ReActor

Copyright © Copyright1999 by HAAGE & PARTNER Computer GmbH

COLLABORATORS

	TITLE : ReActor	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 18, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ReActor

1.1 ReActor

ReActor - A resource construction toolkit for BOOPSI

© 1999 by HAAGE & PARTNER Computer GmbH

Overview What is ReActor? Installation How to install? Main The main window Gadget Groups The gadget group window Attributes The attributes window Profiles Class descriptions Configuration Changing profiles for your needs Making it work How to use the GUI in your programs Tutorial Making a GUI Support You have questions?

This is a beta release. This documentation is only for beta purposes and not finished.

KNOWN BUGS

1. Removing an object that is used by another one (for example a label image that is used by a button) may lead to an incorrect data structure. The test window may not be openable anymore. You must remove the tag that is adressing the removed object by hand.

2. Images that are used by other images must stay upper in the tree.

3. The Raw String buffersize is not supported yet, the string buffer is always as big as the length of the string.

4. Pointer to empty strings are replaced with NULL in the resource file but a locale message identifier is still written. Point to empty localized strings should be written.

1.2 Overview

ReActor is a tool to provide easy creation of GUIs (Graphical User Interfaces). It is directly connected to BOOPSI Classes (see profiles), so you need some knowledge of BOOPSI Classes and how they are initialized and how they work together to make good use of ReActor.

The BOOPSI Classes are created pretty much the same way as one would create them in a programming environment. You create an Object by specifing its Tags. The Tags should be self-explanetory. Detailed Information about how the Tags work can be found in the Reaction autodocs.

The GUI will be saved as a resource file, a linkable object-file (.o) and along with it a catalog-description-file (.cd) is created, as well as an C-header-file (.h). You will need a special version of catcomp to make localization work.

See Making it work.

1.3 Installation

No special installation is needed. Simply move the ReActor directory to a location of your choice. To work properly ReActor needs correct profiles, which reside in the profiles directory.

REQUIERMENTS: wizard.library appmanager.library resource.library (OS 3.5) Reaction classes (OS 3.5)

To use the Resources created with ReActor in your own code you will need a special version of catcomp to create correct assembler files, which contain references to the strings inside the object file created by ReActor for localization purpose.

1.4 Main Window

The main window contains many lists on different pages. Every list contains one type of objects: Windows, Gadget Groups, Images, Interconnections, Requesters and all the rest.

Windows, Images, Interconnections and Requester and Misc. Objects are handled indentically: Select one of the classes from the Add » popup menu. A new object of this class will be created. The Rem button deletes an object, the Up and Down buttons move the objects in the list.

The Object Name string field lets you change the name of the object.

Selecting an object from the list with double-clicking opens the Attributes Window.

The window page contains the Test button. Pressing this button opens the window. To see something in your window, you first have to connect it to your gadgetgroup.

The Gadget Groups page differ from the other pages in that you will not create an object immediatly but an gadget group. Press the Add button for a new gadget group. The Gadget Group Window will be opened for the new group.

Rem, Up, Down and Object Name works like the buttons in the other pages.

1.5 Gadget Groups

The gadget group window contains two lists: A hierarchical list containing the objects and a smaller list containing the classes. You can create an object by selecting a class and pressing the Add button or by double-clicking the class name.

The Rem button deletes an object.

The Up and Down button moves an object in the tree. An object will be moved into a group (an object that can contain other objects) if the group is opened and allows to add that type of moved object.

Typically a gadget group contains one root layout gadget and a couple of children (which can be layout groups or other objects).

Note: Every gadget is a group. But only layout gadgets ("Layout" and "Page") allows to add gadgets as children. Normal gadgets ("Button", "String" etc) only accepts images and interconnections objects as children. You should only add that images and interconnections as children that are used by the parent class.

1.6 Attributes

The attributes window contains 2 lists and free space for additional gadgets which are shown in dependency from the selected tag.

The left list contains the possible tags of the object. Add a tag to the right list by selecting an tag and double-clicking it or pressing the Add button. Every gadget can be added one or more times.

Select a tag from the middle list to change its value. In the right space of the window differen gadgets will be shown in dependency from the type of the tag. Possible tag types are string, integer, file selection, directory selection, selection from cycle gadgets, list of flags, boolean, gadget-list, image-list, object-list, interconnection maps, member list etc.

Localized strings have one string gadget for the value and one for the locale id. If you don't fill the locale id or use "---" as locale id no locale id entry is generated for that string.

The tag name in the middle list is highlighted if the tag is enabled. double-click the tag to enable it. Only enabled tags are copied into the tag lists which are used to create the objects.

1.7 Profiles

Every class is described by one or more profiles. A profile contains commands and tag descriptors. The profiles resides in different subdrawers of the drawer "profiles". Every class tree (Windows, Gadgets, Layout gadgets etc) has its own filename suffixes:

Class type Directory Suffix

Window profiles/windows .window Gadget profiles/gadgets .gadget Layout Gadgets profiles/gadgets .group and .child Images profiles/images .images Interconnections profiles/models .ic Models (IC Groups) profiles/models .model Requesters profiles/requesters .req Misc. Objects profiles/misc .misc ClassAct Lists profiles/gadgets .node Arrays ---- ----

A profile starts with the name of the profile, followed by commands and tag desriptions and end with the command END.

The flags is a string of chars that are interpreted case-insensitiv:

'I' or 'i': User can add this tag to taglist. 'S' or 's': User can add this tag to taglist. 'G' or 'g': Reserved. 'N' or 'n': Tag will be send by object in notifications. 'U' or 'u': Tag can be updated from notifications. 'A' or 'a': Tag will be added automatically to taglist. 'F' or 'f': Tag cannot be removed from taglist. 'W' or 'w': Value is size of UWORD (for arrays). 'B' or 'b': Value is size of UBYTE (for arrays).

The flag 'I' and 'S' should be differ if the tag is used in creation of object or set right after all objects are created. Currently the tagtypes itself makes this decision (Target and Object pointers are set after the objects are created).

Example:

space.gadget LABEL "Space" LIBRARY "gadgets/space.gadget" 0 PRIVATE INCLUDE "classact.baseclass" INTEGER 0x8509001 (I····) SPACE_MinHeight "Minimum Height" 0 0 32767 INTEGER 0x8509002 (I····) SPACE_MinWidth "Minimum Width" 0 0 32767 BOOLEAN 0x8509003 (I····) SPACE_Transparent "Transparent" FALSE "Transparent" IBOX 0x8509006 (I····) SPACE_AreaBox "Area Box" END

Reference of commands:

LABEL <label>

Set a name for the class. This label will be used in the user interface for this class.

INCLUDE <filename>

Includes the filename <filename>. The classname, the LABEL and LIBRARY commands are ignored in the INCLUDE file. This allows to reuse existing profiles (like inheritance).

LIBRARY <libraryname> <version> <PUBLIC|PRIVATE|LVO <lvo>>

Defines the name of the shared library which contains the class. This library will be opened before creating an object.

PUBLIC: The class is a public class. The name of the profile (first line of file) will be used as the class name.

PRIVATE: The class is private. The class pointer is fetched calling the library function at LVO -30.

LVO <lvo>: The class is private. The class pointer is fetched calling the library function at LVO <lvo>. This is used for unusual library layouts (the Reaction page.gadget for example).

NOTIFY <ALL|MARKED>

ALL: Marks a class as interconnection class which accepts every tag as source tag in the ICA_MAP.

MARKED: Only tags which marked as "Notify" of this class will be presented as source tag for ICA_MAP.

CHILDS <filename>

A filename of a profile that is used for the child tags for layout groups. Child tags are tags which can be added to every single object that is child of the layout object.

SPECIALISATION

The profile describes a specialisation of a more general class.

PARENTPROFILE <labelname>

The class is sorted into a list of subclasses or specialisation of the named parent class. This tree is used to select a class in thge gadget group window.

PRIORITY <priority>

The priority of the class in the list of subclasses.

REM

The rest of the line is ignored.

END

End of profile.

Tag descriptions:

Every tag description starts with the type, followed by the tag number, flags, and the name of the tag. The rest is type dependend: Most types expects a title (used in the user interface) as short description of the tag and default values.

STRING <tag> <flags> <name> <title> <default string>

A text string. Every string will be localized.

INTEGER <tag> <flags> <name> <title> <default value> <min> <max>

An integer.

PERCENT <tag> <flags> <name> <title> <default value>

A percantage value (min 0 %, max 100 %).

BOOLEAN <tag> <flags> <name> <title> <FALSE|TRUE> <label>

A boolean. <label> is string which is displayed right of the checkbox.

FLAGS <tag> <flags> <name> <title> <count> <def value> [flag describtions]

A list of flags. The user can enable and disable single flags. The flag values are 'or'ed together into one 32 bit number.

<count> defines the number of flag describtions. Every flag description contains a value, a mask and a name of the flag. If the flag is enabled every other flag that is part of the mask is disabled.

SELECT <tag> <flags> <name> <title> <count> <def value> [select descriptions]

A list of selections. The user can select one of the selections. The values are integers.

<count> defines the number of selections. Every select description contains a numeric value and a name for that selection.

SCREEN <tag> <flags> <name>

The screen pointer.

DRAWINFO <tag> <flags> <name>

The DrawInfo pointer.

ICTARGET <tag> <flags> <name> <title>

An gadget, image or interconnection object.

ICMAP <tag> <flags> <name> <title>

A map of tags. The user can add tags that are flagged for "Notify" as source tags and tags that are flagged for "Update" as target tags.

GADGET <tag> <flags> <name> <title>

A gadget pointer.

IMAGE <tag> <flags> <name> <title>

An image pointer.

TWOPENS <tag> <flags> <name> <title> <default pen1> <default pen2>

Two logical pens (TEXTPEN, SHINEPEN, FILLPEN etc) that are 'or'ed into one word.

PEN <tag> <flags> <name> <title> <default pen>

A logical pen (TEXTPEN, SHINEPEN, FILLPEN etc).

IBOX <tag> <flags> <name> <title>

Pointer to struct IBox.

TEXTATTR <tag> <flags> <name> <title>

Pointer to struct TextAttr.

ICMAPTAG <tag> <flags> <name> <title>

A special tag to use in interconnection classes. It allows to map a tag value into the IDCMP message code field.

MEMBER <tag> <flags> <name> <title>

A pseudo tag that is converted to the object method OM_ADDMEMBER.

FILE <tag> <flags> <name> <title>

A file name.<{PAR}

DIR <tag> <flags> <name> <title>

A directory name.

GADGROUP <tag> <flags> <name> <title>

The user selects one of the gadget groups. Only the first gadget on top level of the group hierarchy and all its children will be created. The pointer to the first gadget on top level will be used as tag value.

LCLABEL <tag> <flags> <name> <title>

A Reaction layout group label.

PATTERN <tag> <flags> <name> <title>

Reserved.

TEXT <tag> <flags> <name> <title> <default string>

A multiline text string. This string will be localized.

STRARRAY <tag> <flags> <name> <title>

An array of string pointers. The user enters a multiline text but every line is translated into a single string.

PENMAP <tag> <flags> <name> <title> <tag2>

The user enters a name of a picture file. This file is loaded with picture.datatype. The first tag gets a pointer to a ClassAct chunky map and the second tag gets a pointer to a ClassAct palette.

CHAR <tag> <flags> <name> <title> <default string>

A char. Note: The default value is a string.

LIST <tag> <flags> <name> <title> <keyword>

A Reaction list. The <keyword> names one of the possible Reaaction lists. These lists are used from some gadgets (Chooser, Clicktab, ListBrowser, Radiobutton and Speedbar).

ARRAY <tag> <flags> <name> <title> <array description> <size> <entries>

The <array description> is a filename of a profile that is loaded to describe the array entries. The tag numbers of this profile are interpreted as offsets into an array entry.

<size> is the size of one array entry in bytes.

<entries> is currently unused. It defines the minimum size of the array.

1.8 Configuration

Changing a profile for your needs:

1. Load a profile in a text editor. Save it under a different name into the same directory. Don't change the suffix of the name.

2. Change the LABEL.

3. For tags you want to add automatically to the taglist add flag 'A'.

4. Change the default values for the tags. If an BOOLEAN tag is added to the taglist its value is automatically inverted: FALSE gets automatically TRUE. So don't change the default value for BOOLEAN tags.

5. Save the profile.

Don't forget to copy the .child profiles if you make a new layout or page profile.

1.9 Making it work

To use the GUIs created with ReActor in your programs you have to:

1. Add the object-file to your project

2. Include the C-header-file in your sources (in order to refer to the objects)

3. Generate an assembler-file from the catalog-descriptor-file (with a special version of catcomp) and adding this file to your project.

4. Open resource.library in your program and initialize your objects.

Here is an example:

#include "test.h"

#include <intuition/gadgetclass.h> #include <intuition/classusr.h> #include <workbench/startup.h> #include <wbstartup.h> #include <classes/window.h> #include <classes/arexx.h>

#include <clib/intuition_protos.h> #include <clib/exec_protos.h> #include <clib/alib_protos.h> #include <clib/resource_protos.h> #include <clib/locale_protos.h> #include <clib/icon_protos.h> #include <clib/dos_protos.h> #include <clib/gadtools_protos.h> #include <stdlib.h>

struct Library *IntuitionBase; struct Library *LocaleBase; struct Library *IconBase; struct Library *GadToolsBase; struct Library *ResourceBase;

struct Screen *GlbScreenP; struct Catalog *GlbCatalogP; struct MsgPort *GlbIDCMPortP; struct MsgPort *GlbAppPortP; RESOURCEFILE GlbResourceP; Object *GlbWindowP; struct Window *GlbIWindowP; struct Gadget **GlbGadgetsP;

void loop(void) { ULONG windowsignals, waitsigs, mask, result, code; BOOL closeF;

GetAttr(WINDOW_SigMask,GlbWindowP,&windowsignals); waitsigs = windowsignals; closeF = FALSE; while(!closeF) { mask = Wait(waitsigs); if(mask & windowsignals) { while((result = DoMethod(GlbWindowP, WM_HANDLEINPUT, &code)) != WMHI_LASTMSG) { switch (result & WMHI_CLASSMASK) { case WMHI_CLOSEWINDOW: closeF = TRUE; break;

case WMHI_GADGETUP: /* Note: we use RL_GADGETMASK instead of WMHI_GADGETMASK * because ReActor ors the gadget ID and the group ID to * the final gadget ID. If you want to know in which group * the gadget was use (result & RL_GROUPMASK). */ switch (result & RL_GADGETMASK) { default: break; } break;

case WMHI_ICONIFY: DoMethod(GlbWindowP, WM_ICONIFY); GetAttr(WINDOW_Window,GlbWindowP,(ULONG *) &GlbIWindowP); break;

case WMHI_UNICONIFY: DoMethod(GlbWindowP, WM_OPEN); GetAttr(WINDOW_Window,GlbWindowP,(ULONG *) &GlbIWindowP); break; } } } } }

void run(void) { GlbRC = 20; GlbCatalogP = OpenCatalogA(NULL,"test.catalog",NULL); if(GlbScreenP = LockPubScreen(NULL)) { if(GlbIDCMPortP = CreateMsgPort()) { if(GlbAppPortP = CreateMsgPort()) { if(GlbResourceP = RL_OpenResource(RCTResource,C { if(GlbWindowP = RL_NewObject(GlbResourceP,WIN_1_ID, WINDOW_SharedPort,GlbIDCMPortP, WINDOW_AppPort,GlbAppP

TAG_END)) { ULONG error; GlbGadgetsP = (struct Gadget **) RL_GetObjectArray(GlbResourceP,GlbWindowP,GROUP_2_ID); DoMethod(GlbWindowP,WM_OPEN); GetAttr(WINDOW_Window,GlbWindowP,(ULONG *) &GlbIWindowP); loop(); DoMethod(G GlbRC = 0; } RL_CloseResource(GlbResourceP); } DeleteMsgPort(GlbAppPortP); } DeleteMsgPort(GlbIDCMPortP); } UnlockPubScreen(NULL,GlbScreenP); } }

void closelibs(void) { CloseLibrary(ResourceBase); CloseLibrary(GadToolsBase); CloseLibrary(IconBase); CloseLibrary(LocaleBase) CloseLibrary(IntuitionBase); }

BOOL openlibs(void) { if(!(IntuitionBase = OpenLibrary("intuition.library",39))) return FALSE; if(!(LocaleBase = OpenLibrary("locale.library",39))) return FALSE; if(!(IconBase = OpenLibrary("icon.library",39))) return FALSE; if(!(GadToolsBase = OpenLibrary("gadtools.library",39))) return FALSE; if(!(ResourceBase = OpenLibrary("resource.library",0))) return FALSE; return TRUE; }

// CLI start

void main(void) { if(openlibs()) { run(); } closelibs(); exit(GlbRC); }

// StormC WB start

void wbmain(struct WBStartup *wb) { if(openlibs()) { run(); } closelibs(); }

Explanation of Symbols:

WIN_1_ID: This is the ID of a window object, as you have named it. (WIN_1_ID is the default name)

GAD_2_ID: This is the ID of a gadget group, as you have named it. (GAD_2_ID is the default name)

RCTResource: This is your GUI, as it is saved in the object-file. This variable is defined in the C header-file as extern char RCTResource[].

GlbResourceP: This is your GUI object, as you refer to it when using resource.library routines.

ResourceBase: This is (of course) the librarybase of resource.library

RL_OpenResource, RL_CloseResource, RL_NewObject, RL_GetObjectArray: These are routines of resource.library. See resource Autodocs for datails.

1.10 Tutorial

Now we will build a GUI step by step using ReActor.

ReActor Tutorial - create a GUI with pages

Getting started Setting up everything. Creating some objects The main stuff. Creating page objects Some special things.

Tips Hints about some objects

And if you want to make your GUI alive read how to make it work.

1.11 Getting started

Start ReActor by doubleclicking on its icon. The main window will open. switch to the page Windows. Add an Application Window and then add two gadget groups in the Gadget Groups page. Now doubleclick on the Windowobject you have created. The Attributes window will open. The first tag in the objects Taglist (the right listview) is WA_Title. Click on it and enter a Window Title. For Example: Test

The last tag in this list is called WINDOW_Layout. Click on it. Now a new listview will appear, where you can choose beetween your gadget groups. Here it is GROUP_2_ID and GROUP_3_ID. Select GROUP_2_ID. This tag is very important, because it attaches the Layout you create with your gadget groups to the window. Otherwise you won`t see anything.

Now we will add some tags. Select the WINDOW_IconTitle from the leftmost listview and doubleclick on it or press the Add button in the upper left corner. The tag is added to the objects taglist. Select the new inserted tag and enter a name for it. For Example: Test.iconified

Press the Use button to save your changes to the object. Now would also be a great time to save the whole resource, you are creating. Do this frequently, because ReActor might crash sometimes. Remember that this is a BETA version. Choose Save from the Project Menu, or press Amiga-S. Enter your GUIs name. For Example: test.res

We created a window and some gadget groups, in order to make our GUI usefull we now will add some objects..

1.12 Creating some objects

When you added your gadget groups, a window for each gadget group has opened, titled with the name of the gadget group. So in the Main Window you can now switch to the Windows page, where the Test button is.

A gadget group normaly consists of a hierarchical list of layout groups, whose children are the gadgets. The first object you add to your gadget group normaly is the Root layout object.

Activate the window titled GROUP_2_ID and doubleclick on Root layout in the right list. Now the fun begins.

In our Example we only need a Vertical Layout group and a Horizontal one. The vertical group will contain a page group and a Clicktab gadget, so the user can choose the page, and the horizontal layout group will consist of two buttons. Therefor the Horizontal Layout group will be a Button Layout group. Here we go in 5 steps:

1. Add a Vertical Layout group
2. Add a Clicktab object
3. Add a Button Layout group
4. Add first Button
5. Add second Button

Now doubleclick on the first button. The Button Attributes window opens. There we add the GA_Text tag and enter for example: _Use

The underscore marks the Key shortcut. The keyhandling is done automatically by the Class-Act classes. Press Use so that the changes aplly to the object. The second button will be labeled for example: _Cancel

Now you may press the Test button in the Main Window and view your GUI. If you want to have direct access to an object from within your program, you should not forget to name your objects. We will proceed with our page gadget now.

1.13 Creating page objects

Page gadgets are created as own gadget groups. Their root layout object is a page object.

So activate the gadget group window titled GROUP_3_ID and add a page gadget as root layout gadget. The page objects, are layout objects, so they are described in the layout.gadget autodocs.

Each (direct) child of a page object is a own page. We want 3 pages so we add 2 vertical layout objects and one horizontal layout object to the page object.

In the first vertical layout group we will just add a palette gadget. In this palette gadget we will add the PALETTE_NumColors and PALETTE_Color tags. For Example: PALETTE_NumColors = 32 PALETTE_Color = 0

This was our first page.

In the second page we now add in this order:

Objectname Tags Value

Label LABEL_JUSTIFICATION Left LABEL_Text "Percent:"

Fuelgauge FUELGAUGE_Min 0 FUELGAUGE_Max 50 FUELGAUGE_Level 20 FUELGAUGE_Orientation Horizontal FUELGAUGE_Percent TRUE CHILD_MinHeight 20

String STRINGA_MaxChars 155 STRINGA_TextVal "A Chain of chars "

Label LABEL_Text "Enter stuff:"

Space CHILD_WeightedHeight 100

We connect the label to the String gadget and add the CHILD_Label = (Label 8) tag to the String gadget. This was page 2.

In page 3 we will now add a Listbrowser and later we will do some fancy stuff with it. But first we have to connect our clicktab gadget in GROUP_2_ID to our page object.

Activate GROUP_2_ID window and doubleclick on the Clicktab object. Add a tag CLICKTAB_PageGroup = GROUP_3_ID

Now we will create the labels for our page gadget. These labels are an Exec list with a special node structure. Some objects have these special Nodes. In your program these Nodes have to be allocated with the corresponding AllocXXXNode Routine. So do not forget to open the gadget as a shared library to have access to these functions. For detailed Information see the Class-Act autodocs.

Add the tag CLICKTAB_Labels. And then add 3 nodes. Edit the nodes by doublclikcking on them as follows:

Node Tags Value

First node TNA_Text "Page 1" TNA_Number 0

Second node TNA_Text "Page 2" TNA_Number 1

Third node TNA_Text "List" TNA_Number 2

Now again you can enter the Test Gadget in the Main Window and your page object should work.

Save your work again and start your StormC compiler...

As we have reached the end of this tutorial you may want some tips.

1.14 Tips

Tip 1:

When you want to create Multicolumn Listviews (Listbrowser) you have to make a ColumnInfo structure. For each column add a ColumnInfo node. And then add one more node with ci_Width set to -1 .

When you add Labels then to your Listbrowser you first must add a LBNA_Column tag and then the tags that you want to apply to that column. Usually you may want a LBNCA_Text tag. So here is an example of a 2 column Listbrowser:

Tags Value

LISTBROWSER_ColumnInfo node 1: ci_Title "First" ci_Width 50 ci_Flags Draggable

node 2: ci_Title "Second" ci_Width 50 ci_Flags Draggable

node 3: ci_Width -1

LISTBROWSER_Labels <node> LBNA_Column 0 LBNCA_Text "This is spread" LBNA_Column 1 LBNCA_Text "over 2 columns" : : .

LISTBROWSER_ColumnTitles TRUE

CHILD_WeightedHeight 100

Tip 2:

If you want to create a button that uses a complex label.image object as its label move embedded images (penmaps or glyphs) before the label image. The embedded images must be created before the label image.

Tip 3:

When you want to use images in your GUI, use the Penmap class, as it needs no extra palette Information. But be carefull, you have to add the PENMAP_Screen tag, otherwise the class will CRASH. This is not a bug of ReActor.

1.15 Support

If you have questions please write an email to

reactor-support@haage-partner.com

Our address

HAAGE&PARTNER Computer GmbH Schlossborner Weg 7 D-61479 Glashuetten

Phone +49 (0) 6174-966100 Fax +49 (0) 6174-966101