The hook will be called with a pointer to the hook structure in A0, a pointer to the calling object in A2 and a pointer to the first parameter (the hook message) in A1. You should use the macro F_HOOK() or F_HOOKM() to create hook functions.

INPUTS

Hook - Pointer to a struct Hook.

... - Zero or more parameters. The hook function will receive a pointer to the first parameter (the hook message) in register A1.

EXAMPLE

standalone:

F_Do(obj,FM_CallHook,&hookstruct,13,42,"foobar","barfoo");

within a notification statement:

F_Do(propobj,FM_Notify,FA_Prop_First,FV_Notify_Always, propobj,FM_CallHook,2,&prophook,FV_Notify_Value);

prophook will be called every time the knob is moving and gets a pointer to the knobs current level in A1.

SEE ALSO

FM_CallHookEntry

1.4 FC_Object / FM_CallHookEntry

NAME

FM_CallHookEntry -- (00.00)

SYNOPSIS

F_Do(Obj,FM_CallHookEntry,(HOOKFUNC) Func,...);

FUNCTION

This method is similar to the FM_CallHook method, but this one doesn't require a Hook structure as the function, usualy referenced in h_Entry, is called directly. Such a function should have been created with the F_HOOKM() or F_HOOKM() macros.

The function will be called with A0 being NULL (as there is no Hook structure), a pointer to the calling object in A2 and a pointer to the first parameter (the hook message) in A1.

INPUTS

Func - Pointer to a function created with the F_HOOK() or F_HOOKM() macros.

... - Zero or more parameters. The function will receive a pointer to the first parameter (the hook message) in register A1.

EXAMPLE

standalone:

F_Do(obj,FM_CallHookEntry,MyFunc,13,42,"foobar","barfoo");

within a notification statement:

F_Do(propobj,FM_Notify,FA_Prop_First,FV_Notify_Always, propobj,FM_CallHookEntry,2,Hook_Prop,FV_Notify_Value);

Hook_Prop will be called every time the knob is moving and gets a pointer to the knobs current level in A1.

SEE ALSO

FM_CallHook

1.5 FC_Object / FM_Connect

NAME

FM_Connect (02.00)

SYNOPSIS

F_DoA(Obj,FM_Connect,FS_Connect);

F_Do(Obj,FM_Connect,FObject Parent, FObject Family);

FUNCTION

This method should be used to connect a child to its parent.

Although defined here this method is not handled by FC_Object . If your object is ment to be member of a family (e.g. like FC_Area objects), you must override the method to connect your object to a family.

Using this method before linking the object to the family has several purposes :

1:. The object becomes member of the family *only* if the method returns TRUE. Only objects that may be linked will return TRUE.

2:. The 'Parent' is not set by the family into the Local Object Data (LOD) of the member. There is not _static_ place in the LOD to save the parent, it's free to the member.

3:. The member may react on the method.

RESULT

If your object is ok to connect you must return TRUE, otherwise return FALSE and your object will not be connected to the family.

EXAMPLE

F_METHODM(ULONG,Area_Connect,FS_Connect) { struct LocalObjectData *LOD = F_LOD(Class,Obj);

F_Log(FV_ERLV_USER,"Parent: %s{%08lx}", _classname (Msg -> Parent),Msg -> Parent);

_parent = Msg -> Parent;

return TRUE; }

SEE ALSO

FM_AddMember FM_Disconnect

1.6 FC_Object / FM_Disconnect

NAME

FM_Disconnect (02.00)

SYNOPSIS

F_DoA(Obj,FM_Disconnect);

FUNCTION

If you have agreed to the FM_Connect method, this method is sent to you when the family needs the object to be disconnected. (e.g. the family is about to be disposed, your object is about to be disposed).

Although defined here the method is not handled by FC_Object . If your object is ment to be member of a family (e.g. like FC_Area objects), you must override the method to diconnect your object from a family.

RESULT

If your object is ok to disconnect you must return TRUE, otherwise return FALSE and your object will not be disconnected from the family, which is not a good option.

EXAMPLE

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

F_Log(FV_ERLV_USER,NULL);

_parent = NULL;

return TRUE; }

SEE ALSO

FM_Connect FM_RemMember

1.7 FC_Object / FM_Dispose

NAME

FM_Dispose -- (00.00) [For use within classes only]

SYNOPSIS

F_SUPERDO();

FUNCTION

This method instruct an object to delete itself.

Don't ever use this method to dispose objects, instead use the feelin.library function F_DisposeObj().

For the FM_Dispose method, an object should do the following :

- Free any additionnal ressources the object explicitly allocated itself in the FM_New method. If a class does not allocate any extra ressources when it creates an object, it can defer all FM_Dispose processing to its superclass.

- Pass the message up to the superclass.

NOTE

If the attribute FA_Parent is not NULL, the parent of the object will be invoked with the method FM_RemMember . This behaviour is needed to keep things nice : an object can be disposed without worring about its parent.

All notifications requested on the object will also be removed with the FM_UnNotify method.

SEE ALSO

FM_New

1.8 FC_Object / FM_Export

NAME

FM_Export -- (00.00)

SYNOPSIS

F_Do(Obj,FM_Object,APTR Dataspace,ULONG id_Add);

FUNCTION

This method is called for each object in the application tree (including menus) during execution of FM_Application_Save. It's purpose is to export an objects "contents" to a dataspace object for later saving to an IFF file.

If you override this method to export your custom data, you are supposed to use your FA_ID as ID for the dataspace entry. Don't export if your FA_ID is 0.

RESULT

Return 0 if you dont want your superclasses to be able to import/export any values themselves. Otherwise return F_SUPERDO();

EXAMPLE

F_METHODM(ULONG,mExport,FS_Export) { struct LocalObjectData *LOD = F_LOD(Class,Obj); ULONG id;

if (id = F_Get(Obj,FA_ID)) { F_Do(Msg -> Dataspace,Msg -> id_Add, &LOD -> Value,sizeof (LONG),id); }

return 0; }

NOTE

The Dynamic ID FM_Dataspace_Add has been resolved for you and is available in the Msg as "id_Add", this may be usefull.

SEE ALSO

FM_Import FM_Application_Load

FM_Application_Save

1.9 FC_Object / FM_Get

NAME

FM_Get -- (00.00)

SYNOPSIS

F_DoA(Obj,FM_Get,struct TagItem *Taglist);

F_Do(Obj,FM_Get,...);

FUNCTION

Tells an object to report some attribute's values. Applications can call this method directly. The function F_Get() can also be used if only one attribute is requested (faster and easier). The return value for this method is not explicitly defined. Should be NULL.

INPUTS

Taglist - A TagItem array describing which attributes should be reported, and where to save values.

EXAMPLE

ULONG val,min,max;

F_Do(obj,FM_Get, FA_Numeric_Value, &val, FA_Numeric_Min, &min, FA_Numeric_Max, &max, TAG_DONE);

SEE ALSO

FM_Set

1.10 FC_Object / FM_Import

NAME

FM_Import (00.00)

SYNOPSIS

F_Do(Obj,FM_Import,APTR Dataspace,ULONG id_Find);

FUNCTION

This method is called for each object in the application tree (including menus) during execution of FM_Application_Load. It's purpose is to import an objects "contents" from a dataspace object after loading from an IFF file.

If you override this method to import your custom data, you are supposed to use your FA_ID as ID for the dataspace entry. Don't import if your FA_ID is 0.

RESULT

Return 0 if you dont want your superclasses to be able to import/export any values themselves. Otherwise return F_SUPERDO();

EXAMPLE

F_METHODM(ULONG,mImport,FS_Import) { struct LocalObjectData *LOD = F_LOD(Class,Obj); ULONG id;

if (id = F_Get(Obj,FA_ID)) { LONG *data = (LONG *) F_Do(Msg -> Dataspace,Msg -> id_Find,id);

if (data) {

/* It's a good idea to use F_Set() as the user may have set a notification on the attribute */

F_Set(Obj,F_IDA(FA_Numeric_Value),*data); } }

return 0; }

NOTE

The Dynamic ID FM_Dataspace_Find has been resolved for you and is available in the Msg as "id_Find", this may be usefull.

SEE ALSO

FM_Export FM_Application_Load

FM_Application_Save

1.11 FC_Object / FM_MultiSet

NAME

FM_MultiSet -- (00.00)

SYNOPSIS

F_Do(Obj,FM_MultiSet,ULONG Attribute,ULONG Value,...);

FUNCTION

Set an attribute for multiple objects. Receiving an attribute/value pair and a list of objects, this method sets the new value for all the objects in the list. This is especially useful for disabling/enabling lots of objects with one single function call.

The object that executes this method isn't affected !

INPUTS

Attribute attribute to set.

Value new value for the attribute.

... list of objects, terminated with a NULL pointer.

EXAMPLE

/* Disable all the addresses related gadgets */

F_Do(xxx, FM_MultiSet, FA_Disabled, TRUE, ST_Name, ST_Street, ST_ity, ST_Country, ST_Phone, NULL);

/* Note that xxx object doesn't get disabled */

SEE ALSO

FM_Set FM_Notify

1.12 FC_Object / FM_New

NAME

FM_New -- (00.00) [For use within classes only]

SYNOPSIS

F_SUPERDO();

FUNCTION

Each Feelin class keeps a record of how much memory its local object data (LOD) requires. When an object is created, using the feelin.library function F_NewObjA(), Feelin looks at the object's true class to find out how much memory to allocate for the object's data. Feelin allocates enough memory for the true class's local object data, plus enough memory for the local object data of each of the true class's superclasses. The object will be setup, then it will be invoked with the FM_New method.

The dispatcher has to initialize the instance data that is local to its class. A dispatcher finds its local object data by using the F_LOD() macro. It has to scan through the tag list of attribute/value pairs passed as argument to the FM_New method. If the dispatcher recognizes attributes it has to initialize them to the value passed in the attribute/value pair.

Then the dispatcher has to pass the method and its arguments to its superclass using the feelin.library function F_SuperDoA(). It's always a better solution to allocate memory or make further job after your superclass, this way if your superclass fails all this job won't have been made in vain, but always remember that when the FM_New method reaches FC_Object the FM_Set method is invoked on the object's true class with the same tag list as the FM_New method. This avoid each class to call their local FM_Set method but can also lead to confusion. Always check dependencies between attributes and any ressource that you may allocate.

F_SuperDoA() will return either the object's pointer or NULL if it where an error. If the object allocates any resources in this step, it must deallocates these resources later when it is disposed in the FM_Dispose method.

Finally, the dispatcher can return. When the dispatcher returns from a FM_New method, it returns a pointer to the object or NULL if it were an error. If Feelin receives a NULL from the FM_New method (indicating failure), the object will be invoked with the FM_Dispose method giving each class a chance to free resources, that may have been allocated before failure.

EXAMPLE

F_METHOD(FObject,myNew) { struct LocalObjectData *LOD = F_LOD(Class,Obj); struct TagItem *Tags = Msg, item;

LOD -> Device = "usb.device" // Default values LOD -> Unit = 0

if (F_SUPERDO()) { // All [.S.] values are already done. Check [I..] only ones.

while (F_DynamicNTI(&Tags,&item,Class)) switch (item.ti_Tag) { case FA_CD_Device: LOD -> Device = item.ti_Data; break; case FA_CD_Unit: LOD -> Unit = item.ti_Data; break; }

if (LOD -> Request = CreateIORequest(sizeof (struct MyRequest)) { return Obj; } }

return NULL; }

RESULT

Return object pointer or NULL if something goes wrong.

SEE ALSO

FM_Dispose

1.13 FC_Object / FM_Notify

NAME

FM_Notify -- (03.00)

SYNOPSIS

F_Do(Obj,FM_Notify,ULONG Attribute,ULONG Val,APTR Target,ULONG Method,ULONG NArgs,...);

FUNCTION

Add a notification event handler to an object. Notification is essential for Feelin applications.

A notification statement consists of a source object, an attribute/value pair, a target object (destination) and a notification method. The attribute/value pair belongs to the source object and determines when the notification method will be executed on the destination object.

Whenever the source object gets the given attribute set to the given value (this can happen because the user pressing some gadgets or because of your program explicitly setting the attribute with F_Set() or FM_Set), the destination object will execute the notification method.

With some special values, you can trigger the notification every time the attribute is changing. In this case, you can include the trigerring attributes value within the notification method. See below.

INPUTS Attribute

Attribute that triggers notification.

Val

Value that triggers the notification. The special value FV_Notify_Always makes Feelin execute the notification method every time when Attribute changes. In this case, the special value FV_Notify_Value in the notification method will be replaced with the value that Attribute has been set to. You can use FV_Notify_Value without restriction. You can also use FV_Notify_Toggle here. In this case, Feelin will replace TRUE values with FALSE and FALSE values with TRUE (In fact values are NOTed .e.g ~Value). This can become quite useful when you try to set "negative" attributes like FA_Disabled.

Target

Object on which to perform the notification method. Either supply a valid object pointer or one of the following special values which will be resolved at the time the event occurs:

FV_Notify_Self - Notifies the object itself. FV_Notify_Parent - Notifies object's parent. FV_Notify_Window - Notifies object's window. FV_Notify_Application - Notifies object's application.

Method

Method to apply to the Target.

NArgs

Number of following arguments to the Method. If you e.g. have a notification method that sets an attribute to a value (like in a F_Do(Obj, FM_Set, attr, val, TAG_DONE)), you have to set NArgs to 2. This allow Feelin to copy the complete notification method into a private buffer for later use.

...

Following in the arguments to the Method.

EXAMPLE

/* Every time when the user release a button (and the mouse is still over it), the button object gets its FA_Pressed attribute set to FALSE. Thats what a program can react on with notification, e.g. by opening another window. */

```
F_Do(buttonobj,FM_Notify, FA_Pressed, FALSE, winobj, FM_Set, 2, FA_Window_Open, TRUE);
```

/* Lets say we want to whow the current value of a prop gadget somewhere in a text field. FV_Notify_Value will be replaced with the current value of FA_Prop_First. */

```
F_Do(propobj,FM_Notify, FA_Prop_First,FV_Notify_Always, textobj,FM_SetAsString, 3, FA_Text,"Value is %ld !",FV_Notify_Value
```

/* Inform our application to shutdown when the user close the main window: */

```
F_Do(win, FM_Notify ,FA_Window_CloseRequest,TRUE, app,FM_Application_Shutdown,0);
```

SEE ALSO

FM_UnNotify

1.14 FC_Object / FM_UnNotify

NAME

FM_UnNotify -- (03.00)

SYNOPSIS

F_Do(Obj,FM_UnNotify,APTR Handler);

FUNCTION

Remove notification(s) attached to the object specified.

INPUTS

Handler - A pointer to a notify handler. Such a pointer is returned by the FM_Notify method. Use the pointer returned by this method to remove the notification. If Handler equals ALL (e.i -1) then all notifications attached to the object are removed.

EXAMPLE

/* e.g. In your setup method */

LOD -> nn_window_active = F_Do(win, FM_Notify ,FA_Window_Active,FV_Notify_Always, obj, FM_Set ,2,FA_Active,FV_Notify_V

/* e.g. In your cleanup method */

F_Do(win,FM_UnNotify,LOD -> nn_window_active);

NOTES

Notification handlers are always removed when the object is disposed.

SEE ALSO

FM_Notify

1.15 FC_Object / FM_RemMember

NAME

FM_RemMember -- (03.00)

SYNOPSIS

F_Do(Obj,FM_RemMember,FObject Child);

FUNCTION

Remove an object from a family.

Although defined here this method is not implemented in FC_Object, it's up to the subclass to implement or not the method. This method replaces the FM_Family_Remove obsolete method.

INPUTS FObject Child

The object to be removed. This child should have been added with the FM_AddMember method. Before removing the object, a FM_Disconnect method should be sent to it.

SEE ALSO

FM_AddMember

1.16 FC_Object / FM_Set

NAME

FM_Set -- (00.00)

SYNOPSIS

F_DoA(Obj,FM_Set,struct TagItem *Taglist);

F_Do(Obj,FM_Set,...);

FUNCTION

This method tells an object to set one or more of its attributes.

Applications can call this method directly. The function F_Set() can also be used if you only need to set one attribute, in which case the function will be faster and easier. The return value for this method is not explicitly defined, should be NULL.

When the method reaches FC_Object, object's notifications are checked and handled. To avoid notifications set the attribute FA_NoNotify to TRUE in the taglist.

INPUTS

Taglist - A pointer to a tag list of attribute/value pairs. These pairs contain the IDs and the new values if the attributes to set. The dispatcher will look through this list using utility.library function NextTagItem() and set its private attributes (not superclass attributes). Once done, it will send the same list to it's superclass.

Classes using Dynamic IDs should use the F_DynamicNTI() function instead of NextTagItem() because ti_Tag can either be a numeric value (resolved value of a Dynamic ID) or a string e.g. "FA_String_Contents".

NOTES

Most objects will redraw themselves if one of their attribute is modified, e.g. a gauge will update its level display if its FA_Numeric_Value attribute is modified.

SEE ALSO

FA_NoNotify FM_Get

FM_Notify

1.17 FC_Object / FM_SetAsString

NAME

FM_SetAsString (00.00)

SYNOPSIS

F_Do(Obj,FM_SetAsString,ULONG Attribute,STRPTR Format,...);

FUNCTION

Set a (text kind) attribute to a string. This can be useful if you want to connect a numeric attribute of an object with a text attribute of another object.

INPUTS

Attribute - Attribute to set.

Format - RawDoFmt() formating string, remember to use %ld !.

... - One or more parameters for the format string.

EXAMPLE

stand alone:

F_Do(txobj,FM_SetAsString,FA_Text, "My name is %s and I am %ld years old.",name,age);

within a notification statement:

F_Do(propobj,FM_Notify,FA_Prop_First,FV_Notify_Always, txobj,FM_SetAsString,3,FA_Text,"prop gadget shows %ld.",FV_Notify_

SEE ALSO

FM_Notify FM_Set

1.18 FC_Object / FM_WriteLong

NAME

FM_WriteLong (00.00)

SYNOPSIS

F_Do(Obj,FM_WriteLong,ULONG Value,ULONG * Address);

FUNCTION

This method simply writes a longword somewhere to memory. Although this seems quite useless, it might become handy if used within a notify statement. For instance, you could easily connect the current level of a slider with some member of your programs data structures.

INPUTS

Value - Value to write.

Address - Memory location to write the value to.

EXAMPLE

/* Let the slider automagically write its level to a variable */

static LONG level;

F_Do(slider,FM_Notify,FA_Numeric_Value,FV_Notify_Always, slider,FM_WriteLong,2,FV_Notify_Value,&level);

SEE ALSO

FM_Notify FM_WriteString

1.19 FC_Object / FM_WriteString

NAME

FM_WriteString (00.00)

SYNOPSIS

F_Do(Obj,FM_WriteString,STRPTR String,STRPTR Address);

FUNCTION

This method simply copies a string somewhere to memory. Although this seems quite useless, it might become handy if used within a notify statement. For instance, you could easily connect the current contents of a string gadget with some member of your programs data structures.

INPUTS

String - String to copy.

Address - Memory location to write the string to.

EXAMPLE

static char buffer[256];

F_Do(string,FM_Notify,F_IDR(FA_String_Contents),FV_Notify_Always, string,FM_WriteString,2,FV_Notify_Value,buffer);

NOTE

The string is copied as is, you must assure that the destination points to enough memory.

SEE ALSO

FM_Notify FM_WriteLong

1.20 FC_Object / FA_Child

NAME

FA_Child -- (03.00) [I..], FObject

FUNCTION

You supply a pointer to a previously created object here. This child object will be added to the parent object at creation time. Of course you can specify any number of child objects, limited only by available memory.

Normally, the value for a FA_Child tag is a direct call to another F_NewObjA(), children are generated on the fly.

When a parent object is disposed, all of its children will also get disposed. If one of the FA_Child attributes found in the taglist is NULL, the parent object fails to create and, before returning, dispose all the valid objects found in the taglist.

This behaviour makes it possible to generate a complete family within one single (but long) F_NewObjA() call. Error checking is not necessary since every error, even if it occurs in a very deep nesting level, will cause the complete call to fail without leaving back any previously created object.

NOTE

Although defined here, this attribute is not handled by the class itself, this is left to subclasses. In other words a class who wish to use this attribute must be sure that the attribute is handled by one of its superclass or must handle the attribute itself (if none of its superclass does).

Check your superclass(es) documentation to know if it handles this attribute.

SEE ALSO

FM_AddMember FM_RemMember

1.21 FC_Object / FA_Class

NAME

FA_Class -- (00.00) -- [..G], struct FeelinClass *

FUNCTION

Returns a pointer to the struct FeelinClass of the object. This attribute is quite obsolete as the macro _class will do the same faster.

SEE ALSO

FA_Revision FA_Version

1.22 FC_Object / FA_ID

NAME

FA_ID -- (00.00) [ISG], ULONG

FUNCTION

Objects with a non NULL FA_ID exports their contents during FM_Application_Save and import them during FM_Application_Load .

You have to use differents IDs for your objects !

NOTE

Although defined here, this attribute is not handled by the class itself, this is left to subclasses. The attribute is not saved in the local object data (LOD) of FC_Object. If a subclass need it, the subclass must save the attribute in its own LOD and handle FM_Set & FM_Get .

In other words a class who wish to use this attribute must be sure that the attribute is handled by one of its superclass or must handle the attribute itself (if none of its superclass does).

Check your superclass(es) documentation to know if it handles this attribute.

SEE ALSO

FM_Import FM_Export

1.23 FC_Object / FA_NoNotify

NAME

FA_NoNotify -- (01.00) [.S.], BOOL

FUNCTION

If you set up a notify on an attribute to react on user input, you will also recognize events when you change this attribute under program control using F_Set() or FM_Set }. Setting FA_NoNotify together with your attribute will prevent this notification from being trigerred.

NOTE

FA_NoNotify is a one time attribute. It's only valid during the current F_Set() or FM_Set method. This works also with multiple attributes, if FA_NoNotify is found TRUE in the taglist, no attribute will trigger notification.

1.24 FC_Object / FA_Parent

NAME

FA_Parent -- (00.00) [..G], FObject

FUNCTION

Get a pointer to the parent object of the current object.

The parent receives the FM_RemMember method before the object is deleted.

NOTE

Although defined here, this attribute is not handled by the class itself, this is left to subclasses. The attribute is not saved in the local object data (LOD) of FC_Object . If a subclass need it, the subclass must save the attribute in its own LOD and handle FM_Set & FM_Get .

In other words a class who wish to use this attribute must be sure that the attribute is handled by one of its superclass or must handle the attribute itself (if none of its superclass does).

Check your superclass(es) documentation to know if it handles this attribute.

SEE ALSO

FM_Dispose

1.25 FC_Object / FA_Revision

NAME

FA_Revision -- (00.00) [..G], ULONG

FUNCTION

Get the revision number of an object's class. Although FA_Revision is documented at object class, you will of course receive the revision number of the object's true class.

NOTE

This attribute is only useful with external classes.

SEE ALSO

FA_Class FA_Version

1.26 FC_Object / FA_UserData

NAME

FA_UserData -- (01.00) [ISG], APTR

FUNCTION

A general purpose value to fill in any kind of information.

1.27 FC_Object / FA_Version

NAME

FA_Version -- (00.00) [..G], ULONG

FUNCTION

Get the version number of an object's class. Although FA_Version is documented at object class, you will of course receive the version number of the object's true class.

NOTE

This attribute is only useful with external classes.

SEE ALSO

FA_Class FA_Revision

1.28 FC_Object / FObject

NAME

FObject -- (00.00)

STRUCT

typedef void * FObject;

FUNCTION

FObject is an abstract definition.

1.29 FC_Object / _class

NAME

_class -- (00.00)

DEFINE

```
#define _class(o) ((struct FeelinClass *)(((ULONG *)(o))[-1]))
```

FUNCTION

Use this macro to obtain a pointer to the class of an object. If the object is a subclass of FC_Object , you can also use the FA_Class attribute to obtain this pointer.

SEE ALSO

_classname

1.30 FC_Object / _classname

NAME

_classname -- (00.00)

DEFINE

```
#define _classname(o) (_class(o) -> Name)
```

FUNCTION

Use this macro to obtain a pointer to the name of the class of an object.

EXAMPLE

```
F_Log(0,"Obj 0x%08lx of Class '%s' (0x%08lx)",Obj,_classname(Obj),_class);
```

SEE ALSO

_classname