in

COLLABORATORS			
	TITLE : in		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 31, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

in

1.1 methods.guide

```
 Search
TABLE OF CONTENTS

methods/--background--
methods/GRM_DIMENSIONS
methods/WM_KEYACTIVE
methods/WM_KEYINACTIVE
methods/WM_KEYINPUT
```

1.2 methods/--background--

```
    DESCRIPTION
  Gadget  classes  that  want to work in a BGUI environment will need to
  know about a set of extra methods on top of the  normal system  gadget
  class methods. This document describes these methods.

  Not    all    system  gadgetclass    methods    will    reach    your
  class.  The following standard   system   gadgetclass   methods  are
  passed onto your class:

    GM_HITTEST
    GM_RENDER
    GM_GOACTIVE
    GM_HANDLEINPUT
    GM_GOINACTIVE

  The  following  methods  are  not used in a BGUI context and therefore
  will not be send to your class:

    GM_HELPTEST
    GM_LAYOUT
```

1.3 methods/GRM_DIMENSIONS

```
  CLASS
groupclass

  NAME
GRM_DIMENSIONS

  FUNCTION
To inquire about a gadget object it's minimum  width and  height.  The
group class sends out this method  to all  it's  members to  ensure  a
correct  layout.   This  method  uses  the  following  custom  message
structure:

struct grmDimensions {
  ULONG     MethodID; /* GRM_DIMENSIONS */
  struct GadgetInfo      *grmd_GInfo;
  struct RastPort        *grmd_RPort;
  struct {
    UWORD        *Width;
    UWORD        *Height;
  }     grmd_MinSize;
  ULONG     grmd_Flags;
};

grmd_GInfo -- This  field  will  always  read  NULL!  It will probably
  become obsolete in one of the next versions.   Please  do  not
  make any assumptions about it's contents.   Simply  ignore  it
  until further notice.

grmd_RPort -- This points to a RastPort which can be  used  to perform
  text width/height computations etc. in.   Do  *not*  render in
  this RastPort.

grmd_MinSize --  This  field  contains  two pointers in which you must
  store the results of your computations.   Note  that you  must
  *add* your results to the results you got from the superclass.

  Example:

  switch ( msg->MethodID ) {
      case    GRM_DIMENSIONS:
    /*
    **  First the superclass...
    **/
    DoSuperMethodA( class, object, msg );
    /*
    **  Compute your minimum sizes.
    **/
    ...
    /*
    **  Add results.
    **/
    *( msg->grmd_MinSize.Width  ) += your_min_width;
    *( msg->grmd_MinSize.Height ) += your_min_height;
    break;
  }
```

There might be cases in which you want to overide the superclass results which is perfectly legal to do but you should be aware that wrong values here might seriously screw up the look of the resulting GUI.

grmd_Flags -- This field may contain any of the following flags:

GDIMF_NO_FRAME -- This will tell the baseclass not to take the attached frame into consideration when computing the minimum size.

1.4 methods/WM_KEYACTIVE

CLASS
windowclass

NAME
WM_KEYACTIVE

FUNCTION
To tell the object that it is activated by a key-press. Uppon receiving this message you can setup any additional resources you may need to go active. This method uses the following custom message structure:

```
struct wmKeyInput {
  ULONG      MethodID; /* WM_KEY_ACTIVE */
  struct GadgetInfo *wmki_GInfo;
  struct InputEvent *wmki_IEvent;
  ULONG     *wmki_ID;
  STRPTR       wmki_Key;
};
```

wmki_GInfo -- This points to a GadgetInfo structure.

wmki_IEvent -- A pointer to a InputEvent structure which is the event that triggered the activation. The event class is always IECLASS_RAWKEY. This event can be used to check for qualifier keys etc.

wmki_ID -- In this field you can store the ID of the object when the activation has resulted in a change that needs to be notified. The value put in here is returned by the windowclass it's WM_HANDLEIDCMP method.

wmki_Key -- This points to the key string which has been assigned to the object with the windowclass it's WM_GADGETKEY method.

RESULT
This method should return any of the following return codes:

WMKF_MEACTIVE -- The object can go/remains active.
WMKF_CANCEL -- The keyboard activation is cancelled.
WMKF_VERIFY -- The keyboard activation is complete and the ID set in the wmki_ID field is notified.

WMKF_ACTIVATE -- Returning this tell's the windowclass to activate the gadget using the intuition.library it's ActivateGadget() call.

SEE ALSO
windowclass/WM_KEYINPUT, windowclass/WM_KEYINACTIVE, windowclass/WM_KEYINACTIVE windowclass/WM_HANDLEIDCMP, windowclass/WM_GADGETKEY, intuition.library/ActivateGadget()

1.5 methods/WM_KEYINACTIVE

CLASS
windowclass

NAME
WM_KEYINACTIVE

FUNCTION
When the key-activation of an object is done or aborted by some other event this method is called to tell to object to go inactive. This gives you the oppertunity to release the resources that you might have obtained with the WM_KEYACTIVE method.

RESULT
No return code defined.

SEE ALSO
windowclass/WM_KEYACTIVE

1.6 methods/WM_KEYINPUT

CLASS
windowclass

NAME
WM_KEYINPUT

FUNCTION
This method is send to the object continually when the WM_KEYACTIVE returned WMKF_MEACTIVE and the object has gone active. This method uses the same custom message structure as the WM_KEYACTIVE method does and it should return any of the same return codes as described in the WM_KEYACTIVE method with the exception of WMKF_ACTIVATE.

A good example of this method is the buttonclass which uses this method to scan for the SHIFT qualifier and/or the ESC key which both cancel a keyboard button selection.

SEE ALSO
windowclass/WM_KEYACTIVE