# A++Basics

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>A++Basics | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | | December 6, 2024 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

# Chapter 1

# A++Basics

## 1.1 A++ Library Basics

Welcome to the world of objects on the Amiga®!
---
by Armin Vogt

The goals of the A++ Library

The concepts of the A++ Library

Some points about the C++ implementation

For a grand tour through the A++ Library just use the 'Browse' buttons.

## 1.2 goals

So, you are really interested in the advantages of the A++ GUI Library.
Here is where we start a quick freshening journey through the yet unknown
but soon becoming well known space of programming a graphical user interface
under the terms of the object oriented paradigm.
First let me explain..

The goals of the A++ GUI Library
---

This Library should provide the programmer with an easy to use, reliable way
of creating graphical user interfaces (GUIs) for his programs.
(It sounds so easy, doesn't it?)

The intention was to..
a) give the programmer the advantage to rely on a set of tested solutions
   he can enhance according to his needs.
b) prevent the programmer from searching through hundreds of manual pages to
   find the right procedures for his purpose.
c) provide an programming interface that has a certain 'knowledge' about the

GUI's application programming interface (API).

The solution was to..
a) use an object oriented programming language, C++.
b) state a system of inheritances and analogies that gives the programmer a chance to find his conclusions through deduction, i.e. that feels logical.
c) provide all classes with a set of useful default parameters and make all classes react graciously on mistaken parameters.

## 1.3 concepts

The concepts behind the A++ GUI Library
---

The A++ Class Library represents all GUI elements as C++ objects. "Really!?", you might have exclaimed now. But let's see the major goals..

The graphical user interface programming classes are positioned in an inheritance tree, in contrast to a class forest approach.
The IntuiObject stands at the top of this tree. It..
- provides the uniform interface to the variety of different GUI elements.
- contains the methods to access GUI objects to gain information about and to manipulate them. ( setAttributes( ) / getAttribute( ) : derived classes only have to overwrite these methods to get the full service.)

The IntuiObject serves as powerful base class due to some features as there are:

a) Runtime dependencies between instances with tracking and releasing of all IntuiObject instances at the end of the program: each IntuiObject has its place in the runtime dependency tree of IntuiObjects in a way that it is attached to exactly one 'Owner' IntuiObject from the time of its creation to its cease from existence. One IntuiObject may have as many IntuiObjects as 'Childs' as you like. An IntuiObject will immediately be deleted when its 'Owner' is deleted. Or, to look at it from the other side, deleting an IntuiObject causes all its directly also as indirectly dependent 'Childs' to be deleted. This way is used to delete all IntuiObjects from the global 'IntuiRoot' at the end of a program.

b) Event handling: GadgetCV derived classes specify a callback method that interpretes the events triggered by the user transforming them into attribute changes via setAttributes( ). These may trigger notifications to other IntuiObjects.

c) Communications between IntuiObjects regarding attribute changes of an IntuiObject that wants to notify other IntuiObjects of new attribute values: Since the attributes that characterize the state of any GUI object are managed by the IntuiObject class it is appropriate to let the IntuiObject also handle automatic notifications between IntuiObjects with exactly the same methods the programmer uses to access objects directly ( setAttributes( ) / getAttribute( ) ). It does this not itself but uses another class, the ITransponder class which actually only supplies a virtual method being called for each attribute change with information about the change that can easily be customized by the

programmer.

## 1.4 implementation

Some fundamental implementation decisions

Being a programmer yourself, you may feel better about using the A++ Library with an increased knowledge about the guidelines and principles I laid down during the development process, from design to implementation to re-design to implementation and so on...

Up to now, I left behind about 12 months of intersected working time, in which some major design details have been reconsidered again and again. Some results of this process, I have written down here..

I. A++ makes extensive use of virtual methods where a class shall be extended or specialized by its subclasses.

II. Useful mechanisms are moved as far as it seems to be wise to the root of the class inheritance tree to give the benefit to all derived classes. (i.e. the IntuiObject Notification Mechanisms or the GraphicObject Border class)

III. I always had an eye on the code efficiency and memory usage and weighed both against certain designs. The code had to go conform with the C++ Reference Manual.
True performance were rather gained from a good design and consistent implementation then from "tricky" coding, which often leads to errors that are difficult to trace.

IV. The A++ Library is not merely a C++ wrapper library but recollects Amiga® system functions into objects with a consistent functionality and a well-defined interface.