HiGV

开发指南

文档版本　06

发布日期　2021-12-01

商标声明

 、HISILICON、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

上海海思技术有限公司

地址：　深圳市龙岗区坂田华为总部办公楼　邮编：518129

网址：　http://www.hisilicon.com/cn/

客户服务邮箱：　support@hisilicon.com

前 言

概述

本文档主要介绍 HiGV 图形组件的基本架构、各模块的功能特点和流程，场景设计及部分用法的参考。

本文档能够帮助客户了解 HiGV 图形组件的基本架构和各模块的功能，使客户能够对简单的问题进行处理。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3559A	V100
Hi3559C	V100
Hi3556A	V100
Hi3559	V200
Hi3556	V200
Hi3519A	V100
Hi3518E	V300
Hi3516D	V300
Hi3516C	V500
Hi3562	V100
Hi3566	V100
Hi3569	V100

产品名称	产品版本
Hi3568	V100

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2021-12-01	06	第六次正式版本发布 3.5.4、4.7.2、4.16.1 和 5.9.2 小节涉及修改
2021-04-30	05	第五次正式版本发布 3.5.2 和 4.17.1 小节涉及修改
2021-03-15	04	第四次正式版本发布 4.11.1 和 4.20.1 小节涉及修改
2020-12-12	03	第三次正式版本发布 4.11.1、4.12、4.13.1、4.22.1、5.1.4 小节涉及修改
2020-09-25	02	第二次正式版本发布 1.3.2、4.1.3、4.11.1 和 4.19.5 涉及修改 新增 4.22 小节

修订日期	版本	修订说明
2020-05-30	01	第一次正式版本发布 新增 4.21 小节 5.7.3 小节涉及更新。
2020-04-30	00B07	第七次临时版本发布 新增 4.20 小节
2020-02-28	00B06	第六次临时版本发布 新增 4.19 小节
2020-01-20	00B05	第五次临时版本发布 3.7.4 小节，表 3-3 涉及修改 4.16 小节涉及修改 新增 4.17 和 4.18 小节
2019-10-23	00B04	第四次临时版本发布 4.1.2、5.2 和 5.10.4 小节涉及修改 删除 4.1.4、4.9、4.16、5.2.2 和 5.2.3 小节 新增 4.16 小节
2019-07-30	00B03	第三次临时版本发布 新增 3.7.4 小节
2018-07-20	00B02	第二次临时版本发布 4.6 小节涉及修改，新增 4.6.1 小节
2018-05-24	00B01	第一次临时版本发布 3.5.2 节，增加“动画信息” 2.2 和 3.5.3 小节涉及修改 新增 5.9.2 和 5.9.3 小节

目 录

前 言 ii

1 概 述 1

1.1 HiGV 架构 1

1.2 重要概念 2

1.3 应用开发综述 4

1.3.1 组成部分 4

1.3.2 应用程序示例 4

2 环境配置 10

2.1 开发环境 10

2.2 运行环境 11

2.3 运行和调试 11

3 模块详解 12

3.1 控件 12

3.1.1 控件的基本属性 12

3.1.2 控件的使用 13

3.1.3 控件继承关系 14

3.2 消息 15

3.2.1 消息的产生 15

3.2.2 消息特点 16

3.2.3 消息传递流程 17

3.2.4 消息回调事件 17

3.2.5 典型消息例举 19

3.2.6 内部消息处理列表 21

3.2.7 Xml 定义的消息回调事件.....22
3.3 绘制.....23
3.3.2 控件绘制.....24
3.3.3 窗口绘制流程.....25
3.3.4 隐藏控件.....27
3.4 定时器.....27
3.4.1 定时器的使用.....27
3.4.2 定时器与 HiGV 主线程关系.....28
3.5 资源.....29
3.5.1 资源的创建.....29
3.5.2 资源的使用.....29
3.5.3 Xml 设置资源.....34
3.5.4 HiGV 资源加载与释放.....35
3.6 数据模型.....37
3.6.1 数据形态.....37
3.6.2 ADM 使用.....38
3.6.3 数据模型 xml.....41
3.7 多语言.....47
3.7.1 多语言 xml 描述.....47
3.7.2 注册与切换语言环境.....49
3.7.3 语言环境书写方向切换.....49
3.7.4 多国语言支持.....50
3.8 Xml 文件描述.....51
3.8.1 Xml 界面描述.....52
3.8.2 Xml 解析.....54
4 控件详细介绍.....55
4.1 Window 控件.....55
4.1.1 窗口重叠.....55
4.1.2 窗口与 surface.....56
4.1.3 窗口私有属性.....56
4.2 GroupBox 控件.....57

4.3 Button 控件 57
4.3.1 Button 类型 58
4.3.2 Button 私有属性 59
4.3.3 Button 的独特事件 59
4.4 Label 控件 59
4.5 Image 控件 60
4.5.1 存储在内存的图片 60
4.6 ImageEx 控件 62
4.6.1 ImageEx 的独特事件 62
4.7 ListBox 控件 62
4.7.1 添加数据 63
4.7.2 ListBox 私有属性 63
4.7.3 ListBox 独特事件 65
4.7.4 单元格焦点 65
4.7.5 单元格小图标 65
4.7.6 列回调函数 67
4.8 ScrollBar 控件 68
4.9 ProgressBar 控件 69
4.10 Clock 控件 69
4.10.1 Clock 定内部时器 69
4.10.2 Clock 文本显示 70
4.10.3 时间设置与修改 82
4.10.4 Clock 私有属性 82
4.10.5 Clock 的独特事件 82
4.11 ScrollGrid 控件 83
4.11.1 ScrollGrid 私有属性 83
4.11.2 ScrollGrid 的独特事件 84
4.12 TrackBar 控件 84
4.12.1 TrackBar 私有属性 85
4.12.2 TrackBar 的独特事件 85
4.13 ScrollView 控件 85

4.13.1 ScrollView 私有属性 86
4.13.2 ScrollView 的独特事件 87
4.14 SlideUnlock 控件 87
4.14.1 SlideUnlock 私有属性 87
4.14.2 SlideUnlock 的独特事件 87
4.15 WheelView 控件 88
4.15.1 WheelView 私有属性 88
4.15.2 WheelView 的独特事件 88
4.16 ScaleView 控件 88
4.16.1 ScaleView 私有属性 89
4.16.2 ScaleView 的独特事件 90
4.17 Edit 控件 90
4.17.1 Edit 特殊风格 90
4.17.2 字符编码 90
4.18 ScrollText 控件 90
4.19 MsgBox 控件 91
4.19.2 MsgBox 私有属性 92
4.19.3 MsgBox 的独特事件 94
4.19.4 设置 MsgBox 按钮 94
4.19.5 阻塞式显示 MsgBox 94
4.20 Charts 控件 94
4.20.1 Charts 私有属性 94
4.21 MultiEdit 控件 96
4.21.1 MultiEdit 边界 96
4.21.2 字符编码 96
4.22 Cursor 鼠标控件 96
4.22.1 Cursor 种类 96
4.22.2 鼠标相关属性 97
4.22.3 参考代码 97
5 HiGV 编程技术 99
5.1 自定义绘制 99

5.1.1 创建绘制环境 99
5.1.2 自定义绘制接口 99
5.1.3 自定义绘制流程 100
5.1.4 参考代码 101
5.2 输入设备适配 103
5.2.1 适配输入事件 103
5.2.2 触摸屏适配参考代码 103
5.3 多图层 114
5.3.1 开发应用 114
5.3.2 图层创建参数解析 114
5.3.3 参考代码 115
5.4 MXML 118
5.4.1 开发应用 118
5.4.2 参考代码 118
5.5 CLUT8 图层格式 120
5.5.1 开发应用 120
5.5.2 注意事项 120
5.5.3 参考代码 121
5.6 RLE 格式使用 123
5.6.1 开发应用 124
5.6.2 参考代码 124
5.7 输入法 127
5.7.1 开发应用 128
5.7.2 参考代码 129
5.8 线程安全 135
5.9 动画 135
5.9.1 创建动画方法 1 135
5.9.2 创建动画方法 2 136
5.9.3 列表控件回弹效果 137
5.10 性能优化指导 137
5.10.1 控件/资源实例 137

5.10.2 图片型皮肤优化 137
5.10.3 数据模型合理使用 137
5.10.4 隐藏释放风格和窗口切换性能 138

插图目录

图 1-1 HiGV 架构图 2
图 1-2 单击“OK”按钮前的界面 9
图 1-3 单击“OK”按钮后的界面 9
图 3-1 控件继承关系图 14
图 3-2 消息回调流程图 18
图 3-3 HiGV 界面图示 24
图 3-4 OK 按钮绘制流程图 25
图 3-5 窗口显示时序图 26
图 3-6 九宫格皮肤示意图 32
图 3-7 列表框示例 37
图 3-8 控件与数据模型绑定关系图 39
图 4-1 界面示例 56
图 4-2 GroupBox 图示 57
图 4-3 Normal 类型 Button 图示 57
图 4-4 Switch 类型 Button 图示 58
图 4-5 Toggle 类型 Button 图示 58
图 4-6 Label 图示 60
图 4-7 Image 图示 60
图 4-8 内存图片说明图示 61
图 4-9 单元格 ListBox 图示 63

图 4-10 ScrollBar 与绑定控件图示 68
图 4-11 ProgressBar 图示 69
图 4-12 时钟图示 69
图 4-13 ScrollGrid 图示 83
图 4-14 TrackBar 图示 84
图 4-15 ScrollView 图示 86
图 4-16 SlideUnlock 图示 87
图 4-17 WheelView 图示 88
图 4-18 ScaleView 图示 89
图 4-19 MsgBox 图示 92
图 5-1 拼音输入法图示 127
图 5-2 数字软键盘图示 128

表格目录

表 3-1 内部消息处理列表 21
表 3-2 xml 消息回调列表 22
表 3-3 多国语言支持列表 50
表 4-1 Clock 显示格式定义 70

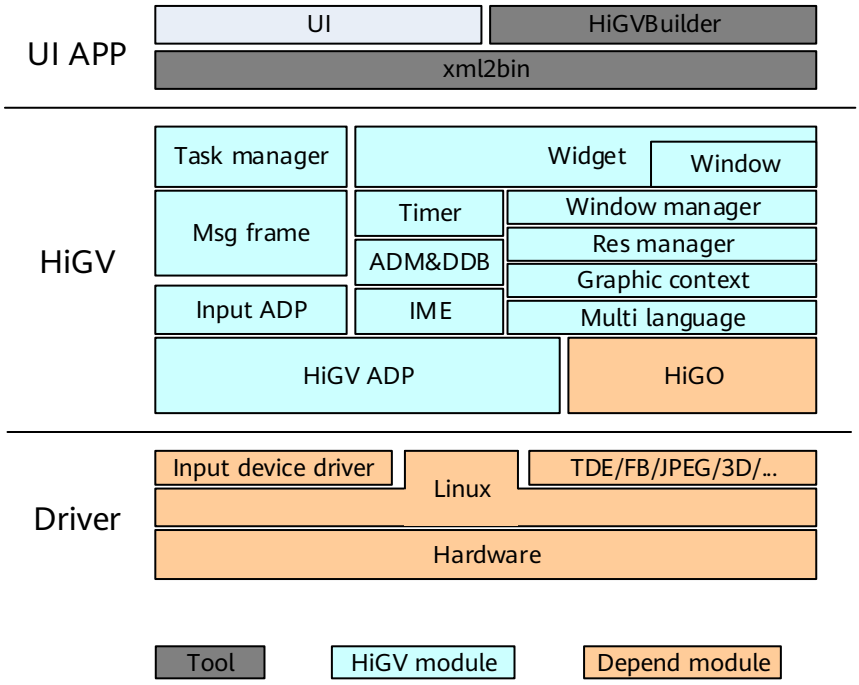
1 概 述

1.1 HiGV 架构

HiGV 是一个轻量级的 GUI 系统，主要是为芯片平台提供统一的轻量级、高效、易用的 GUI 解决方案。该系统采用分层机制实现，其中底层图形库依赖 SDK 中 HiGO 库，而 HiGO 建立在基本的图形驱动(FrameBuffer、芯片 2D 加速驱动、图片编解码等)之上，如图 1-1 所示。

HiGV 吸取了优秀图形系统设计思想，采用大量成熟的设计模式，使用了 XML 标记语言进行界面描述，具有统一高效的系统资源管理功能，并且 HiGV 还提供了丰富的控件。另外，为了解决易用性问题，还提供了快速可视化界面开发工具 HiGVBuilder 来帮助用户进行界面开发。

图1-1 HiGV 架构图



架构中主要模块的功能如下：

- HiGO：基本图形组件，提供画线、填充、搬移图片等基本绘制接口。
- TDE：芯片 2D 加速驱动接口。
- FB：图形层驱动接口。
- HiGV 主要模块：详见 3 “模块详解”。
- Xml2bin：xml 文件解析工具。
- HiGVBuilder：可视化的界面制作辅助工具。

1.2 重要概念

在进行 HiGV 系统开发之前，须要了解图形系统的一些基本概念和重要术语：

- 图层（layer）：独立的图形缓冲区，多个图层可以使用 apha 值进行合成输出，对应硬件图层。
- 表面（surface）：用来保留像素数据的内存区域，可以将其理解为画布，通常图形在 surface 绘制完成，再将 surface 搬移至图层进行表现。

- 视图（view）：xml 界面描述文件中的 view 虚拟作为界面描述 xml 文件的识别 ID，不具有对界面数据的实际意义。
- 控件（widget）：界面的基本组成元素，图形的具体表现载体，我们所看到的界面即由若干个控件组成，例如按钮、文本框、图片框等。控件有自己的矩形区域，这决定了它在界面中的显示位置及大小。控件可以分为容器类控件和非容器类控件，非容器类控件只能布置在容器类控件中。
- 窗口（window）：界面基础元素，容器类控件。窗口内可以放置任何非窗口控件，且窗口本身不能放置在其他容器类控件内。窗口与窗口之间是同级兄弟关系可以互相叠加，窗口内不能再包括窗口，窗口必须附属在图层上。
- 父子关系（parent and children）：当控件 A(非窗口)放置在容器控件 B 内时，我们将 B 称为 A 的父，A 称为 B 的子。有着相同父容器的控件，建议不要重叠并且同时显示。
- Z 序（window level）：即窗口层级，决定窗口之间叠加时的上下层顺序。
- 皮肤（skin）：控件的基本外观描述，控件的每种状态对应一个皮肤，控件的基本状态有以下四种，
  - 普通状态，控件的基本状态；
  - 激活状态，控件获得了焦点后的状态；
  - 高亮状态，没有获得焦点但希望区别于普通状态的状态；
  - 禁用状态，控件被禁止激活的状态。
- 抽象数据模块（ADM）：Abstract Data Model，对控件展示的数据操作进行抽象，为用户提供统一的数据操作接口。
- 默认数据缓冲（DDB）：Default Data Base，为数据模型提供简单的数据缓冲和管理。
- 资源（resource）：字库、图片、多语言字串。
- 句柄（handle）：控件、ADM、多语言字串、字体等实例的身份标识。
- 消息（msg）：由 GUI 系统或用户触发的能引起系统行为改变的事件。当消息产生，GUI 系统或应用程序会根据消息类型做出响应，并回调用户注册的消息事件。
- 消息回调事件（msg call back）：用户通过注册消息回调事件将函数与控件绑定，当消息传达到控件时，回调该控件所绑定的消息回调函数，通常将具体的业务事件作为消息回调，详见 3.2.4 “消息回调事件”。

1.3 应用开发综述

1.3.1 组成部分

通常，一个 HiGV 工程除了 main 函数外，还有其他几个重要的组成部分：

- xml 界面描述文件
- 业务处理文件
- 界面使用到的资源文件。

UI 由若干个 HiGV 控件组成，这些控件可以使用 xml 文件描述，也可以通过调用 HiGV 的接口创建。

通常使用 xml 文件描述界面及界面使用的资源数据，xml 描述界面方便易用，可以省略大量的控件创建代码，HiGV 还会自动生成控件句柄，详见 Xml 文件描述。

也可以直接调用 HiGV 的接口创建界面控件，接口创建控件需要大量的变量放置这些控件的句柄以便于对这些控件进行操作，调用接口方式多用于动态创建或销毁控件。

1.3.2 应用程序示例

如下以简单的 Hello World 程序为例，通过该例子来说明 XML 文件、描述界面事件文件、界面元素属性及事件的关联关系。

Hello World 程序要求为：

- 一个窗口上有一个“确定”按钮和一个文本框，文本框默认内容为空。
- 当按确定按钮时，把文本框内容设置为 Hello World!并显示。

XML 描述文件为：

```
<view
id = “hello_sample”                    <!—view name-->
onload = “”
unload = “”>
    <window
        id = “hello_sample”                <!—window name-->
        top = “0”                    <!—widget pos-->
        left = “0”
        width = “720”
        height = “576”
```



```
        normalskin = “commonpic_skin_colorkey” <!—normal skin-->
        transparent = “no”
        isrelease = “yes”
        opacity = “255”
        winlevel = “0”
        onshow = “hello_window_onshow”>   <!—onshow call back-->
        <button
            id = “hello_button_ok”              <!—ok button ID-->
            top = “285”
            left = “235”
            width = “246”
            height = “40”
            normalskin = “commonpic_normalskin_button”
            disableskin = “”
            highlightskin = “”
            activeskin = “commonpic_activeskin_button”      <!—active state skin-->
            transparent = “no”
            isrelease = “no”
            text = “ID_STR_OK”          <!—multi language ID-->
            alignment = “hcenter|vcenter”
            onclick = “hello_button_onclick”/>   <!—onclick call back-->

        <label
            id = “hello_label_helpinfo”     <!—label ID-->
            top = “200”
            left = “235”
            width = “246”
            height = “50”
            normalskin = “common_skin_black”
            disableskin = “”
            highlightskin = “”
            activeskin = “”
            transparent = “yes”
            isrelease = “no”
            text = ""
            alignment = "hcenter|vcenter"/>
    </window>
</view>
```


UI_XXX.c ：

```
/**Window onshow call back*/
HI_S32 hello_window_onshow (HI_HANDLE hWidget,HI_U32 wParam, HI_U32lParam)
{
    return HIGV_PROC_GOON;
}
/**OK button onclick call back*/
HI_S32 hello_button_onclick (HI_HANDLE hWidget,HI_U32 wParam, HI_U32lParam)
{
    HI_S32 s32Ret;

    /**Change the label content*/
    s32Ret = HI_GV_Widget_SetText(hello_label_helpinfo,  “Hello World!”);
    return HIGV_PROC_GOON;
}
```

Main.c：

```
#define SCREEN_WIDTH   1280
#define SCREEN_HEIGHT   720
#define PROJECT_ID       hello_sample //from xml

HI_S32 main(HI_S32 argc, HI_CHAR *argv[])
{
    HI_S32 Ret;
    HI_S32 g_hApp ;

    /** The layer info, layer width is 1280 and height is 720. The pixel format is HIGO_PF_8888, Each pixel occupies 32 bits, and the A/R/G/B components each occupies 8 bits. Use Dual buffers supported .*/
    HIGO_LAYER_INFO_S LayerInfo = {SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_WIDTH, SCREEN_HEIGHT,

(HIGO_LAYER_FLUSHTYPE_E)(HIGO_LAYER_BUFFER_DOUBLE),
                                        HIGO_LAYER_DEFLICKER_AUTO,
                                        HIGO_PF_8888, HIGO_LAYER_HD_0};
```



```
/**higv init*/
Ret = HI_GV_Init();
if (HI_SUCCESS != Ret)
{
      return Ret;
}

/**Init parser module to parser binary file from xml.*/
Ret = HI_GV_PARSER_Init();
if (HI_SUCCESS != Ret)
{
    printf("HI_GV_PARSER_Init failed! Return: %d\n",Ret);
    return Ret;
}

/**Load higv.bin*/
Ret = HI_GV_PARSER_LoadFile("./higv.bin");
if (HI_SUCCESS != Ret)
{
      HI_GV_Deinit();
      HI_GV_PARSER_Deinit();
      return Ret;
}
/**Create layer for draw graphics*/
HI_HANDLE LAYER_0 = INVALID_HANDLE;
Ret = HI_GV_Layer_Create(&LayerInfo, &LAYER_0);
if (HI_SUCCESS != Ret)
{
      printf("HI_GV_Layer_CreateEx failed! Return: %x    \n",Ret);
      HI_GV_Deinit();
      HI_GV_PARSER_Deinit();
      return Ret;
}
else
      printf("LAYER_0 create ok ...\n");
```



```
/**Load view，PROJECT_ID is view ID from xml file. Create all child widget of PROJECT_ID.*/
Ret = HI_GV_PARSER_LoadViewById(PROJECT_ID);
if (HI_SUCCESS != Ret)
{
    printf("HI_GV_PARSER_LoadViewById failed! Return: %x\n",Ret);
    HI_GV_Deinit();
    HI_GV_PARSER_Deinit();
    return Ret;
}

/**Create HiGV app.*/
Ret = HI_GV_App_Create("MainApp", (HI_HANDLE*)&g_hApp);
if (HI_SUCCESS != Ret)
{
    printf("HI_GV_App_Create failed! Return: %d\n",Ret);
    HI_GV_Deinit();
    HI_GV_PARSER_Deinit();
    return Ret;
}

/**Show window.*/
Ret = HI_GV_Widget_Show(PROJECT_ID);
if(Ret !=0)
    printf("HI_GV_Widget_Show() failed ....: %d \n",Ret);

Ret = HI_GV_Widget_Active(PROJECT_ID);
if(Ret !=0)
    printf("HI_GV_Widget_Active() failed ....: %d \n",Ret);

/*Start HiGV app*/
HI_GV_App_Start(g_hApp);
/** If the HiGV app over, the HI_GV_App_Start will be return.*/
HI_GV_App_Destroy(g_hApp);
HI_GV_PARSER_Deinit();
HI_GV_App_Stop(g_hApp);

return 0;
```



```
}
```

程序运行后，当按下“OK”键，产生 onclick 事件，导致调用 hello_button_onclick 函数，在该函数中修改文本框内容为“Hello World!”。如图 1-2、图 1-3 所示。

图1-2 单击“OK”按钮前的界面

图1-3 单击“OK”按钮后的界面

2 环境配置

2.1 开发环境

要基于上海海思芯片平台上进行开发，最简单的方法是采用上海海思数字媒体 SDK 开发包：

- 开发所需要的头文件全部位于 SDK 的 pub/include 目录下；
- 需要用到的库文件全部位于 SDK 的 pub/lib 目录下；
- 驱动文件(ko 形式)全部位于 SDK 的 rootbox 的 kmod 目录下。

上海海思数字媒体 SDK 开发包只是提供最基本的开发环境，并不包括 HiGV，而要在 HiGV 平台上开发界面相关的应用软件，则需要 HiGV 的开发库和工具 xml2bin、HiGVBuilder。

- HiGVBuilder 是基于 Eclipse 平台基础上的快速可视化界面开发工具，可以在 PC 机上，只通过鼠标的拖、拉、点、放就可以生成界面，在工具中设置属性和资源路径等，并且最终可自动生成界面布局和资源 XML 描述文件。
- xml2bin 是 HiGV 提供的可以将界面和资源 XML 描述文件转换成 HiGV 可识别的二进制文件（higv.bin）和界面事件描述的 C 代码文件（higv_cextfile.c）的工具。

HiGV 库 libhigv 主要提供以下功能：

- 提供 HiGV 的核心功能和基本的图形框架；
- 提供 HiGV 基本的控件；
- 解析 xml2bin 工具转换得到的二进制文件；
- 提供平台相关的适配功能。

2.2 运行环境

在运行应用程序前，请如下设置运行环境：

- 在系统运行前，烧写 fastboot、hi_kernel（Linux 内核）、rootbox 映像文件。详细请参见《Hi35xx Vx00 开发环境用户指南》和《HiBurn 工具使用指南》。
- LCD 屏与开发板连接或者通过视频信号线连接电视机与开发板。
- 点亮 LCD 屏或者打开电视机，选择正确的输入源。

2.3 运行和调试

- 加载各个 HiGV 依赖的内核驱动模块。在运行程序之前，需要正确加载图形系统依赖内核驱动模块，如 TDE、FB 等。
- 运行程序。进入“xxx_sample”文件夹，执行可执行程序“./ xxx_sample”，即可观察到在显示终端上输出的控件。
- 调试验证。通过调试串口监视运行情况，如果存在异常，根据调试串口输出信息分析具体原因，通过系统返回的错误码分析错误产生的原因。

须知

在调试 HiGV 前最好先运行 SDK 中 HiGO 的 Sample，确保 HiGO 作为 HiGV 的绘制基础能够正常运行。

3 模块详解

3.1 控件

控件是最为重要的界面元素。原因如下：

- HiGV 的图形、文字需要依赖控件得以表现
- 消息事件的产生及执行以控件为基础
- 界面行为改变也是控件数据的改变

3.1.1 控件的基本属性

控件的基本属性决定了控件的功能、外观、位置等内容，控件必须设置的基本通用属性如下：

- Type：控件的类型。决定了控件的基本功能，xml 文件以控件标签表示。
- Rect(rectangle)：控件的矩形。决定控件的宽、高以及控件相对父容器左上角的坐标偏移位置。
- Parent：父容器。为控件指定父容器，窗口没有父容器，窗口的坐标偏移相对图层。
- Handle：句柄。控件、皮肤、字体、多语言字串、数据模型的身份标识，HiGV 通过识别 handle 找到对应的实例。
- skin：控件皮肤。控件的基本外观表现，至少须设置普通皮肤，否则控件可能因没有皮肤而无法显示。

除了以上必须设置的基本属性外，控件中还有兄弟控件用来设置字体、文本、对齐方式、各消息回调事件、特殊风格等属性，不同的控件类型还有各自的私有属性。头文件《hi_gv_widget.h》包含了控件的公共接口，消息定义及控件其他公共属性。

3.1.2 控件的使用

使用控件前需要先创建一个控件。应用程序示例展示了使用 xml 创建控件的方法。此外，也可以调用 HiGV 的创建接口生成控件，如下是一段简单的窗口创建代码：

```
HIGV_WCREATE_S infoWindow;
HIGV_WINCREATE_S WinCreate;
HI_HANDLE hWindow;

memset(&infoWindow, 0x00, sizeof(infoWindow));
infoWindow.rect.x = 20;//窗口相对图层x坐标
infoWindow.rect.y = 20;//窗口相对图层y坐标
infoWindow.rect.w = 400;//窗口宽度
infoWindow.rect.h = 400;//窗口高度
WinCreate.hLayer = HIGO_LAYER_HD_0;//窗口所属图层
WinCreate.PixelFormat = HIGO_PF_BUTT;//不设置像素格式，缺省使用图层的像素格式
infoWindow.pPrivate = &WinCreate;
infoWindow.hParent = INVALID_HANDLE;//窗口没有父容器
infoWindow.style = 0;//不设置特殊风格
infoWindow.type = HIGV_WIDGET_WINDOW;//指定为窗口控件
/**调用接口创建控件，得到窗口的句柄hWindow*/
if(HI_SUCCESS == HI_GV_Widget_Create(&infoWindow,  &hWindow))
{
    /**为hWindow设置一个普通皮肤window_skin，这里省略了皮肤的创建过程*/
    HI_GV_Widget_SetSkin(hWindow, HIGV_SKIN_NORMAL, window_skin);
}
```

控件创建成功后就可以使用了，以 1.3.2 应用程序示例作为例：

- 调用 HI_GV_Widget_Show 接口显示一个控件；
- 调用 HI_GV_Widget_Hide 接口隐藏一个控件；
- 通过按下按钮后触发 HI_GV_Widget_SetText 为控件设置文本后调用 HI_GV_Widget_Paint 重绘控件让文本显示。
- 通过调用不同的 HiGV 接口来改变控件的行为和外观达到改变界面图形的目的。
- 调用 HI_GV_Widget_Destroy 销毁控件。

3.1.3 控件继承关系

控件可分为：

- 普通控件

  普通控件是在控件继承树中处于叶子节点的控件。

- 容器控件

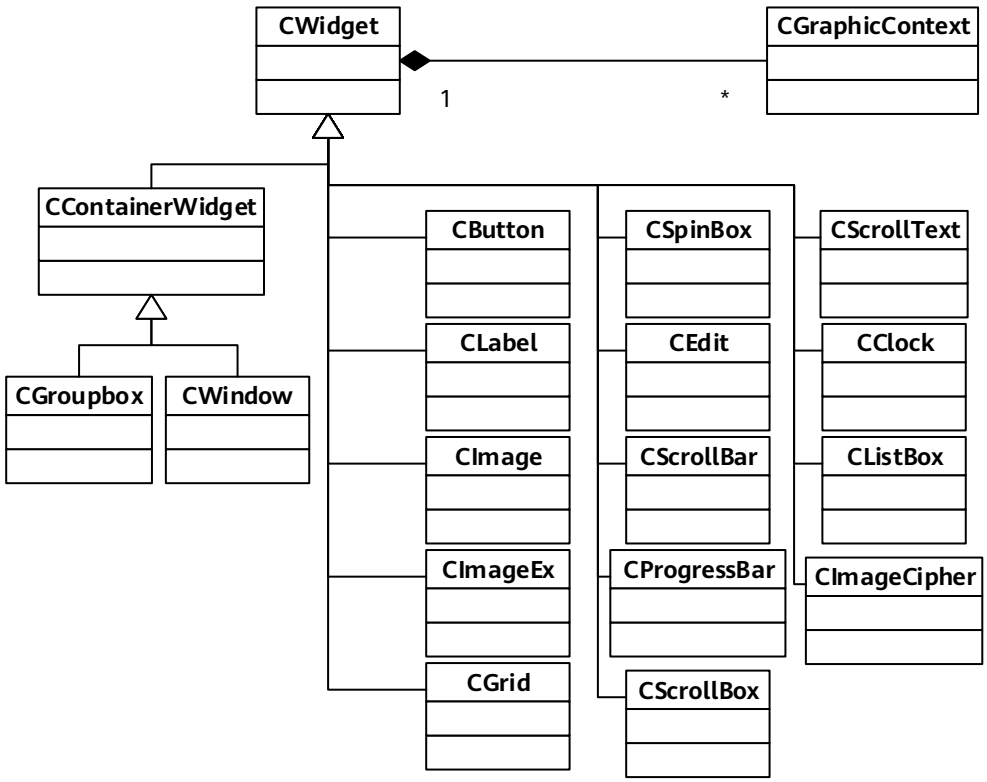
  容器控件中可以放置若干个普通控件。

- 组合控件

  组合控件则是由多个普通控件或容器控件组成的控件，如 Combobox 控件是由 Window、Spin、Listbox 等控件组合而成。

HiGV 控件内部采用 C++面向对象设计方法实现，控件间存在的继承关系如图 3-1 所示。

图3-1 控件继承关系图



3.2 消息

消息是由 GUI 系统或用户触发的能引起系统行为改变的事件。当消息产生，GUI 系统或应用程序会根据消息类型做出响应，并回调用户注册的消息事件。

HiGV 的主线程负责消息的捕获和处理，从调用接口 HI_GV_App_Start 开始至调用接口 HI_GV_App_Stop 停止。HiGV 获取到消息后，将消息传至焦点控件进行处理，完成后继续捕获下一条消息。

HiGV 消息定义在头文件《hi_gv_widget.h》中，有统一的前缀“HIGV_MSG_”。

3.2.1 消息的产生

消息可以是 HiGV 内部数据改变触发产生，也可以是用户触发产生。消主要来源有：

- 按键、鼠标、触摸屏等输入设备产生的消息事件。由 HiGV 适配或用户适配接收输入设备事件，再将其转为消息发送至消息队列，通用输入设备事件如下：

1. 遥控器，面板消息类型
   - 遥控器、前面板等按键设备按下产生 HIGV_MSG_KEYDOWN。
   - 遥控器、前面板等按键设备抬起产生 HIGV_MSG_KEYUP。
   - 鼠标左、右键按下产生 HIGV_MSG_MOUSEDOWN。
   - 鼠标左、右键抬起产生 HIGV_MSG_MOUSEUP。
2. 鼠标消息类型
   - 鼠标中键滚动产生 HIGV_MSG_MOUSEWHEEL。
   - 鼠标左键长按产生 HIGV_MSG_MOUSELONGDOWN。
   - 鼠标双击产生 HIGV_MSG_MOUSEDBCLICK。
   - 鼠标移入某控件范围产生 HIGV_MSG_MOUSEIN。
   - 鼠标移出某控件范围产生 HIGV_MSG_MOUSEOUT。
3. 触摸屏消息类型
   - 触摸屏上某控件范围进行触摸操作产生 HIGV_MSG_TOUCH
   - 触摸屏上某控件范围进行轻触并抬起操作产生 HIGV_MSG_GESTURE_TAP
   - 触摸屏上某控件范围进行长按操作产生 HIGV_MSG_GESTURE_LONGTAP
   - 触摸屏上某控件范围进行轻扫操作产生 HIGV_MSG_GESTURE_FLING
   - 触摸屏上某控件范围进行滑动操作产生 HIGV_MSG_GESTURE_SCROLL
   - 触摸屏上某控件范围进行捏合操作产生 HIGV_MSG_GESTURE_PINCH

触摸事件处理，主要是获取响应触摸事件的目标控件进行判断，基本处理和鼠标的 hittest 相似，但是需要在控件树进行截取触摸事件，这样父控件可优先进行处理触摸事件。手势事件是 Touch 事件的高级包装，其判断的依据是 Touch 事件流，主要识别的事件有：

- HIGV_GESTURE_TAP：手指轻触触摸屏并抬起。
- HIGV_GESTURE_LONGTAP：长按手势，超过两秒触发。
- HIGV_GESTURE_FLING：轻扫手势，可以任何方向进行操作。
- HIGV_GESTURE_SCROLL：滑动手势，手指在触摸屏上滑动。
- HIGV_GESTURE_PINCH：捏合手势，两个手指在触摸屏上做捏合动作。

- 调用 HI_GV_Msg_SendAsync 等消息发送接口直接对控件发送消息。

说明
所有消息发送接口见头文件《hi_gv_msg.h》。

- 调用控件接口触发 HiGV 事件，如 HI_GV_Widget_Show 产生 HIGV_MSG_SHOW 和 HIGV_MSG_PAINT，HI_GV_List_SetSelItem 产生 HIGV_MSG_ITEM_SELECT。
- 定时器产生的 HIGV_MSG_TIMER。
- 控件内部数据改变产生的事件，如控件获取焦点触发 HIGV_MSG_GET_FOCUS 失去焦点触发 HIGV_MSG_LOST_FOCUS。

3.2.2 消息特点

消息的特点有：

- 先触发先执行，序列稳定；
- 短时间向内同一目标发送多次重复消息时将合并，避免多余操作；

  会合并的消息有：

  - HIGV_MSG_PAINT 绘制消息。
  - HIGV_MSG_REFRESH_WINDOW 刷新窗口消息。
  - HIGV_MSG_FORCE_REFRESH_WINDOW 强制刷新窗口消息。
  - HIGV_MSG_TIMER 定时器消息。
  - HIGV_MSG_DATA_CHANGE 数据改变消息。
  - HIGV_MSG_ST_UPDATE 滚动字幕更新消息。
  - HIGV_MSG_MOUSEMOVE 鼠标移动消息。
  - HIGV_MSG_MOUSEWHEEL 鼠标滚轮消息。

- 异步发送消息触发 GUI 业务，不阻塞主线程；
- 发送消息时伴随带入参数，使 HiGV 和回调函数得到准确的消息事件数据。

说明
不同的消息参数意义不一样，具体请参考头文件《hi_gv_widget.h》。

3.2.3 消息传递流程

消息产生时，通常直接将消息分发至目标控件处理。

但输入设备触发的输入事件（鼠标、按键）在产生时不会指定目标控件，HiGV 对这类消息的传递流程如下。

- 按键消息事件传递流程：

步骤 1 将按键消息分发给当前焦点窗口

步骤 2 如果焦点窗口有子控件，窗口将按键传递给子控件，若子控件为容器类控件则继续逐层传递直至末端焦点子控件。

步骤 3 末端控件处理按键消息。

步骤 4 如果用户注册的按键消息回调函数返回值为 HIGV_PROC_GOON，消息会逐层向当前处理消息控件的父容器传递，直至消息到达窗口或消息回调函数返回值为 HIGV_PROC_STOP。

----结束

- 鼠标消息事件传递流程：

步骤 1 计算当前鼠标的坐标位置，将消息发送至坐标当前最顶层的窗口。

步骤 2 由窗口将消息传递至鼠标坐标位置的末端控件。

步骤 3 末端控件处理鼠标消息。

步骤 4 如果用户注册的鼠标消息回调函数返回值为 HIGV_PROC_GOON，消息会逐层向当前处理消息控件的父容器传递，直至消息到达窗口或消息回调函数返回值为 HIGV_PROC_STOP。

----结束

3.2.4 消息回调事件

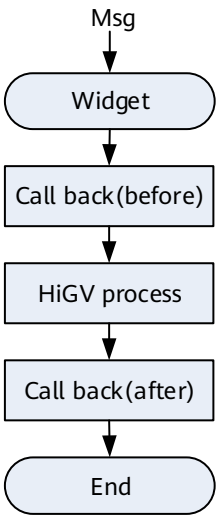
须知

因为消息回调事件都是在 HiGV 线程中执行的，所以消息回调事件的堵塞会阻塞整个 HiGV 线程运行。

如果希望在 HiGV 控件收到消息时，处理一些业务事件，可以使用消息回调功能。消息回调流程如图 3-2 所示。

回调事件是控件和业务交互的主要途径，通常把业务功能放在回调函数中，实现人机交互、用户操作等功能。

图3-2 消息回调流程图



接口 HI_GV_Widget_SetMsgProc 向控件注册消息回调事件，该接口有四个参数：

- hWidget：注册的控件句柄。
- Msg：注册的消息 ID，触发回调事件的消息。
- CustomProc：收到 Msg 的回调事件。
- ProcOrder：事件处理优先级，如果 HiGV 对该消息有内部处理，则：
  - HIGV_PROCORDER_BEFORE 表示 CustomProc 在控件内部处理之前回调；
  - HIGV_PROCORDER_AFTER 表示 CustomProc 在控件内部处理之后回调。

回调函数的通用格式如下：

```
typedef HI_S32 (*HIGV_MSG_PROC)(HI_HANDLE hWidget,
HI_U32 wParam, HI_U32 lParam)
```


其中：

- hWidget：回调函数所绑定的控件句柄。
- wParam：第一个事件参数。部分事件 HiGV 内部处理会对其进行处理，可能会使用到该参数，具体请参考《hi_gv_widget.h》。
- lParam：第二个事件参数，通常作为私有数据代入参数。
- 返回值：因为回调函数是穿插在控件数据改变过程中的，其返回值可以影响到控件对事件处理。返回值一般使用 HIGV_PROC_GOON 表示该条消息继续传递，影响到按键和鼠标事件是否传递给父容器，返回值 HIGV_PROC_STOP 可以结束事件处理。

调用发送消息接口触发消息事件，发送消息事件可以代入参数。

- HI_GV_Msg_SendAsync(HI_HANDLE hWidget, HI_U32 MsgId, HI_U32 Param1, HI_U32 Param2)
  - 参数 Param1 对应回调事件 wParam，它会影响到控件内部处理；
  - 参数 Param2 对应回调事件 lParam，如果控件内部有处理可能会被覆盖为 0。
- HI_GV_Msg_SendAsyncWithData(HI_HANDLE hWidget, HI_U32 MsgId, HI_VOID *pBuf, HI_U32 BufLen)

  该接口可以代入私有数据，pBuf 对应回调函数的 lParam。

3.2.5 典型消息例举

HIGV_MSG_SHOW

控件显示消息。控件在显示之前的准备或其他业务可以注册该消息事件。

- 控件接口触发：HI_GV_Widget_Show 接口可以使隐藏状态的控件显示出来，它同步回调 HIGV_MSG_SHOW 事件，再异步绘制控件，代入参数 wParam 和 lParam 都为 0。
- MSG 接口发送消息：不能显示控件，只能回调事件，代入参数 wParam 为 Param1，lParam 为 Param2 或 pBuf。
- 返回值影响：无。

HIGV_MSG_PAINT

控件绘制消息。控件绘制前、后的业务处理可以注册该消息事件。

- 触发条件：所有会引起控件绘制的操作和事件。

- 参数意义：控件绘制的区域，可以用 HIGV_U32PARAM_TORECT 将参数转换为 HI_RECT。
- MSG 接口发送消息：不能绘制控件，只能回调事件，代入参数 wParam 为 Param1，lParam 为 Param2 或 pBuf。
  - 返回值影响：处理优先级为 HIGV_PROCORDER_BEFORE 的回调事件返回值如果不为 HIGV_PROC_GOON 会**中断控件绘制。**

HIGV_MSG_KEYDOWN/HIGV_MSG_MOUSEDOWN

控件按下业务事件。

- 输入设备触发：wParam 为按键值或鼠标键值，lParam 为 0，同时还能触发控件对输入事件的响应。
- MSG 接口发送消息：与输入设备触发行为一致。
- 返回值影响：处理优先级为 HIGV_PROCORDER_BEFORE 的回调事件。
  - 如果返回值不为 HIGV_PROC_GOON 会中断控件对该消息的响应；
  - 如果返回值为 HIGV_PROC_GOON **消息会继续传递给控件父容器。**

HIGV_MSG_TOUCH

控件按下业务事件。

- 输入设备触发：wParam 为 TOUCH 事件结构体长度，lParam 为 TOUCH 事件结构体指针，同时还能触发控件对输入事件的响应。
- MSG 接口发送消息：与输入设备触发行为一致。
- 返回值影响：处理优先级为 HIGV_PROCORDER_BEFORE 的回调事件。
  - 如果返回值不为 HIGV_PROC_GOON 会中断控件对该消息的响应；
  - 如果返回值为 HIGV_PROC_GOON **消息会继续传递给控件父容器。**

HIGV_MSG_GESTURE_TAP/ HIGV_MSG_GESTURE_LONGTAP/HIGV_MSG_GESTURE_FLING/HIGV_MSG_GESTURE_SCROLL/HIGV_MSG_GESTURE_PINCH

控件按下业务事件。

- 输入设备触发：wParam 为 GESTURE 事件结构体长度，lParam 为 TOUCH 事件结构体指针，同时还能触发控件对输入事件的响应。
- MSG 接口发送消息：与输入设备触发行为一致。

- 返回值影响：处理优先级为 HIGV_PROCORDER_BEFORE 的回调事件。
  - 如果返回值不为 HIGV_PROC_GOON 会中断控件对该消息的响应；
  - 如果返回值为 HIGV_PROC_GOON **消息会继续传递给控件父容器。**

3.2.6 内部消息处理列表

并不是所有控件的所有消息都会有内部处理，表 3-1 是 HiGV 对消息的内部处理列表。

表3-1 内部消息处理列表

Msg ID	Class	HiGV process
HIGV_MSG_PAINT	All	Paint
HIGV_MSG_SHOW	All	Show widget
HIGV_MSG_HIDE	All	Hide widget
HIGV_MSG_KEYDOWN	All	Switch HIGV_FOCUS_STATE_E; Enter; Switch focus
HIGV_MSG_LAN_CHANGE	All	Renewedly paint
HIGV_MSG_DATA_CHANGE	ADM Listbox Spin Combobox Scrollgrid	Data update
HIGV_MSG_STATE_CHANGE	Containerwidget	Change HIGV_STATENAME_E
HIGV_MSG_ST_UPDATE	Listbox Multiedit Scrolltext	Listbox:update scroll text Scrolltext:paint Multiedit:update cursor
HIGV_MSG_SCROLLBAR_CHANGE	Listbox Multiedit Scrollview	Update scrollbar
HIGV_MSG_REFRESH_WINDOW	Window	Refresh window
HIGV_MSG_FORCE_REFRESH_WINDOW	Window	Force update the window to the screen

Msg ID	Class	HiGV process
HIGV_MSG_MOUSEIN	All	Switch HIGV_SKIN_HIGHLIGHT
HIGV_MSG_MOUSEDOWN	All	Switch HIGV_SKIN_MOUSEDOWN
HIGV_MSG_MOUSEOUT	All	Lost HIGV_MSG_MOUSEIN
HIGV_MSG_MOUSEUP	All	Lost HIGV_MSG_MOUSEDOWN
HIGV_MSG_MOUSEMOVE	All	Move event
HIGV_MSG_MOUSEDBCLICK	All	Mouse event
HIGV_MSG_MOUSEWHEEL	All	Mouse event
HIGV_MSG_MOUSELONGDOWN	All	Mouse event
HIGV_MSG_TOUCH	All	Touch event
HIGV_MSG_GESTURE_TAP	All	Gesture event
HIGV_MSG_GESTURE_LONGTAP	All	Gesture event
HIGV_MSG_GESTURE_FLING	All	Gesture event
HIGV_MSG_GESTURE_SCROLL	All	Gesture event
HIGV_MSG_VALUEONCHANGE	Trackbar	Value change event
HIGV_MSG_FINISHSEEK	Trackbar	Finish seek event
HIGV_MSG_UNLOCK	Slideunlock	Unlock event
HIGV_MSG_MOVE	Slideunlock	Move event
HIGV_MSG_KICKBACK	Slideunlock	Kick back to origin event

3.2.7 Xml 定义的消息回调事件

Xml 文件可以注册常用的消息回调，省去使用接口 HI_GV_Widget_SetMsgProc（没有的可以通过接口注册），适用于所有控件的回调事件如表 3-2 所示。

表3-2 xml 消息回调列表

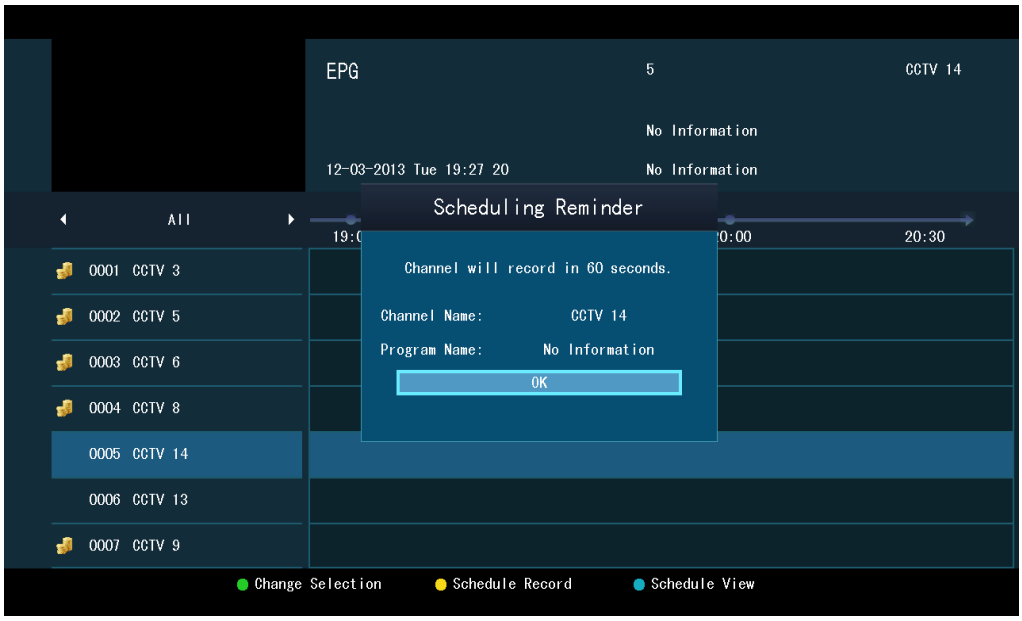
Xml attribute	Msg ID	Message order
onkeydown	HIGV_MSG_KEYDOWN	HIGV_PROCORDER_AFTER
onkeyup	HIGV_MSG_KEYUP	HIGV_PROCORDER_AFTER
ontimer	HIGV_MSG_TIMER	HIGV_PROCORDER_AFTER

Xml attribute	Msg ID	Message order
ongetfocus	HIGV_MSG_GET_FOCUS	HIGV_PROCORDER_AFTER
onlostfocus	HIGV_MSG_LOST_FOCUS	HIGV_PROCORDER_AFTER
onmousein	HIGV_MSG_MOUSEIN	HIGV_PROCORDER_AFTER
onmousedown	HIGV_MSG_MOUSEDOWN	HIGV_PROCORDER_AFTER
onmouseout	HIGV_MSG_MOUSEOUT	HIGV_PROCORDER_AFTER
onmouseup	HIGV_MSG_MOUSEUP	HIGV_PROCORDER_AFTER
onmousemove	HIGV_MSG_MOUSEMOVE	HIGV_PROCORDER_AFTER
onmousewheel	HIGV_MSG_MOUSEWHEEL	HIGV_PROCORDER_AFTER
onmousedbclick	HIGV_MSG_MOUSEDBCLICK	HIGV_PROCORDER_AFTER
onfocuseditexit	HIGV_MSG_FOCUS_EDIT_EXIT	HIGV_PROCORDER_BEFORE
onlanchange	HIGV_MSG_LAN_CHANGE	HIGV_PROCORDER_AFTER
ontouchaction	HIGV_MSG_TOUCH	HIGV_PROCORDER_AFTER
ongesturetap	HIGV_MSG_GESTURE_TAP	HIGV_PROCORDER_AFTER
ongesturelongtap	HIGV_MSG_GESTURE_LONGTAP	HIGV_PROCORDER_AFTER
ongesturefling	HIGV_MSG_GESTURE_FLING	HIGV_PROCORDER_AFTER
ongesturescroll	HIGV_MSG_GESTURE_SCROLL	HIGV_PROCORDER_AFTER
onvaluechange	HIGV_MSG_VALUEONCHANGE	HIGV_PROCORDER_BEFORE
onfinishseek	HIGV_MSG_FINISHSEEK	HIGV_PROCORDER_BEFORE
onunlock	HIGV_MSG_UNLOCK	HIGV_PROCORDER_BEFORE
onmove	HIGV_MSG_MOVE	HIGV_PROCORDER_BEFORE
onkickback	HIGV_MSG_KICKBACK	HIGV_PROCORDER_BEFORE

3.3 绘制

图 3-3 是一个 HiGV 界面，如何在屏幕上呈现这样较为复杂的完整场景界面呢，本章将会对单个控件的绘制，窗口与子控件的绘制关系等内容作详细讲解。

图3-3 HiGV 界面图示



3.3.2 控件绘制

控件的外观是界面图形的具体表现，控件的外观由皮肤和文字组成，其绘制处理分同步和异步两种。HiGV 内部数据改变、用户调用绘制接口、用户向控件发送绘制消息可以触发控件绘制。

一些操作可能需要重新绘制已经显示的控件，比如，动态为 label 控件设置文本，为控件设置新文本后需要重绘才能看到新文本。

HI_GV_Widget_Paint 接口触发的绘制为异步绘制，其基本流程如下：

步骤 1 计算绘制区域。

步骤 2 发送绘制消息，区域为步骤 1 得到的绘制区域。

步骤 3 收到绘制消息后，透明风格或强制绘制父风格的非窗口控件会先绘制父容器的背景。

步骤 4 回调优先级为 HIGV_PROCORDER_BEFORE 的消息回调事件。

步骤 5 控件绘制匹配当前状态的皮肤。

步骤 6 绘制控件的其他内容(图片、文字等)。

步骤 7 回调优先级为 HIGV_PROCORDER_AFTER 的消息回调事件。

----结束

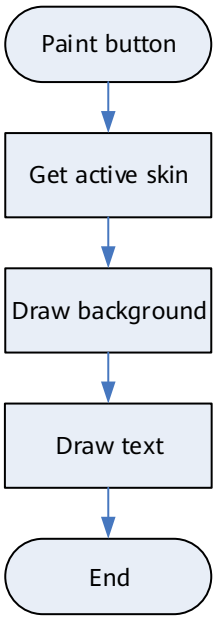
其他不同方式触发绘制：

- 调用发送消息接口发送绘制消息会直接从步骤 4 开始。
- 调用接口 HI_GV_Widget_Update 会省略发送消息的过程，同步完成绘制。

一般场景使用接口 HI_GV_Widget_Paint，因为该接口异步绘制不会阻塞主线程的其他处理，在部分特殊场景要求控件同步绘制的，可以调用 HI_GV_Widget_Update。

以图 3-3 的弹出框 OK 按钮为例，它的父容器为弹出窗口 Scheduling Reminder，在父窗口中的位置即为绘制矩形。

图3-4 OK 按钮绘制流程图



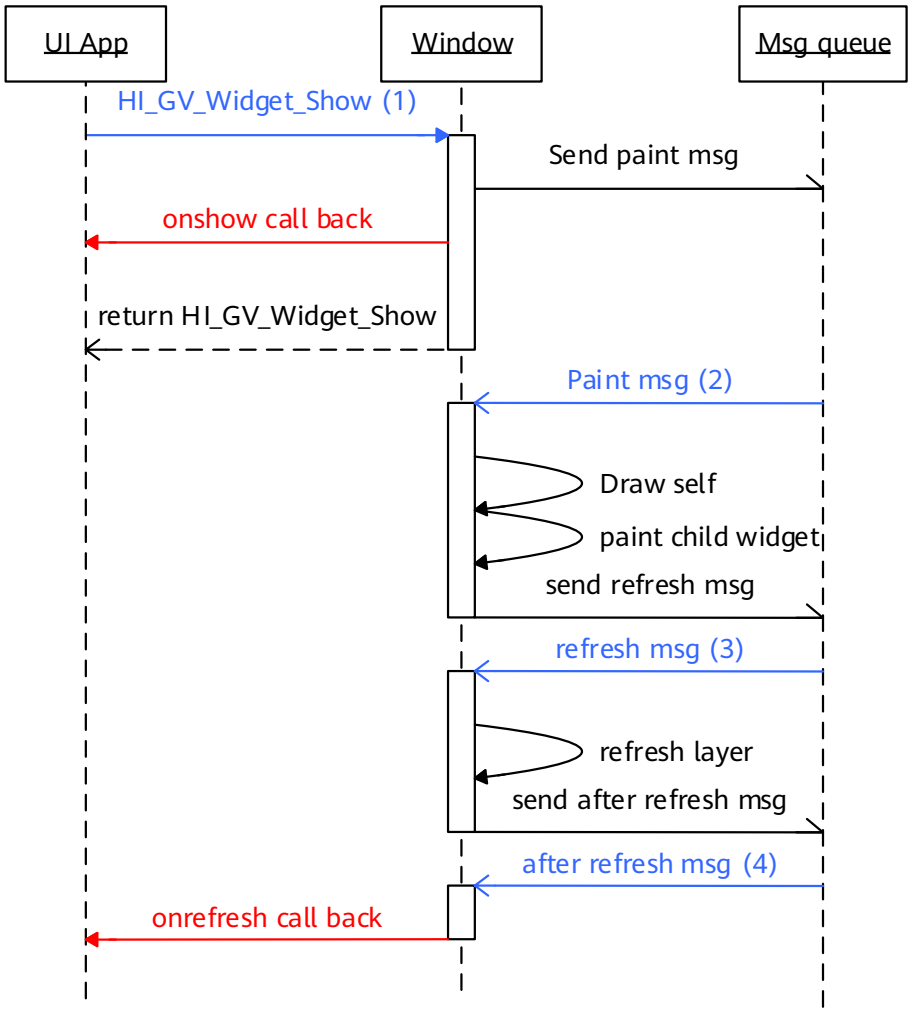
3.3.3 窗口绘制流程

须知

onshow 中不能调用接口 HI_GV_Widget_IsShow 获取该窗口显示状态，此时绘制还未成功，显示状态并不准确。也不要在 onshow 中隐藏自己，这会导致接下去的绘制和显示状态发生异常。

窗口显示流程如图 3-5 所示。

图3-5 窗口显示时序图



窗口的显示分为多个步骤：

步骤 1 调用 HI_GV_Widget_Show，如果该窗口为隐藏状态且可以显示，则向自己发送一个异步的绘制消息，异步绘制可以避免因为绘制时间较长导致短暂停滞。回调用户注册的onshow 事件后返回接口，**用户可以在 onshow 做窗口绘制之前的准备工作。**

步骤 2 收到绘制消息后，窗口绘制自己，再绘制显示状态的孩子控件，然后向自己发送一个刷新图层消息。

步骤 3 绘制和刷新图层都是异步消息，因此收到步骤 2 发送的刷新图层消息后，窗口会检查当前的绘制状态。

- 如果该窗口还在绘制过程中或已经隐藏则结束本条消息处理；
- 如果窗口状态符合刷新条件则刷新图层，并向自己发送一个已经完成刷新图层的消息，刷新图层后窗口已经显示在屏幕上了。

步骤 4 窗口收到步骤 3 发送的 after refresh 消息，回调用户注册的 onrefresh 事件，**用户可以在 onrefresh 做窗口绘制完成后的其他处理。**

----结束

3.3.4 隐藏控件

隐藏和显示是对应的，HI_GV_Widget_Hide 为隐藏控件接口。

绘制是建立在该控件处于显示状态的基础上，而控件的显示状态依赖父容器的显示状态。

非窗口控件的隐藏和显示要在其父容器已经是显示状态下才能进行，比如：一个 Window 控件中有一个 GroupBox 控件，GroupBox 中有一个 Button 控件。

- 当 Window 隐藏时，GroupBox 和 Button 也是隐藏状态并且无法显示；
- 当 Window 显示时，可以显示或隐藏 GroupBox；当 Window 显示且 GroupBox 显示时，可以显示或隐藏 Button。

控件的显示与隐藏时可能伴随资源加载和释放，请参考 3.5 资源。

3.4 定时器

定时器的作用是延期或定期触发事件。HiGV 的定时器是与控件绑定的，一个定时器只能绑定一个控件，而控件可以同时拥有多个定时器。定时器模块详细内容请参见头文件《hi_gv_timer.h》。

3.4.1 定时器的使用

定时器的使用步骤如下：

步骤 1 调用 HI_GV_Timer_Create(HI_HANDLE hWidget, HI_U32 TimerID, HI_U32 Speed) 接口创建定时器。

- hWidget：定时器绑定的控件。

- TimerID：定时器 ID，一个控件可以绑定多个定时器，绑定同一个控件的定时器有着不同的 TimerID，不同控件的定时器 TimerID 可以相同。
- Speed：定时器的触发时间，单位为 ms。

步骤 2 注册定时器回调事件。

Xml 文件的 ontimer 属性或 HI_GV_Widget_SetMsgProc 接口为定时器事件注册一个回调函数，OnTimer(HI_HANDLE hWidget, HI_U32 wParam, HI_U32 lParam)。

- hWidget：定时器绑定的控件。
- wParam：定时器 ID，对应创建接口的 TimerID，用于区别绑定了多个定时器时，触发本次回调的定时器身份。
- lParam：0。

步骤 3 使用定时器。

HiGV 定时器是一次性定时器，即启动定时器后只会执行一次回调事件，再次激活需要重启启动。

- HI_GV_Timer_Start(HI_HANDLE hWidget, HI_U32 TimerID)启动定时器，如果定时器已经运行无法使定时器重新计时。
- HI_GV_Timer_Stop(HI_HANDLE hWidget, HI_U32 TimerID)停止定时器，如果非首次启动定时器。

说明
在 start 之前先 stop 可以避免定时器状态不正确时，导致无法正常启动定时器的问题。

- HI_GV_Timer_Reset(HI_HANDLE hWidget, HI_U32 TimerID)重启定时器，定时器处于**启动状态时**重新计时，多用于在 OnTimer 中调用实现循环定时器。

步骤 4 销毁定时器。

调用 HI_GV_Timer_Destory(HI_HANDLE hWidget, HI_U32 TimerID)接口可以实现定时器销毁。定时器使用完毕应该及时销毁，以免浪费资源。

----结束

3.4.2 定时器与 HiGV 主线程关系

定时器启动（HI_GV_Timer_Start）后计算时间，定时器到达时间点后，向绑定的控件发送 HIGV_MSG_TIMER 事件。HiGV 主线程收到 HIGV_MSG_TIMER 后，分发至绑定控件，回调注册的 OnTimer 函数。也就是说回调函数是在 HiGV 主线程中执行的，保

证 HiGV 主线程顺畅才能保证准时地进入定时器回调函数，一些耗时较长的业务会延误定时器事件响应。

3.5 资源

HiGV 提供了统一资源管理机制，主要资源包括：字体、图片、皮肤和动画信息。资源管理模块通过对资源的引用计数，使得多处使用同一资源时只需要加载一次即可，这样可以避免造成系统内存和性能的浪费。HiGV 资源管理的对外接口统一在《hi_gv_resm.h》中。

3.5.1 资源的创建

要使用字库或图片，首先需要将其转为 HiGV 识别的资源。

接口 HI_GV_Res_CreateID(const HI_CHAR* pFileName, HIGV_RESTYPE_E ResType, HI_RESID* pResID)根据资源路径和选择的资源类型创建 res，并为其分配一个 ResID。

- 资源路径是相对于最终的可执行程序。
- ResID 是用来识别资源身份的标识，功能和控件句柄的功能类似。

为方便资源路径配置，HiGV 提供了资源环境变量设置。在控制台中输入：

- export HIGV_RES_IMAGE_PATH=xxx 配置图片资源路径前缀。
- export HIGV_RES_FONT_PATH=xxx 配置字体资源路径前缀。

如 export HIGV_RES_IMAGE_PATH=./res/image/，pFileName 为“button.png”，那么 HiGV 会以“./res/image/button.png”作为资源路径，注意“/”不要重复。配置了资源环境变量后，接口 HI_GV_Res_CreateID_NoPrefixPath 可以创建不读取环境变量的 ResID。

3.5.2 资源的使用

字体

```
typedef struct hiHIGV_FONT_S
{
    HI_RESID SbFontID;
    HI_RESID MbFontID;
    HI_U32    Size;
```



```
    HI_BOOL    bBold;
    HI_BOOL    bItalic;
}HIGV_FONT_S;
```

- HIGV_FONT_S 是创建字体的信息结构。
- SbFontID 和 MbFontID 表示两个物理设备字库创建的资源 ID，由 HI_GV_Res_CreateID 得来。
- Size：字体的大小。
- bBold：粗体。
- bItalic：斜体。

至少需要一个**字体资源**才能创建字体，创建成功后将字体句柄设置给控件实例，代码示例：

```
#define SBFONT_FILE "./res/sbfont.ttf"
#define MBFONT_FILE "./res/mbfont.ttf"

HI_S32 AppCreateSysFont(HI_HANDLE *pFont)
{
    HI_S32 Ret;
    HI_RESID SbFont, MbFont;
    HI_HANDLE hFont;
    HIGV_FONT_S FontInfo;

    Ret = HI_GV_Res_CreateID(SBFONT_FILE, HIGV_RESTYPE_FONT, &SbFont);
    if (HI_SUCCESS != Ret)
    {
        return Ret;
    }

    Ret = HI_GV_Res_CreateID(MBFONT_FILE, HIGV_RESTYPE_FONT, &MbFont);
    if (HI_SUCCESS != Ret)
    {
        HI_GV_Res_DestroyID(SbFont);
        return Ret;
    }

    FontInfo.MbFontID = MbFont;
```



```
    FontInfo.SbFontID = SbFont;
    FontInfo.Size = 22;
    FontInfo.bBold = HI_FALSE;
    FontInfo.bItalic = HI_FALSE;

    Ret = HI_GV_Font_Create((const HIGV_FONT_S *)&FontInfo, &hFont);
    if (HI_SUCCESS != Ret)
    {
        HI_GV_Res_DestroyID(SbFont);
        HI_GV_Res_DestroyID(MbFont);
        return Ret;
    }

    *pFont = hFont;
    return HI_SUCCESS;
}
```

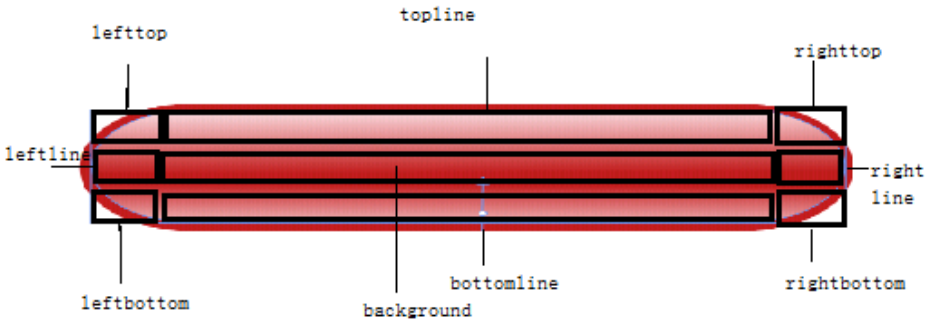
图片和皮肤

```
typedef struct hiHIGV_STYLE_S
{
    HIGV_STYLE_TYPE_E       StyleType;
    HIGV_STYLE_MEMBER_U     Top;
    HIGV_STYLE_MEMBER_U     Bottom;
    HIGV_STYLE_MEMBER_U     Left;
    HIGV_STYLE_MEMBER_U     Right;
    HIGV_STYLE_MEMBER_U     LeftTop;
    HIGV_STYLE_MEMBER_U     LeftBottom;
    HIGV_STYLE_MEMBER_U     RightTop;
    HIGV_STYLE_MEMBER_U     RightBottom;
    HIGV_STYLE_MEMBER_U     BackGround;
    HI_COLOR                FontColor;
    HI_U32                  bNoDrawBg;
    HI_U32                  LineWidth;
} HIGV_STYLE_S;
```

HIGV_STYLE_S 是创建皮肤的信息结构，HiGV 控件皮肤由九宫格组成，分为色块皮肤和图片皮肤两种类型：

- 如果是色块皮肤，则 HIGV_STYLE_MEMBER_U 为十六进制的 ARGB 颜色值；如果是图片皮肤，则 HIGV_STYLE_MEMBER_U 为图片资源 ID。
- FontColor 决定控件的字体颜色，我们也称 FontColor 为**前景色。**
- bNoDrawBg 决定是否绘制皮肤的背景，设置为真时皮肤不会绘制 BackGround 部分。
- LineWidth：色块皮肤外框线的宽度。

图3-6 九宫格皮肤示意图



图片资源除了作为皮肤的组成元素外，还能直接作用于一些使用到图片的控件，如 image、scrollbar、spin 等控件。

动画信息

```
typedef struct hiHIGV_ANIM_INFO_S
{
    HI_U32              AnimHandle;
    HI_U32              DurationMs;
    HI_U32              RepeatCount;
    HI_U32              DelayStart;
    HIGV_ANIM_TYPE_E AnimType;
    union
    {
        HIGV_ANIM_TRANSLATE_INFO_S Translate;
        HIGV_ANIM_ALPHA_INFO_S Alpha;
        HIGV_ANIM_ROLL_INFO_S Roll;
        HIGV_ANIM_ANY_INFO_S Any;
    }AnimParam;
```



```
} HIGV_ANIM_INFO_S;
```

- HIGV_ANIM_INFO_S 是创建动画的数据结构。
- AnimHandle 动画信息句柄。
- DurationMs：动画持续时间。
- RepeatCount：循环次数，-1 为无限循环。
- RepeatMode：循环模式。0 表示从新开始，1 表示从结束位置反向。该参数在 RepeatCount >0 或等于-1 有效。

动画信息建议在 XML 里面定义（可以由代码创建但不推荐），有了动画信息句柄后，可以创建动画并启动动画，代码示例：

```
HI_S32 AppCreateStartAnimation(HI_HANDLE *phAnim)
{
HI_S32 s32Ret;

//创建动画
// ANIM_BTN_MOVE 动画信息句柄，ANIM_BUTTON：控件句柄；phAnim：动画句柄.
    s32Ret = HI_GV_Anim_CreateInstance(ANIM_BTN_MOVE, ANIM_BUTTON, phAnim);
if (HI_SUCCESS != s32Ret)
{
        printf("create animation error, s32Ret=%d\n", s32Ret);
    return s32Ret;
}

//启动动画
// WND_HANDLE：当前窗口句柄，BUTTON1：控件句柄；phAnim：动画句柄.
    s32Ret = HI_GV_Anim_Start(WND_HANDLE, *phAnim);
if (HI_SUCCESS != s32Ret)
{
        printf("start animation error, s32Ret=%d\n", s32Ret);
}
return s32Ret;
}
```


须知

资源不再使用时，请调用 HI_GV_Res_Destroy 接口销毁资源。

3.5.3 Xml 设置资源

使用 xml 创建字体，动画和皮肤，只需要在 xml 文件中设置资源路径，具体的创建工作由 HiGV 完成，非常便捷。

```
<font
    id="common_font_text_18"
    sbname="../resource/font/ttf/sbfont.ttf"
    mbname="../resource/font/ttf/mbfont.ttf"
    size="18"
    isbold=""
    isitalic=""/>

<translate
    id="ANIM_BTN_MOVE"
    duration_ms=”500”
repeatcount=”3”
repeatmode=”1”
    delaystart=”20”
    animtype=”0”
    fromx=”20”
    fromy=”6”
    tox=”252”
    toy=”260”
/>

skin
    id="button_default"
    type="pic"
    btmlineidx="./res/default/button_bottom.png"
    toplineidx="./res/default/button_top.png"
    llineidx="./res/default/button_midleft.png"
    rlineidx="./res/default/button_midright.png"
    ltopidx="./res/default/button_topleft.png"
    rbtmidx="./res/default/button_bottomright.png"
```



```
    rtopidx="./res/default/button_topright.png"
    lbtmidx="./res/default/button_bottomleft.png"
    bgidx="./res/default/button_mid.png"
    isnodrawbg="no"
    linewidth="2"
    fgidx=""/>

<skin
    id="gray_skin"
    type="color"
    btmlineidx="0xFFC0C0C0"
    toplineidx="0xFFC0C0C0"
    llineidx="0xFFC0C0C0"
    rlineidx="0xFFC0C0C0"
    ltopidx="0xFFC0C0C0"
    rbtmidx="0xFFC0C0C0"
    rtopidx="0xFFC0C0C0"
    lbtmidx="0xFFC0C0C0"
    bgidx="0xFFC0C0C0"
    isnodrawbg="no"
    linewidth="2"
    fgidx=""/>
```

Xml 属性设置对应接口的结构属性，如下示例中的 common_font_text_18、button_default、gray_skin 即为创建的字体和皮肤资源句柄，可以直接使用，如：

```
HI_GV_Widget_SetFont(hButton0, common_fonttext_18);//为hButton0控件设置字体
HI_GV_Widget_SetSkin(hButton0, HIGV_SKIN_NORMAL, button_default);//为hButton0设置普通皮肤
```

3.5.4 HiGV 资源加载与释放

加载与释放的资源主要是皮肤和图片，HiGV 中的控件加载资源有两种方式：

- 隐藏控件不释放资源，控件第一次显示时加载资源，直至控件注销（HI_GV_Widget_Destroy）释放资源。

  在内存充足的情况下，或部分使用率极高的控件建议使用这种方式，其**优点是再次显示控件不用花费时间加载资源，但需要使用更多内存。**

- 隐藏释放资源风格，控件显示时加载资源，隐藏即可释放资源。

  这种风格的**优点是节省内存，但因为每次显示都需要加载资源，控件显示速度有所降低。**

控件默认为隐藏释放资源，如果希望控件是隐藏不释放资源，可以在 xml 中设置控件通用属性 isrelease，isrelease 是 bool 类型，isrelease=”no”表示隐藏不释放风格。如果是接口创建控件，在创建控件时 HIGV_WCREATE_S 结构的 style 成员不要设置 HIGV_STYLE_HIDE_UNLODRES。

加载

加载控件时，如果皮肤为图片类型，将解码这个皮肤的所有图片成员；再加载控件的其他私有图片。HiGV 会对每一个资源的引用计数，每当资源被引用，它的引用计数加一。

显示控件必须要调用接口 HI_GV_Widget_Show()，如果显示的是一个容器控件，则会加载该容器所有显示状态的子控件。前文的消息介绍中提到过 HIGV_MSG_SHOW 事件，该事件在控件显示时触发并回调注册事件函数(onshow 事件)。加载资源、onshow 事件、发送消息 HIGV_MSG_PAINT 的触发顺序是：

- 加载资源；
- 资源加载成功后，回调 onshow 事件函数；
- 发送绘制消息。

说明
隐藏不释放资源风格是在控件接收到 HIGV_MSG_PAINT 事件后，第一次绘制时加载资源再进行绘制的。

释放

因释放资源时可能有其他控件引用该资源，所以释放资源会对其引用计数递减，只有引用次数为零时，才是真正的释放该资源。

常驻

HI_GV_Res_SetResident(HI_RESID ResID)可以将图片及字体资源设为常驻，常驻资源不会释放，不能将皮肤直接设为常驻。

须知

如果是隐藏释放资源风格，在切换页面时可以先调用 HI_GV_Widget_Show 显示新界面，再调用 HI_GV_Widget_Hide 隐藏老界面，避免两个界面共用的资源释放后重新加载。

3.6 数据模型

ADM（Abastract Data Model）抽象数据模型，主要是为了显示批量的数据，使用控件有 ListBox、Scaleview、Charts、ScrollGrid 等。HiGV 采用界面显示与数据管理分离的原则，由控件实例显示数据，ADM 控制数据。

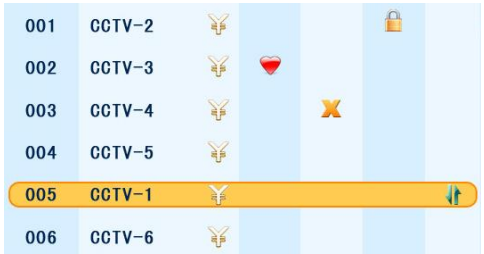
系统提供了一个默认数据缓存（DDB，全称：Default Data base）模块来进行基本的数据管理，用户还可以指定外部数据库，例如 SQLite DB 或者在嵌入式领域常用的 Berkeley DB 等。

数据模型可以与控件动态绑定或解绑，当数据模型数据发生改变时，可以通知与其绑定的控件自动刷新。数据模型的头文件为《hi_gv_adm.h》。

3.6.1 数据形态

如图 3-7 所示，这是一个绑定了数据模型的列表框，该列表框中的所有字串和图片都来源于数据模型。从图中可以看出，每列数据的类型是一样的，它对应的是数据模型中的一个字段；每一行都包括了所有类型的字段，称作一条数据。字段包括“数据类型和最大长度”两个元素，若干个字段组成完整的数据模型。

图3-7 列表框示例



字段在代码中的结构：

```
typedef struct hiHIGV_CELLATTR_S
{
```



```
    HIGV_DT_E eDataType;
    HI_U32 MaxSize;
}HIGV_FIELDATTR_S;

typedef enum
{
    HIGV_DT_S8 = 0, /**< char */
    HIGV_DT_U8,      /**< unsigned char */
    HIGV_DT_S16,     /**< short */
    HIGV_DT_U16,     /**< unsigned short */
    HIGV_DT_S32,     /**< int */
    HIGV_DT_U32,     /**< unsigned int */
    HIGV_DT_S64,     /**< long long */
    HIGV_DT_U64,     /**< unsigned long long */
    HIGV_DT_F32,     /**< float */
    HIGV_DT_D64,     /**< double */
    HIGV_DT_STRING, /**< char * */
    HIGV_DT_HIMAGE, /**< image handle */
    HIGV_DT_STRID,   /**< multi-langugae   string ID*/
    HIGV_DT_BUTT
} HIGV_DT_E;
```

- 从 HIGV_DT_S8 至 HIGV_DT_D64，其长度为数字除以 8，比如 HIGV_DT_S8 长度为 1，HIGV_DT_D64 长度为 8；
- HIGV_DT_STRING 是字串类型，其长度为字符串的最大长度，请设置为 4 的整数倍；
- HIGV_DT_HIMAGE 和 HIGV_DT_STRID 为图片资源 ID 以及多语言字串 ID，长度为 4。

3.6.2 ADM 使用

ADM 的数据来源有两种：

- OwnDB：HiGV 提供的数据缓存 DDB，事先将数据存储到 DB 中，静态或数量较少数据适用这种方式；
- UserDB：使用用户指定的数据库，数据模型通过用户注册的回调函数获取数据。

无论是哪种方式，控件与 ADM 绑定，ADM 与数据库绑定，再由数据库管理数据。

图3-8 控件与数据模型绑定关系图

创建 ADM

ADM 的创建参数如下：

```
typedef struct hiADM_OPT_S
{
    HI_U32 DBSource;
    HI_U32 FieldCount;
    HIGV_FIELDATTR_S *pFieldAttr;
    HI_U32 BufferRows;
    GetCountFunc GetCount;
    GetRowValueFunc GetRowValue;
    RegisterDataChangeFunc RegisterDataChange;
    UnregisterDataChangeFunc UnregisterDataChange;
}HIGV_ADMOPT_S;
```

HIGV_ADMOPT_S 是创建数据模型的参数结构，其中：

- DBSource：OwnDB 时，ADM 所绑定的 HiGV 的 DB 句柄，UserDB 不使用。
- FieldCount：该 ADM 的字段总个数。
- pFieldAttr：每一个字段的属性，该指针节点个数应该等于 FieldCount。
- BufferRows：OwnDB 不使用，UserDB 时决定了每次从数据库获取的数据条目数。
- GetCount：OwnDB 不使用，UserDB 获取数据总条目数的函数。
- GetRowValue：OwnDB 不使用，UserDB 获取数据的函数。
- RegisterDataChange：创建 ADM 时的注册事件回调函数。
- UnregisterDataChange：销毁 ADM 时的反注册事件回调函数。

也就是说，OwnDB 必须设置 DBSource、FieldCount、pFieldAttr；UserDB 必须设置 FieldCount、pFieldAttr、BufferRows、GetCount、GetRowValue。

OwnDB

HiGV 的数据缓存头文件为《hi_gv_ddb.h》，对数据缓存的所有接口都在这个头文件中。

要使用 OwnDB 首先需要创建 DB，接口为：

```
HI_GV_DDB_Create(HI_U32 FieldCount, const HIGV_FIELDATTR_S* pFieldAttr, HI_HANDLE* phDDB)
```

- FieldCount：DB 的字段个数，该项应该和绑定的 ADM 相同。
- pFieldAttr：每一个字段的属性，该指针的节点个数应该等于 FieldCount。
- phDDB：创建成功后 HiGV 返回给用户的 DB 句柄，句柄是 DB 身份的标识。

将 DB 句柄作为 HIGV_ADMOPT_S 的 Dbsource（即设置为 ADM 的数据源）。可以通过 HI_GV_DDB_Append、HI_GV_DDB_Modify、HI_GV_DDB_Insert 等接口操作 DB 数据。

UserDB

用户指定的外部数据库管理数据，HiGV 不会对这些数据进行保存，当控件需要用到数据时，通知绑定的 ADM 通过用户注册的回调事件直接获取数据。

typedef HI_S32 (*GetCountFunc)(HI_U32 DBSource, HI_U32 *RowCnt)是获取数据总数回调函数。

- DBSource：ADM 绑定的 DB 句柄，因为用到的是用户指定的数据库，往往回调函数已经决定了所绑定的数据库，不需要用到这个参数。
- RowCnt：用于告知 HiGV 外部数据总条数(即数据形态提到的一条数据)。用户要自己实现这个函数，在函数中将数据总条数以*RowCnt 传出。

typedef HI_S32 (*GetRowValueFunc)(HI_U32 DBSource, HI_U32 Row, HI_U32 Num, HI_VOID *pData, HI_U32 *pRowNum)是获取具体数据的回调函数，需要用户实现这个函数，在此函数中把外部数据库的数据传给 HiGV 控件。

- DBSource：ADM 绑定的 DB 句柄，因为用到的是用户指定的数据库，往往回调函数已经决定了所绑定的数据库，不需要用到这个参数。
- Row：获取数据的起始行，因为控件绘制的区域可能不是从第零行开始，比如 listbox 翻页以后。
- Num：HiGV 希望获取到的行数。
- pData：用户组织的具体数据。

- pRowNum：实际传出的数据行数，可能小于参数 Num，还需要告知 HiGV 真实的数据条目。

绑定控件

目前可以绑定 ADM 的控件有：

- Listbox
- Scrollgrid
- ScaleView
- Charts

一个 ADM 可以同时绑定多个控件，而控件只能和一个 ADM 绑定，绑定了同一个 ADM 的控件使用着相同的数据源。

- 接口 HI_GV_Widget_BindDataSource 可以将控件和 ADM 互相绑定。
- 接口 HI_GV_Widget_UnbindDataSource 用于解绑。

同步数据

要将数据库的内容显示在控件上，需要通知控件从数据库中获取数据。OwnDB 和 UserDB 获取数据的方式不同，通知控件的方式相同。

如下 4 种方式的任何一种都能使新的数据内容显示到控件上，只需要选择其中一种，如果重复多次调用会造成不必要的时间消耗。

- HI_GV_Widget_SyncDB 通知控件同步并显示新的数据，只对单个控件生效。
- HI_GV_ADM_Sync 同步 ADM 绑定的所有控件，对 ADM 所有绑定的控件生效。
- 向控件发送 HIGV_MSG_DATA_CHANGE 消息，其效果与 1 相同。
- 向 ADM 发送 HIGV_MSG_ADM_DATACHANGE 消息，其效果与 2 相同。

同步数据会通知 ADM 在数据库中获取数据，并存储在缓存中，这些缓存是在 ADM 创建时申请的，相关控件绘制时，从缓存中获取数据。

3.6.3 数据模型 xml

使用 xml 文件编写界面可以一步到位创建好 ADM、DB、空的回调函数，并在控件创建时就绑定 ADM，非常便捷简单。

```
/** 数据模型描述*/
<datamodel
id = "datamodel_record"
```



```
field=" ID:u32:4;Name:string:20;HasExperience:u32:4;Mobile:string:12;hImg:resid:4"
datasource = "owndb"
getrowcount = ""
getrowvalue = ""
registerdatachange = ""
unregisterdatachange = ""/>
```

数据模型与 HIGV_ADMOPT_S 相似，需要注意的是：

- datamodel 的 id 是 ADM 句柄，而 DBsource 是 DDB 句柄。
- field 是数据模型中存储的数据字段。

  字段之间用分号“;”分隔开来，字段由三个成员组成，它们之间用冒号“:”隔开：

  - 第一个成员是于区分字段的名称，名称并不会影响到字段的实际功能，
  - 第二、三成员与 HIGV_FIELDATTR_S 相似，如 u32 对应 HIGV_DT_U32，具体可参考《HiGV 控件标签说明》。

  可以抽象的把这个数据模型的字段内容看做一个结构体：

```
struct
{
    HI_U32 ID;
    HI_CHAR Name[20];
    HI_U32 HasExperience;
    HI_CHAR Mobile[12];
    HI_RESID hImg;
}
```

这是一个绑定了数据模型“datamodel_record”，有着五列数据的 listbox，从第 0 列到第 4 列依次对应的是数据模型中的第 0 到第 4 个字段。

/**界面控件描述*/

```
<view
    id = "allwidget"
    onload = "onload"
    unload = "unload">
    <window
            id = "allwidget"
            top = "0"
            left = "0"
```



```
width = "720"
height = "576"
normalskin = "common_skin_previewbg"
isrelease = "yes"
isnofocus = "no"
islanchange = "yes"
iswinmodel = "no"
isskinforcedraw = "no"
opacity = "255"
onshow = "allwidget_window_onshow"
onhide = "allwidget_window_onhide"
onrefresh = "allwidget_window_onrefresh"
onevent = "allwidget_window_onevent"
onkeydown = "allwidget _window_onkeydown"
ontimer ="allwidget_window_ontime"
onlanchange="allwidget_window_lanchange"
layer = "LAYER_0" >
 <listbox
        id = "listbox_record"
        top = "150"
        left = "70"
        width = "480"
        height = "192"
        normalskin = "common_skin_buttonbg"
        transparent = "no"
        font = ""
        rownum = "6"
        colnum = "5"
        leftorderobj = ""
        rightorderobj = ""
        uporderobj = "allwidget_button1"
        downorderobj = "spin_array"
        datamodel = "datamodel_record"
        hlineheight = "2"
        hlinecolor = "0xFF0000FF"
        vlinewidth = "2"
        vlinecolor = "0xFF0000FF"
        rowselectskin = "commonpic_skin_rowsel1"
```



```
rownormalskin = "commonpic_skin_rowsel2"
oncellselect = "listbox_record_oncellsel">
<listcol
    id = "listbox_record_ID"
    colindex = "0"
    coltype = "text"
    colwidth = "30"
    colalignment = "center"
    colbinddb = "yes"
    coldbindex = "0"
    colimage = ""
    coldataconv = ""/>
<listcol
    id = "listbox_record_name"
    colindex = "1"
    coltype = "text"
    colwidth = "120"
    colalignment = "center"
    colbinddb = "yes"
    coldbindex = "1"
    coldataconv = ""/>
<listcol
    id = "listbox_record_experience"
    colindex = "2"
    coltype = "image"
    colwidth = "40"
    colalignment = "center"
    colbinddb = "yes"
    coldbindex = "2"
    colimage = "./res/Fav.gif"/>
<listcol
    id = "listbox_record_mobile"
    colindex = "3"
    coltype = "text"
    colwidth = "250"
    colalignment = "center"
    colbinddb = "yes"
    coldbindex = "3"
```



```
                coldataconv = ""/>
            <listcol
                id = "listbox_record_himg"
                colindex = "4"
                coltype = "image"
                colwidth = "40"
                colalignment = "center"
                colbinddb = "yes"
                coldbindex = "4"/>
        </listbox>
    </window>
</view>
```

/**主应用程序 app.c*/

```
typedef struct {
    HI_U32 ID;
    HI_CHAR Name[20];
    HI_BOOL HasExperience;
    HI_CHAR Mobile[12];
    HI_HANDLE hImg;
}TestRecord;
```

/**添加数据*/

```
static HI_S32 ListBox_GenerateRecord(HI_U32 Num)
{
    HI_S32 Ret;
    TestRecord Record;
    HI_U32 i;
    HIGV_DBROW_S DBRow;
    HI_HANDLE hDDB = INVALID_HANDLE;

    ListBox_CreateImgRes();

    DBRow.Size = sizeof(Record);
    DBRow.pData = &Record;

    HI_GV_ADM_GetDDBHandle(datamodel_record, &hDDB);
    printf("hDDB is %d\n", hDDB);
```



```
    HI_GV_DDB_EnableDataChange(hDDB, HI_FALSE);//关闭数据通知

    for(i = 0; i < Num; i++)
    {
        Record.ID = i;
        snprintf(Record.Name, 20, "Name%d", i);
        snprintf(Record.Mobile, 20, "Tel%d", i);
        Record.HasExperience      = (HI_BOOL)(i%2);
        Record.hImg = s_hRecord2Img[i%4];
        Ret = HI_GV_DDB_Append(hDDB, &DBRow);//添加数据
        assert(Ret == HI_SUCCESS);
    }

    HI_GV_DDB_EnableDataChange(hDDB, HI_TRUE); //打开数据通知
    return Ret;
}

HI_S32 main(HI_S32 argc, HI_CHAR *argv[])
{
...

    HI_GV_Widget_Show(allwidget);
    HI_GV_Widget_Active(allwidget);

    /**添加数据*/
    Ret = ListBox_GenerateRecord(20);
    assert(Ret == HI_SUCCESS);
    Ret = HI_GV_App_Start(hApp);
    if (HI_SUCCESS != Ret)
    {
        HI_GV_PARSER_Deinit();
        (HI_VOID)HI_GV_App_Destroy(hApp);
        HI_GV_Deinit();
        return NULL;
    }
```



```
    HI_GV_PARSER_Deinit();
    (HI_VOID)HI_GV_App_Destroy(hApp);
    HI_GV_Deinit();

    return 0;
}
```

须知

数据模型中的字段长度、类型一定要和写入数据模型中的一一对应。如：

- u32 和 resid 等类型长度必须为 4。
- “Text:string:12”代码中写入数据的字串长度必须为 12。
- 在批量添加数据记录过程中一般需要关闭数据通知消息函数，添加数据记录结束后使能数据通知。

在更新数据后需要立即刷新控件的，须调用接口将数据同步至控件：

HI_GV_Widget_SyncDB 同步某个控件的 DB 或 HI_GV_ADM_Sync 同步绑定该数据模型的所有控件；如果是非 HIGV 线程同步数据，直接调用以上接口可能发生段错误，正确做法是向控件发送 HIGV_MSG_ADM_DATACHANGE 消息，通知控件同步 DB。

3.7 多语言

HiGV 部分控件（Label、Button、Multiedit 等）可以设置并显示文本字串。HiGV 提供了整体界面切换语言环境的功能，即多语言。多语言头文件请参见《hi_gv_lan.h》。

3.7.1 多语言 xml 描述

多语言功能采用 xml 描述语言种类和多语言字符串，并利用 xml2bin 工具自动生成字符串资源文件。一个多语言字串 ID 对应所有语言环境下的字符串，只需要为控件设置多语言字串 ID，在不同的语言环境下就能显示对应的字符串文本。目前 HiGV 只支持 xml 生成多语言字串。

多语言 XML 示例：

```
<language
    id = "language_sample"
```



```
    languageinfo = "ru:russian;en:english;ar:arabic"
    locale = "en">


    <lanstr
    id = "STRID_AMPM_AM"
    en = "AM."
    zh = "上午"
    ar = "ص"
    />

    <lanstr
    id = "STRID_AMPM_PM"
    en = "PM."
    zh = "下午"
    ar = "م"
    />

    <strset
    id = "STRSET_AMPM"
    tt = "STRID_AMPM_AM;STRID_AMPM_PM;"
    />
    <timefmt
    id = "TIMEFMT_SHORT_TIME"
    en = "<tt> [h]:[m]:[s]"
    zh = "[H]:[m]:[s]"
    ar = "<tt> [h]:[m]:[s]"/>
</language>
```

示例说明如下：

- <language
  标签<language 是语言环境设定，多语言必须且只能有一个<language 标签。
  - 属性 Languageinfo 以 xx:xx;xx:xx;...的格式描述了多语言的种类和对应的名称缩写，其中使用“;”将每一个语言环境分隔开来。
    语言环境描述的“:”的左边是该语言的缩写，也是实际用到的语言环境标识，右边是语言的全称，其用作是告知开发人员，没有实际用途。

- Locale 表示本地语言，本地语言是最重要和常用的语言，它是 languageinfo 中描述的某个语言环境，必须设置。

- <lanstr

  多语言字串的详细描述。示例中的 STRID_AMPM_AM、STRID_AMPM_PM 等 id 即是可以使用的多语言字串 ID，可以通过接口 HI_GV_Widget_SetTextByID 为控件设置多语言 ID，en、zh、ar 对应 Languageinfo 中描述的语言环境。

  其中，locale 是不可以缺省的，必须设置字符串；其他语言环境可以为空，如果为空 HiGV 会找到 locale 代替它。

- <strset 和<timefmt

  专门为控件 clock 使用，不同国家地区显示时间的格式和字串有所不同，tt 是多语言字串 ID 组合，timefmt 为不同语言环境的时间显示顺序。

3.7.2 注册与切换语言环境

使用 Xml 文件描述多语言，必须在 c 代码中通过接口注册和使用它们。Xml 文件通过 xml2bin 工具的转换可以得到各语言的.lang 二进制文件，如 en.lang、ar.lang。在 HiGV 线程启动之前，可以进行如下操作：

- 调用接口 HI_GV_Lan_Register 将语言注册到 HiGV 系统。
- 调用接口 HI_GV_Lan_Change 实现语言之间的切换。

3.7.3 语言环境书写方向切换

阿拉伯语、希伯来语等部分语种是从右向左书写的，当语言环境切换引起整体文本书写方向也发生切换时，可能希望整体的控件布局也发生对应的改变。HiGV 为此提供了便捷的控件自动镜像功能。

接口 HI_GV_Widget_EnableMirror(HI_HANDLE hWidget, HI_BOOL bPosMirror, HI_BOOL bInteriorMirror)设置控件的自动镜像属性。

其中：

- bPosMirror 表示该控件在其父容器（窗口相对于图层）的坐标位置是否镜像；
- bInteriorMirror 表示该控件的内容（如文本对齐方式、表格的左右顺序）是否镜像。

发生语言环境变化时，HiGV 会根据变化前后的语言环境判断是否需要自动镜像控件。

3.7.4 多国语言支持

当前 HiGV 可以支持设置的多国语言如表 3-3 所示。

表3-3 多国语言支持列表

语言类型	多国语言宏定义
简体中文	LAN_ZH
繁体中文	LAN_ZH_TW
捷克文	LAN_CS
丹麦文	LAN_DA
德文	LAN_DE
希腊文	LAN_EL
英文	LAN_EN
西班牙文	LAN_ES
芬兰文	LAN_FI
法文	LAN_FR
意大利文	LAN_IT
日文	LAN_JA
韩文	LAN_KO
荷兰文	LAN_NL
葡萄牙文	LAN_PT
俄文	LAN_RU
瑞典文	LAN_SV
土耳其文	LAN_TR
波兰文	LAN_PO
越南文	LAN_VN

语言类型	多国语言宏定义
匈牙利文	LAN_HU
乌克兰文	LAN_UKR
罗马尼亚文	LAN_RO
克罗地亚文	LAN_HR
塞尔维亚文	LAN_SR
波黑文	LAN_BS
马其顿文	LAN_MK
保加利亚文	LAN_BG
爱沙尼亚文	LAN_ET
拉脱维亚文	LAN_LV
立陶宛文	LAN_LT
印度尼西亚文	LAN_IND

3.8 Xml 文件描述

前文中多次提到 xml 文件，本章节将对 xml 进行详细的介绍。

xml 文件可以用于描述界面控件、皮肤、多语言字串、字体、数据模型和常用回调事件等内容，再使用工具 xml2bin 将这些文件转为二进制（.bin 和.lang）文件，并生成句柄头文件、回调事件的头文件和回调事件的空函数 c 文件。

使用 xml 可以大量减少创建、设置各类控件和资源的代码，还可以统一为管理句柄、回调函数的设置。

xml 文件分为五类：

- Skin xml：皮肤描述文件，生成控件使用的皮肤，具体见 Xml 设置资源；
- Language xml：多语言描述文件，生成多语言字串，具体见多语言；

- Font xml：字体描述文件，字体决定了 UI 文字的大小、风格，具体见 Xml 设置资源；
- Datamodel xml：ADM 描述文件，具体见数据模型；
- View xml：界面控件描述文件，生成 HiGV 控件并注册常用的消息回调事件。通常会使用到多个视图 xml，将一些关系紧密的控件写在一个视图 xml 内，根据控件的功能为 xml 文件命名。

说明
详细的 xml 标签说明请参考《HiGV 的 xml 标签说明》。

3.8.1 Xml 界面描述

```
<view
id = “hello_sample”                    <!—视图名-->
onload = “”
unload = “”>
    <window
        id = “hello_sample”                <!—视图窗口名称-->
        top = “0”                    <!—控件坐标-->
        left = “0”
        width = “720”
        height = “576”
        normalskin = “commonpic_skin_colorkey” <!—普通皮肤-->
        transparent = “no”
        isrelease = “yes”
        opacity = “255”
        winlevel = “0”
        onshow = “hello_window_onshow”>   <!—注册的窗口onshow事件回调-->
        <button
            id = “hello_button_ok”            <!—确定按钮名称-->
            top = “285”
            left = “235”
            width = “246”
            height = “40”
            normalskin = “commonpic_normalskin_button”
            disableskin = “”
            highlightskin = “”
```



```
            activeskin = “commonpic_activeskin_button”      <!—焦点状态皮肤-->
            transparent = “no”
            isrelease = “no”
            text = “ID_STR_OK”          <!—多语言字符串ID-->
            alignment = “hcenter|vcenter”
            onclick = “hello_button_onclick”/>   <!—按钮的onclick事件回调-->

        <label
            id = “hello_label_helpinfo”     <!—文本框ID-->
            top = “200”
            left = “235”
            width = “246”
            height = “50”
            normalskin = “common_skin_black”
            disableskin = “”
            highlightskin = “”
            activeskin = “”
            transparent = “yes”
            isrelease = “no”
            text = ""
            alignment = "hcenter|vcenter"/>
    </window>
</view>
```

- 标签：一项描述以标签开头，如<window>。“>”表示控件本身的描述结束，后面可能有子元素的描述内容，“</标签名>”或者“/>”为整个控件的结束符。通常容器类控件的结束符有标签名。如下：

```
<window
Window的描述>
<label>
Label的描述
/>
Window其他子控件的具体描述
</window>
```


标签名决定了该项描述的类型。可以将所有界面描述按模块或功能拆分成多个 xml 文件，便于修改和管理。

- **View：**实际上 view 并不是一个控件实例，view 为界面描述 xml 提供一个身份标识，HiGV 提供了接口 HI_GV_PARSER_LoadViewById 通过加载 view 的 ID 创建该 view 内包含的的所有控件实例。

须知

window 必须在 view 内描述，必须有且只有一个 window 的 ID 和 view 的 ID 是相同的。

3.8.2 Xml 解析

通过 xml2bin 工具，xml 描述信息生成二进制文件 higv.bin 和 xx.lang；头文件 higv_cextfile.h 和 higv_language.h，及 higv_cextfile.h 对应的 higv_cextfile.c；如果在视图 xml 注册了回调事件，还会生成一个同名的 c 文件，如 view.xml 生成对应的 view.c 文件，这些文件中有回调事件的空函数，另外数据模型也会生成一个 c 文件。

- 多语言字串的句柄声明在 higv_language.h 中。
- 控件、皮肤等句柄，通过 xml 直接注册的事件回调函数声明在 higv_cextfile.h 中。
- 多语言信息保存在后缀名为 lang 的二进制文件。
- Xml 界面、皮肤、数据模型描述的所有信息，都保存在二进制文件 higv.bin 中，通过调用接口 HI_GV_PARSER_LoadFile 加载 higv.bin 文件，可以获得除多语言字串外的所有 xml 描述信息。

4 控件详细介绍

4.1 Window 控件

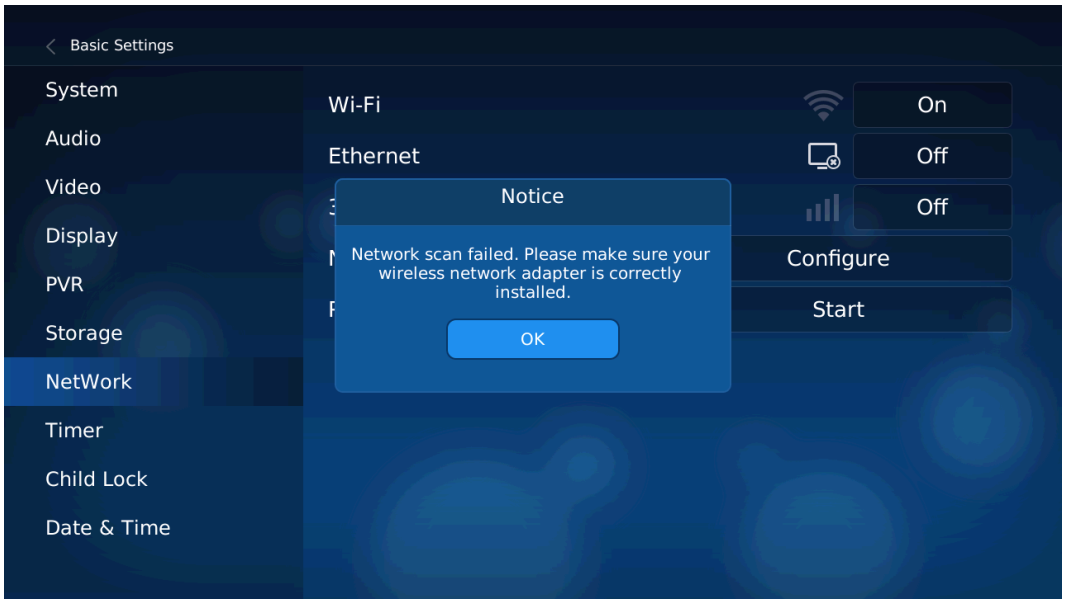
窗口(window)是图层的绘制矩形区域，它的坐标矩形就是映射到图层的区域。图层中可以同时存在若干个窗口，他们之间可以互相重叠。由 HiGO 统一管理窗口的重叠关系，在窗口中创建和使用具体的功能控件，制作出丰富的界面图形将相关业务与具体控件关联起来。窗口的头文件为《hi_gv_win.h》。

4.1.1 窗口重叠

窗口往往有着不同的坐标和大小，它们之间存在不规则的重叠，就像两张纸叠加在一起，重叠一定有上下层关系。A、B 两个窗口有重叠部分时，这两个窗口的 Z 序（window level）决定了它们显示的上下层关系。Z 序范围为 0～15，一共 16 层，Z 序越高显示等级越高，即重叠部分显示 Z 序高的窗口，如果两个窗口的 Z 序相同，焦点窗口的显示优先级更高。

图 4-1 中有多个窗口重叠，Notice 作为弹出框 Z 序更高，所占位置完全覆盖主页面，窗口中的子控件绘制等级由窗口决定。

图4-1 界面示例



4.1.2 窗口与 surface

当窗口 xml 属性 isrelease=”yes”（接口创建为 HIGV_STYLE_HIDE_UNLODRES 风格）时，surface 会在窗口显示的时候创建，隐藏的时候销毁；否则 surface 在窗口创建时一并创建，直至控件销毁（HI_GV_Widget_Destroy）时释放。

4.1.3 窗口私有属性

- opacity

  在非共享模式下，窗口可以设置透明度，透明度影响到窗口内的所有控件。透明度的有效范围为 0~255，0 为全透明，255 为完全不透明。xml 属性 opacity 或接口 HI_GV_Win_SetOpacity 可以设置该值。

- pixelformat

  窗口可以为自己设置专有的像素格式，缺省为所属图层的像素格式。xml 属性 pixelformat 可设置该值。

- onrefresh

  窗口绘制会触发刷新图层事件，onrefresh 为刷新图层后的回调函数。

- onshow

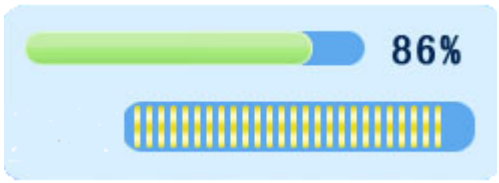
  显示触发事件，在控件绘制之前的回调函数。在 xml 中只有窗口可以直接设置 onshow 事件，非窗口可以调用接口 HI_GV_Widget_SetMsgPorc 注册 HIGV_MSG_SHOW 消息事件。

4.2 GroupBox 控件

GroupBox 是一个控件容器，里面可以放置其它的任何控件(除窗口外)。GroupBox 通常用作窗口内部组织控件群，在窗口内区别管理组合框的子控件，它没有独特的属性和事件。

图 4-2 是一个包含了两个 ProgressBar 控件和一个 Label 控件的 GroupBox。

图4-2 GroupBox 图示



4.3 Button 控件

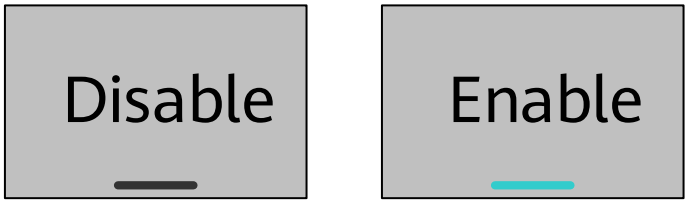
Button 是一个比较常用的控件，它通常用来响应用户的按下操作，链接终端和业务处理的交互。按钮的头文件为《hi_gv_button.h》。

Button 可支持不同状态下背景皮肤设置，同时还支持显示前景文字，其中前景文字的对齐方式可配置。不同的 Button 类型有不同的功能和用途，图 4-3~图 4-5 所示是几种常用的 Button 类型。

图4-3 Normal 类型 Button 图示

图4-4 Switch 类型 Button 图示

图4-5 Toggle 类型 Button 图示



4.3.1 Button 类型

Button 有多种类型，不同类型的 Button 有不同的功能，根据需求选择 Button 类型。

- 普通按钮

  普通按钮是最简单常用的按钮，用作基础的输入设备交互处理。

- 单选按钮

  单选按钮通常多个同时使用，单选按钮有 check 状态，按下后会设置或取消 check 状态，该状态有不同的皮肤。相同父容器的单选按钮中只能有一个是 check 状态，也因此把它称作单选。

- 复选按钮

  复选按钮和单选按钮一样也是多个同时使用，并且有 check 状态，但它与单选按钮不同在于相同父容器中的兄弟复选按钮可以同时有多个处于 check 状态，也因此把它称作复选。

- 开关按钮

  开关按钮可以设置两个文本字串分别用来表示“打开”和“关闭”，在不同状态下皮肤和字串的显示位置不同，两个字串文本的显示也是互斥的，同一时刻只有一个文本得以显示。

- 纽扣开关按钮

纽扣按钮和开关按钮的功能类似，不同的是纽扣开关的状态改变后的字串位置和皮肤不变，纽扣按钮有像棒状纽扣一样的颜色条用于区分开关状态。

- 软键盘按钮

  普通按钮用于软键盘时会自动转为软键盘按钮，它会存储一个字符作为按钮键值，具体了解请参考 5.7 输入法。

4.3.2 Button 私有属性

- Button 的私有皮肤

  Button 有 checkednormal、checkeddisable、checkedhighlight、checkedactive、checkedmousedown 五种皮肤用于匹配 check 时的各种状态。

- 文本边距

  Button 上的文本绘制区域默认为整个按钮范围，可以通过设置 Button 的上、下、左、右边距，虚拟指定文本在 Button 内的绘制区域，文本以绘制区域对齐。

- 开关、纽扣按钮设置

  开关按钮、纽扣按钮的打开、关闭状态时的字串设置，纽扣颜色，以及打开和关闭的布局的设置属性。

4.3.3 Button 的独特事件

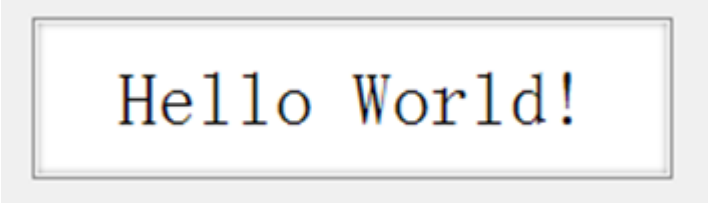
Check 状态切换可以设置一个回调函数 onstatuschange，用于响应状态改变。该事件对应的是 HIGV_MSG_BUTTON_STATUSCHANGE。

4.4 Label 控件

Label 是常用的文本基础控件，主要功能是显示静态文本，包括如下特点：

- 支持单行、多行文本显示。
- 支持多种对齐方式。
- 支持多行文本翻页。
- 支持与滚动条挂接。
- 不可获得焦点。

图4-6 Label 图示



Label 也可以和 Button 一样设置边距来约束文本绘制区域，除此之外 Label 没有其他独特的属性和事件。Label 的头文件为《hi_gv_label.h》。

4.5 Image 控件

Image 主要用于显示静态图片，Image 不能获得焦点，3.5 资源中介绍的图片资源可设置作为 Image 的显示内容。Image 的头文件为《hi_gv_image.h》。

图4-7 Image 图示

须知

图片分辨率过高会导致申请内存失败的情况发生，需结合系统资源的能力状态来使用 Image 控件显示静态图片。

4.5.1 存储在内存的图片

图片可能并不是以文件的形式存储，图片框还具有显示 MMZ 内存中的图片数据。只需要有内存图片的首地址和数据长度，就可以将图片显示出来。

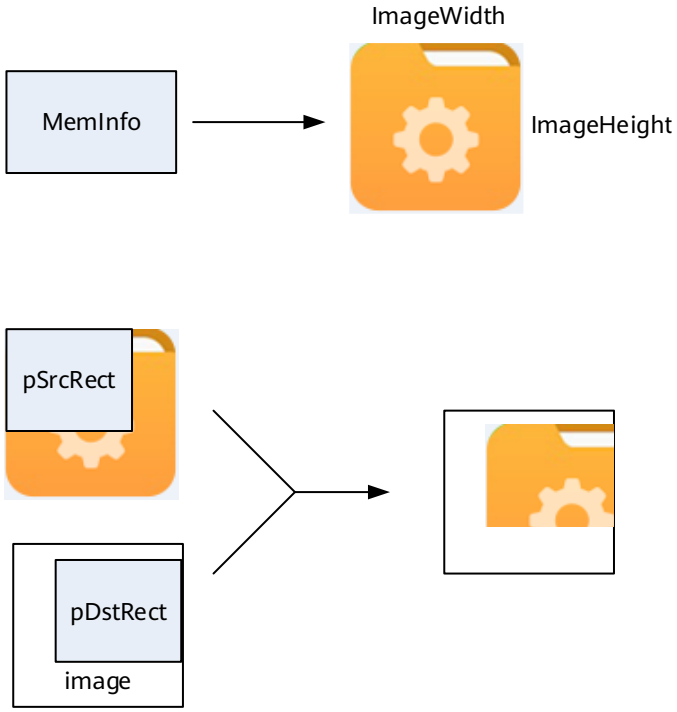
```
HI_GV_Image_DrawMemImage(HI_HANDLE hImage, HI_GV_MemInfo*  MemInfo,HI_U32 ImageHeight , HI_U32 ImageWidth, HI_RECT* pSrcRect, HI_RECT* pDstRect, HIGO_BLTOPT_S* pBlitOpt, HI_BOOL Transparent)
```

其中：

- hImage：Image 控件句柄。

- MemInfo：图片内存首地址和长度。
- ImageHeight：生成的图片高度。
- ImageWidth：生成的图片宽度。
- pSrcRect：图片的源矩形。
- pDstREct：显示到图片框的目标矩形。
- pBlitOpt：图片的搬移混合操作运算属性，如镜像、缩放、旋转等功能。
- Transparent：图片框的背景皮肤是否透明。

图4-8 内存图片说明图示



在贴图过程中会将 MemInfo 解码成为一块 surface，因此在绘制完成后，需要调用接口 HI_GV_Image_FreeMemSurface 将内存图片释放。遵循谁申请谁释放的原则，接口 HI_GV_Image_FreeMemSurface 并不会释放 MemInfo，该接口只会释放贴图过程中生成的图片 surface。

4.6 ImageEx 控件

ImageEx 主要用于显示动态 GIF 图片，图片显示支持居中和拉伸处理，如果待显示区域比图片小，该控件的显示将进行裁剪，ImageEx 不能获得焦点。ImageEx 的头文件为《hi_gv_imageex.h》。

ImageEx 功能包括：

- 设置图片显示类型。
- 动态图片的帧显示时间间隔。
- 动态图片的循环次数。

4.6.1 ImageEx 的独特事件

动态图片循环完成上报事件 onendofrepeat，对应 HIGV_MSG_IMAGEEX_ENDOF_REPEAT。

4.7 ListBox 控件

ListBox 主要用于显示批量数据，其中批量数据可存放在数据库或内存中。ListBox 的头文件为《hi_gv_listbox.h》。图 4-9 为 ListBox，其功能包括：

- 支持绑定数据模型。
- 每一列数据可以指定来自数据模型的某个字段，也可以是用户自定义。
- 每一个单元格可以显示图片或者文字，如果宽度为 0 则不显示。
- 支持多行多列。
- 每一列宽度可设置。
- 支持绑定滚动条。
- 支持上下按键进行焦点切换，翻页键进行翻页。
- 条目高亮状态、焦点状态背景可设置。
- 支持单元格焦点模式。
- 支持单元格焦点模式下自定义单元格宽度。
- 支持单元格焦点模式下滚动显示文本（文本内容必须超出单元格宽度）。
- 支持自左向右和自右向左两种滚动方式。
- 支持设置滚动步长。

- 支持设置滚动时间间隔。
- 支持选择是否绘制网格外框。
- 支持设置列文本字体颜色。
- 支持条目焦点模式下，切换焦点到顶或底条目时不循环焦点。
- 支持文本中添加 icon。
- 支持条目焦点模式设置滚动列。

图4-9 单元格 ListBox 图示

4.7.1 添加数据

ListBox 的数据依赖 ADM，添加数据请参考数据模型章节。

4.7.2 ListBox 私有属性

```
typedef struct
{
    HI_U32 RowNum;
    HI_U32 ColNum;
    HI_BOOL NoFrame;
    HI_BOOL Cyc;
    HI_BOOL IsCellActive;
    HI_BOOL AutoSwitchCell;
    HI_BOOL Scroll;
    HI_BOOL Fromleft;
    HI_U32 Timeinterval;
    HI_U32 Step;
    HI_U32 ScrollCol;
```



```
    HIGV_GET_WIDTH_CB GetWidthCb;
    HIGV_LIST_COLATTR_S *pColAttr;
} HIGV_LIST_ATTRIBUTE_S;
```

- 显示行数

  ListBox 在界面显示的行数 RowNum。

- 总列数

  ListBox 数据总列数 ColNum，对应着数据库的字段个数。

- 线框

  可以设置 ListBox 的横、竖线框的宽度和颜色，并且 NoFrame 控制是否绘制线框的外框。

- ListBox 的焦点切换

  当 ListBox 获得焦点时，可以控制 ListBox 的内部焦点切换。ListBox 内部焦点分为条目焦点和单元格焦点两种模式，默认为条目焦点模式。在按键控制焦点切换场景下，还可以设置是否循环焦点。循环焦点意为当焦点在边缘时再往边缘方向移动是否会跳至列表框另一端。

- 滚动字幕

  当单元格内的文本长度超过单元格长度时，可以令文本滚动显示，文本的滚动方向、滚动间隔、滚动步长都可以设置。

  条目焦点模式下，需指定 ScrollCol 滚动的列索引，焦点条目只有该列内容可以滚动。

  单元格焦点模式下，获得焦点的单元格可以滚动。

须知

请不要用半透明皮肤作为滚动的高亮条皮肤，这会导致滚动时与不滚动的高亮皮肤不一致。

- 列属性

  ListBox 的每一列对应数据库的一个字段，每一列的内容都不一样。需指定每一列的显示内容类型（如文本、图片、文本+图片），列宽，文本的对齐方式，文本的颜色，对应 ListBox 的字段索引等。

- 内部焦点皮肤

  为了区分内部焦点和非焦点区域，要设置焦点的皮肤，包括 ListBox 获得界面焦点时的 rowselectskin 皮肤和 ListBox 未获得界面焦点时的 rownormalskin 皮肤。

4.7.3 ListBox 独特事件

- 内部焦点切换事件 onselect，对应 HIGV_MSG_ITEM_SELECT，当内部焦点变化时发生。
- 焦点选中事件 oncellselect，对应 HIGV_MSG_LISTBOX_CELLSEL，当焦点被按下时发生。
- 数据变化事件 ondatachange，对应 HIGV_MSG_DATA_CHANGE，同步数据时发生。

4.7.4 单元格焦点

Xml 属性 iscellactive=”yes”可设置单元格焦点模式，接口创建须设置 IsCellActive。单元格焦点可以自定义每一个单元格的宽度，单元格焦点模式下注册一个单元格宽度获取函数 GetWidthCb 如下：

```
typedef HI_S32 (* HIGV_GET_WIDTH_CB)(HI_HANDLE hList,HI_U32 Item,HI_U32 Col)
```

该回调函数在 ListBox 同步数据时调用，调用次数为 ListBox 的总列数乘以总行数。每一次进入 GetWidthCb 函数，用户以返回值回馈希望设置的单元格宽度。返回值的取值范围是 0~100，意为占总行的百分比，0 表示隐藏单元格，100 表示该单元格占满整行。

如果没有注册 GetWidthCb，默认每一列的单元格宽度相等，为列属性设置的宽度。如果返回值不在 0~100 范围内，该单元格宽度使用前一次的值。

单元格焦点切换默认用户在 onkeydown 事件中通过接收按键调用接口设置。因为宽度为零的单元格虽然不能显示出来，但它是存在的，可能并不希望焦点切到不显示的单元格上。也可以设置 AutoSwitchCell 让 HiGV 内部处理焦点切换。

4.7.5 单元格小图标

ListBox 提供了单元格内设置小图标作为标记的功能。在很多场景中，ListBox 的某个单元格被按下时希望为该单元格做一个已被选中的标记，单元格的小图标可以实现这个功能。

单元格小图标在不影响单元格原有数据和线框的同时，在单元格左侧或右侧出现的微图片。这些图片的资源句柄也是通过 ADM 获取的。将列属性的类型设置为 LIST_COLTYPE_TXTICONLEFT 或 LIST_COLTYPE_TXTICONRIGHT，Fgidx 用 a | b 的方式表示 text 和 icon 对应的 ADM 字段索引。

有 icon 的列表框 XML 示例：


```
<listbox
isrelease="yes"
id="test_listbox01"
top="20"
left="20"
width="200"
height="150"
normalskin="black_skin"
disableskin = ""
activeskin="button_default"
transparent="no"
iscellactive="no"
widgetposmirror="yes"
widgetinteriormirror="yes"
oncellselect="listbox_cellselect"
rownum="5"
colnum="2"
font="common_font_text_22"
datamodel="test_datamodel1"
rowselectskin="common_skin_combobox_row_select"
rownormalskin="common_skin_combobox_row_select"
leftorderobj=""
rightorderobj=""
onkeydown=""
hlineheight="1"
vlinewidth="1"
hlinecolor="0xFF000000"
vlinecolor="0xFF000000"
scrollbar=""
ontimer=""
noframe=""
ongetfocus=""
onlostfocus=""
onselect="">
<!--coldbindex用text|icon的形式描述字段 -->
<listcol
id="test_listcol01"
```



```
coltype="texticonleft"
colwidth="100"
colalignment="hcenter|vcenter"
colbinddb="yes"
fgidx=""
coldbindex="0|1"
coldataconv=""/>
<!--coldbindex用text|icon的形式描述字段  -->
<listcol
id="test_listcol02"
coltype="texticonright"
colwidth="100"
colalignment="hcenter|vcenter"
colbinddb="yes"
fgidx=""
coldbindex="2|3"
coldataconv=""/>
</listbox>

<!--将listbox的icon存储在DB中，这需要DB预先设置存放图片资源ID的字段-->
<datamodel
    id="test_datamodel1"
    field="name:string:20;icon0:resid:4;num:u32:4;icon1:resid:4"
    datasource="owndb"
    getrowcount=""
    getrowvalue=""
    cacherows=""
    registerdatachange=""
    unregisterdatachange="" />
```

4.7.6 列回调函数

列属性中可以注册回调函数 HIGV_LIST_CONV_CB ConvString，在 xml 中为 coldataconv 属性。该回调本来是用作字符串的转换。当列表框绘制每一个单元格内容时，将数据库中存储的数据代入该回调，数据经过转换后再通过参数代出给 ListBox 用于显示。

因为每个单元格绘制都会回调此函数，可以巧妙地利用这一点。


```
typedef HI_S32 (*HIGV_LIST_CONV_CB)(HI_HANDLE hList, HI_U32 Col, HI_U32 Item, const HI_CHAR *pSrcStr, HI_CHAR *pDstStr, HI_U32 Length)
```

其中：

- Col 和 Item 可以知道当前调用函数的单元格准确位置
- pSrcStr 为当前单元格所对应的数据
- pDstStr 为 ListBox 最终显示数据
- Length 为 pSrcStr 的长度

当某列的数据都是同一个图片资源的时候，可以不绑定 ADM 字段，直接设置结构 HIGV_LIST_COLATTR_S 的属性 hImage，因为此时并不依赖数据库，列表框对这种用法有特殊的处理。

- pDstStr[0]=1 时显示图片
- pDstStr[0]=0 时隐藏图片

4.8 ScrollBar 控件

ScrollBar 用于绑定一些内容可能超出显示区域的控件，辅助控件显示和操作，指示当前界面显示区域在整个需要显示的区域中的位置。ScrollBar 不能获得焦点。ScrollBar 的头文件为《hi_gv_scrollbar.h》。

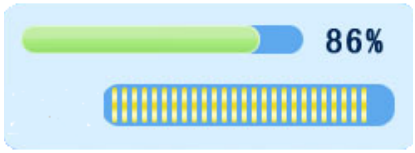
图4-10 ScrollBar 与绑定控件图示

ScrollBar 需要设置上下箭头资源、滑块资源及背景皮肤。ScrollBar 没有特殊事件，唯一需要注意的是滑块的起始偏移位置以背景皮肤的左上角图片决定，因此背景皮肤的左上角部分最好与上箭头保持一致。

4.9 ProgressBar 控件

ProgressBar 用于表示进度的变化和进度的完成程度，可用于表示文件压缩进度、电视节目播放进度等信息，ProgressBar 中的格子是用图片填充的。ProgressBar 头文件为《hi_gv_progressbar.h》。

图4-11 ProgressBar 图示



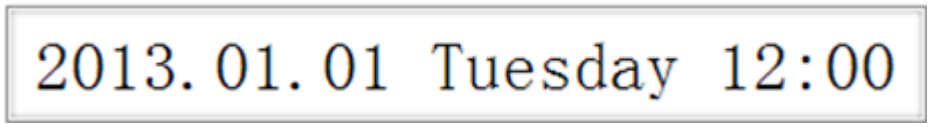
ProgressBar 有横、竖两种风格，它们的基本设置大致相同。ProgressBar 的背景色是控件本身的皮肤，除此之外还需要设置进度皮肤、步长、最大值及最小值。

ProgressBar 获取焦点时可以通过左右按键改变他的进度，改变进度时会产生事件，对应 HIGV_MSG_ITEM_SELECT。

4.10 Clock 控件

Clock 用于时间显示和日期，该控件即支持直接设置文本，也支持设置固定的格式用于显示。时钟的头文件为《hi_gv_clock.h》。

图4-12 时钟图示



4.10.1 Clock 定内部时器

在 Clock 显示之前，需为 Clock 设置最小单位。xml 创建时钟须指定属性 minunit；接口创建时钟，调用 HI_GV_Clock_Init 初始化时钟后还须要调用 HI_GV_Clock_SetTimeUnit 为时钟设置最小单位。如果设置最小单位为“秒”， Clock 会自动创建一个每秒触发一次的定时器；否则创建一个每分钟触发一次的定时器。

Clock 在加载资源时会创建内部定时器，也就是说：

- 当时钟是 HIGV_STYLE_HIDE_UNLODRES 风格时，在显示时创建定时器，隐藏时销毁定时器；
- 当时钟不是 HIGV_STYLE_HIDE_UNLODRES 风格时，创建 Clock 的同时会创建内部定时器，定时器生命周期与时钟相同。

定时器创建成功后，即刻循环启动并回调 HIGV_MSG_TIMER（ontimer）事件，定时器 ID 为 0x1001。接口 HI_GV_Clock_Run 控制 Clock 定时器的启动和停止，非 HIGV_STYLE_HIDE_UNLODRES 风格往往要先停止，显示控件时再启动。

须知

minunit 是必须设置的属性，如果没有设置闹钟的最小单位，控件不会自动创建定时器更新时间。

4.10.2 Clock 文本显示

创建 Clock 时须指定显示格式 dispmode，接口创建为参数 HIGV_CLOCK_MODE_EDispMode；xml 文件创建时为属性 dispmode。

设置文本模式

当 DispMode 为 HIGV_CLOCK_MODE_TEXT 时，可以通过接口 HI_GV_Widget_SetText 或 HI_GV_Widget_SetTextByID 为 Clock 设置时间。

固定格式模式

当 DispMode 为 HIGV_CLOCK_MODE_FORMAT 时，首先要为 Clock 设置固定的显示格式，目前只支持 xml 设置显示格式。显示格式通常定义在多语言 xml 文件中，以 timefmt 作为标签，将 timefmt 的 ID 作为控件的 text 属性。

注意：固定格式的 Clock 不要再通过接口设置文本。表 4-1 是 timefmt 中会被替换的特殊字符。

表4-1 Clock 显示格式定义

补全格式	缺省格式	含义	是否可编辑	隔离符号
yyyy	yy	year	Y	[ ]
MM	M	month	Y	[ ] ,< >

补全格式	缺省格式	含义	是否可编辑	隔离符号
dd	d	day	Y	[ ]
dddd	ddd	week	N	< >
HH	H	hour/2412 小时制	Y	[ ]
hh	h	hour/1212 小时制	Y	[ ]
mm	m	minute	Y	[ ]
ss	s	second	Y	[ ]
tt	-	a.m./p.m.	N	< >

表格中的字符在文本中会被替换成其他字符，用中括号[ ]或尖括号< >区别这些字符，其中：

- [ ]表示括号中的字符为数字
- < >表示括号中的内容是一个 ID 字符集

[<strset>fmt]类型表示 fmt 中的数字 0 到 9 会被 ID 字符集 strset 代替，这里的 strset 一定要是 10 个，顺序代替 0 到 9。

补全格式会显示完整的字串，缺省格式会将多余字符省略，比如“2015 年用 yyyy 显示为 2015 而 yy 则显示为 15”，又比如“9 号用 dd 会显示为 09 而 d 则显示为 9”。

以一段 xml 为例：

```
<language
    id = "language"
    languageinfo = "zh:chinese;en:english;ar:arabic"
    locale = "en">
     <strset
          id = "STRSET_MONTH"
          MM =
"STRID_MM_JAN;STRID_MM_FEB;STRID_MM_MAR;STRID_MM_APR;STRID_MM_MAY;STRID_MM_JUN;STRID_MM_JUL;STRID_MM_AUG;STRID_MM_SEP;STRID_MM_OCT;STRID_MM_NOV;STRID_MM_DEC;"
    />
    <strset
          id = "STRSET_SHORT_MONTH"
```



```
        M =
"STRID_MM_JAN;STRID_MM_FEB;STRID_MM_MAR;STRID_MM_APR;STRID_MM_MAY;STRID_MM
_JUN;STRID_MM_JUL;STRID_MM_AUG;STRID_MM_SEP;STRID_MM_OCT;STRID_MM_NOV;STRID_
MM_DEC;"
    />

    <strset
        id = "STRSET_AMPM"
        tt = "STRID_AMPM_AM;STRID_AMPM_PM;"
    />
    <strset
        id = "STRSET_DIGIT"
        digit =
"STRSET_DIGIT_0 ;STRSET_DIGIT_1;STRSET_DIGIT_2;STRSET_DIGIT_3;STRSET_DIGIT_4;STRSET_DI
GIT_5;STRSET_DIGIT_6;STRSET_DIGIT_7;STRSET_DIGIT_8;STRSET_DIGIT_9;"
    />

    <lanstr
        id = "STRSET_DIGIT_0"
        en = "0"
        zh = "零"
        ar = "٠"
    />
    <lanstr
        id = "STRSET_DIGIT_1"
        en = "1"
        zh = "壹"
        ar = "١"
    />
    <lanstr
        id = "STRSET_DIGIT_2"
        en = "2"
        zh = "贰"
        ar = "٢"
    />
    <lanstr
        id = "STRSET_DIGIT_3"
```



```
        en = "3"
        zh = "叁"
        ar = "٣"
    />
    <lanstr
        id = "STRSET_DIGIT_4"
        en = "4"
        zh = "肆"
        ar = "٤"
    />
    <lanstr
        id = "STRSET_DIGIT_5"
        en = "5"
        zh = "伍"
        ar = "٥"
    />
    <lanstr
        id = "STRSET_DIGIT_6"
        en = "6"
        zh = "陆"
        ar = "٦"
    />
    <lanstr
        id = "STRSET_DIGIT_7"
        en = "7"
        zh = "柒"
        ar = "٧"
    />
    <lanstr
        id = "STRSET_DIGIT_8"
        en = "8"
        zh = "捌"
        ar = "٨"
    />
    <lanstr
        id = "STRSET_DIGIT_9"
```



```
        en = "9"
        zh = "玖"
        ar = "٩"
    />

    <lanstr
        id = "STRID_AMPM_AM"
        en = "AM."
        zh = "上午"
        ar = "ص"
    />

    <lanstr
        id = "STRID_AMPM_PM"
        en = "PM."
        zh = "下午"
        ar = "م"
    />

    <lanstr
        id = "STRID_MM_JAN"
        en = "January"
        zh = "一月"
        />

    <lanstr
        id = "STRID_MM_FEB"
        en = "February"
        zh = "二月"
        />

    <lanstr
        id = "STRID_MM_MAR"
        en = "March"
        zh = "三月"
        />
```



```
<lanstr
    id = "STRID_MM_APR"
    en = "April"
    zh = "四月"
    />
<lanstr
    id = "STRID_MM_MAY"
    en = "May"
    zh = "五月"
    />

<lanstr
    id = "STRID_MM_JUN"
    en = "June"
    zh = "六月"
    />

<lanstr
    id = "STRID_MM_JUL"
    en = "July"
    zh = "七月"
    />

<lanstr
    id = "STRID_MM_AUG"
    en = "August"
    zh = "八月"
    />

<lanstr
    id = "STRID_MM_SEP"
    en = "September"
    zh = "九月"
    />
<lanstr
```



```
        id = "STRID_MM_OCT"
        en = "October"
        zh = "十月"
        />
    <lanstr
        id = "STRID_MM_NOV"
        en = "November"
        zh = "十一月"
        />

    <lanstr
        id = "STRID_MM_DEC"
        en = "sunday"
        zh = "十二月"
        />

    <lanstr
        id = "STRID_WEEK_SUN"
        en = "sunday"
        zh = "星期天"
        ar = "يوم الأحد"
        />

    <lanstr
        id = "STRID_WEEK_MON"
        en = "monday"
        zh = "星期一"
        ar = "يوم الاثنين"
        />

    <lanstr
        id = "STRID_WEEK_TUE"
        en = "Tuesday"
        zh = "星期二"
        ar = "يوم الثلاثاء"
        />
```



```
<lanstr
    id = "STRID_WEEK_WED"
    en = "wednesday"
    zh = "星期三"
    ar = "يوم الأربعاء"
    />

<lanstr
    id = "STRID_WEEK_THUR"
    en = "thursday"
    zh = "星期四"
    ar = "يوم الخميس"
    />

<lanstr
    id = "STRID_WEEK_FRI"
    en = "friday"
    zh = "星期五"
    ar = "يوم الجمعة"
    />

<lanstr
    id = "STRID_WEEK_SAT"
    en = "saturday"
    zh = "星期六"
    ar = "يوم السبت"
    />

<lanstr
    id = "STRID_WEEK_SHORT_SUN"
    en = "sun."
    zh = "周天"
    ar = "يوم الأحد"
    />
```



```
<lanstr
    id = "STRID_WEEK_SHORT_MON"
    en = "mon."
    zh = "周一"
    ar = "يوم الاثنين"
    />

<lanstr
    id = "STRID_WEEK_SHORT_TUE"
    en = "Tues."
    zh = "周二"
    ar = "يوم الثلاثاء"
    />

<lanstr
    id = "STRID_WEEK_SHORT_WED"
    en = "wed."
    zh = "周三"
    ar = "يوم الأربعاء"
    />

<lanstr
    id = "STRID_WEEK_SHORT_THUR"
    en = "thu."
    zh = "周四"
    ar = "يوم الخميس"
    />

<lanstr
    id = "STRID_WEEK_SHORT_FRI"
    en = "fri."
    zh = "周五"
    ar = "يوم الجمعة"
    />

<lanstr
```



```
        id = "STRID_WEEK_SHORT_SAT"
        en = "sat."
        zh = "周六"
        ar = "يوم السبت"
        />
    <strset
        id = "STRSET_SHORT_WEEK"
        ddd =
"STRID_WEEK_SHORT_SUN;STRID_WEEK_SHORT_MON;STRID_WEEK_SHORT_TUE;STRID_WEEK_
SHORT_WED;STRID_WEEK_SHORT_THUR;STRID_WEEK_SHORT_FRI;STRID_WEEK_SHORT_SAT;"/
>

    <strset
        id = "STRSET_WEEK"
        dddd =
"STRID_WEEK_SUN;STRID_WEEK_MON;STRID_WEEK_TUE;STRID_WEEK_WED;STRID_WEEK_THU
R;STRID_WEEK_FRI;STRID_WEEK_SAT;"/>


    <timefmt
        id = "TIMEFMT_DATE"
        en = "[yyyy]-<MM>-[dd] week: <dddd>"
        zh = "[yyyy]-<MM>-[dd] <dddd>"
        ar = "<dddd>,[dd] <MM>,[yyyy]"
        />
    <timefmt
        id = "TIMEFMT_TIME"
        en = "[HH]:[mm]:[s]"
        zh = "[H]:[mm]:[ss]"
        ar = "<tt> [hh]:[mm]:[s]"
        />


    <timefmt
        id = "TIMEFMT_DATETIME"
        en = "[yyyy].<MM>.[dd] week: <dddd>   [HH]:[mm]:[s]"
        zh = "[<digit>yyyy]年<MM> [dd]日  <dddd> <tt> [hh]:[mm]:[s]"
```



```
        ar = "<dddd>,[dd] <MM>,[yyyy] <tt> [h]:[mm]:[s]"
        />

    <timefmt
        id = "TIMEFMT_SHORT_TIME"
        en = "<tt> [h]:[m]:[s]"
        zh = "[H]:[m]:[s]"
        ar = "<tt> [h]:[m]:[s]"
        />

    <timefmt
        id = "TIMEFMT_SHORT_DATE"
        en = "[yy]-<M>-[d] week: <ddd>"
        zh = "[yy]-<M>-[d] <ddd>"
        ar = "<ddd>,[d] <M>,[yy]"
        />
    <timefmt
        id = "TIMEFMT_SHORT_DATETIME"
        en = "[yy].<M>.[d]  [H]:[m]:[s]"
        zh = "[<digit>yy]/<M>/[d] <tt> [h]:[m]:[s]"
        ar = "<ddd>,[d] <M>,[yy] <tt> [h]:[m]:[s]"
        />
</language>
```

时间格式 TIMEFMT_DATETIME

```
    <timefmt
        id = "TIMEFMT_DATETIME"
        en = "[yyyy].<MM>.[dd] week: <dddd>    [HH]:[mm]:[s]"
        zh = "[<digit>yyyy]年<MM> [dd]日  <dddd> <tt> [hh]:[mm]:[s]"
        ar = "<dddd>,[dd] <MM>,[yyyy] <tt> [h]:[mm]:[s]"
        />
```

其中：

- [yyyy]表示补全格式的数字年份如 2015。
- <MM>表示一到十二月，因为是< >，会找到 strset 中的 MM 显示文本。

```
    <strset
        id = "STRSET_MONTH"
```



```
        MM =
"STRID_MM_JAN;STRID_MM_FEB;STRID_MM_MAR;STRID_MM_APR;STRID_MM_MAY;STRID_MM
_JUN;STRID_MM_JUL;STRID_MM_AUG;STRID_MM_SEP;STRID_MM_OCT;STRID_MM_NOV;STRID_
MM_DEC;"
    />
```

strset 有 id= STRSET_MONTH，这个 ID 并不会被使用，xml2bin 工具会自动寻找到 MM 替换尖括号中的内容。在不同的语言环境下，<MM>会被翻译成十二个月的多语言字串，如一月对应的 STRID_MM_JAN 在 en 环境下会找到“January”，zh 环境下会找到“一月”，ar 环境下因为缺省会使用“January”。

- [dd]表示一个月当中的日期。
- <dddd>表示一周内的某天，它会自动搜索多语言字串 ID 对应的语言环境下的字符串，strset 中使用分号把每个多语言字串 ID 分隔开来。

```
    <strset
        id = "STRSET_WEEK"
        dddd =
"STRID_WEEK_SUN;STRID_WEEK_MON;STRID_WEEK_TUE;STRID_WEEK_WED;STRID_WEEK_THU
R;STRID_WEEK_FRI;STRID_WEEK_SAT;"/>
```

- [HH]表示 24 小时制的 0~24 小时。
- [hh]表示 12 小时制的 0~12 小时。
- [mm]表示 0~59 分钟，补全格式显示为“00”到“59”。
- [s]表示 0~59 秒，缺省格式显示为“0”到“59”。
- <tt>表示上午/下午，它会自动搜索多语言字符集 tt。
- [<digit>yyyy]表示补全格式的年份，其中的数字会找到字符集 digit 中对应语言环境下的字符。如 2015 年在 zh 环境下会显示为“贰零壹伍”。

假设现在的时间为 2015 年 4 月 7 日 15:40:05 星期二：

- "[yyyy].<MM>.[dd] week: <dddd> [HH]:[mm]:[s]"作为 en 语言环境的显示格式，最后会显示为“2015. April.07 week: Tuesday 15:40: 5”。
- "[<digit>yyyy]年<MM> [dd]日 <dddd> <tt> [hh]:[mm]:[s]"作为 zh 语言环境的显示格式，最后会显示为“贰零壹伍年四月 07 日 星期二下午 03:40:5”。
- "<dddd>,[dd] <MM>,[yyyy] <tt> [h]:[mm]:[s]"作为 ar 语言环境的显示格式，最后会显示为“07,يوم الثلاثاء April,2015 3:40:5 م”。

4.10.3 时间设置与修改

Clock 的内部定时器循环启动时会回调用户注册的事件函数。通常在如下回调事件中设置当前的时间：

- ontimer（对应 HIGV_MSG_TIMER）
- 或者 ontimeupdate（对应 HIGV_MSG_CLOCK_UPDATE）

这两个回调事件都可以用作更新事件，它们的区别是：

- ontimer 的代入参数为定时器 ID（0x1001）和零
- ontimeupdate 代入的参数是当前获取的系统时间和零

系统时间是强制转换为 HI_U32 类型的 HIGV_TIME_S 指针。

设置文本模式下，只需要对 Clock 进行文本设置即可，固定格式模式下在回调函数中将预设置的时间日期转换为 time_t，并调用接口 HI_GV_Clock_SetUTC 更新 Clock。

除了在回调事件中更新 Clock 时间，还可以通过按键调整时间。当 Clock 获取焦点时，按下 OK 键进入编辑状态。在编辑状态下，上下键可以修改当前光标显示位置的数字，左右键切换光标位置。其中“周”和上、下午是不可编辑的，它们由系统自动计算。编辑完成后，再次按下 OK 键退出编辑状态，可以切换控件之间的焦点。

4.10.4 Clock 私有属性

- 显示设置

  设置时间最小单位和显示模式，如果是固定格式模式还需要在 xml 中设置时间格式。

- 内部焦点设置

  内部焦点是为了区别当前编辑位置的光标，可以指定为色块或图片资源，根据内部焦点类型还需指定颜色值或图片资源路径。

4.10.5 Clock 的独特事件

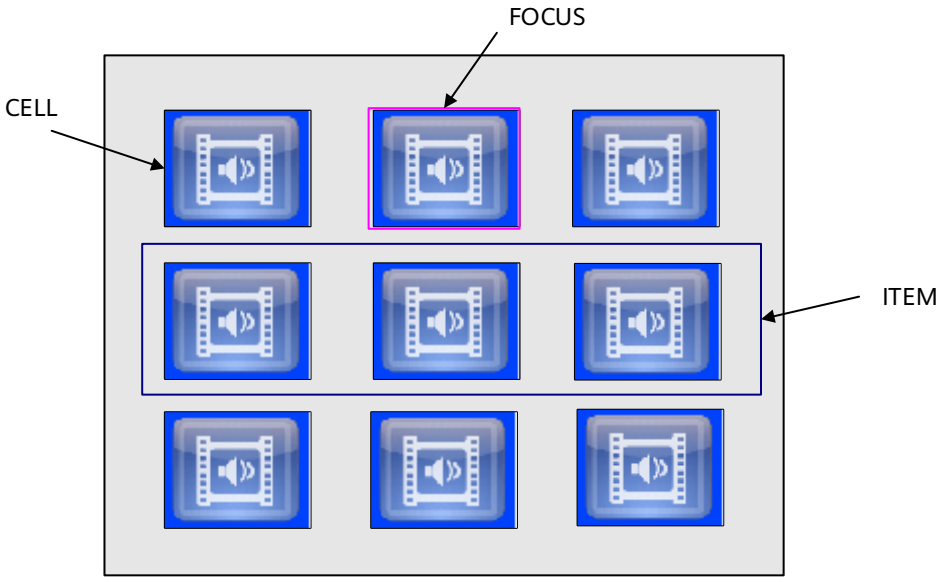
- 事件 onfocusmove 对应 HIGV_MSG_CLOCK_FOCUS_MOVE，在内部焦点移动时发生。
- 事件 ontimeadjust 对应 HIGV_MSG_CLOCK_TIME_ADJUST，在内部焦点的数据被修改时发生。
- 事件 ontimeupdate 对应 HIGV_MSG_CLOCK_UPDATE，定时器循环触发。

- 事件 onposition 对应 HIGV_MSG_POSITION，上报切换焦点后的位置信息。

4.11 ScrollGrid 控件

ScrollGrid 是具有 2D 滚动效果的网格控件，该控件的数据来源于数据模型，可以更加生动美观的显示排列数据，比较消耗资源。ScrollGrid 头文件为《hi_gv_scrollgrid.h》。该控件属于复杂控件，在使用前需要调用接口 HI_GV_ScrollGrid_RegisterWidget()注册控件。

图4-13 ScrollGrid 图示



ScrollGrid 的功能和列表框类似，它们的区别是滚动网格以 2D 动画的效果体现内部焦点的移动，因此 ScrollGrid 需要设置较多的属性来描述控件外观。

4.11.1 ScrollGrid 私有属性

- 网格页面：

  页显示行数 rownum 与页显示列数 colnumColNum，图 4-13 的行数与列数都为 3。

  - cellcolnum 决定网格列的总数目。
  - leftmargin、rightmargin、topmargin、btmmargin 左右上下四项边距约束数据内容的绘制区域。

  - 行间距 rowspace 与列间距 colspace 决定了单元格之间的距离。
- 内部焦点设置：
  - focusactiveskin 控件获得焦点时的焦点框皮肤。
  - focusnormskin 控件未获得焦点时焦点框皮肤。
- 网格列属性：
  - 列类型，图片或文字。
  - colindex 列索引，指定该列的索引。
  - coltop 内容相对于单元格的纵偏移。
  - colleft 内容相对于单元格的横偏移。
  - colwidth 该类型内容占据的列宽度。
  - colheight 该类型内容占据的列高度。
  - colalignment 列的对齐方式。
  - colbinddb 内容是否来源于 ADM。
  - colbindex 指定 ADM 的字段索引。
  - colinmage 数据不来源于索引且为图片类型时的图片。

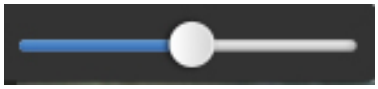
4.11.2 ScrollGrid 的独特事件

- 焦点框切换事件 onfocusmove，对应 HIGV_MSG_ITEM_SELECT。
- 单元格的属性列选中事件 oncellcolselect，对应 HIGV_MSG_SCROLLGRID_CELL_COLSEL。
- 数据状态改变事件 ondatachange，对应 HIGV_MSG_DATA_CHANGE。
- fling 手势滑动结束事件 onfinishfling，对应 HIGV_MSG_SCROLLGRID_FLING_FINISH。

4.12 TrackBar 控件

TrackBar 是通过移动滑块来设定相应数值的控件。TrackBar 的头文件为《hi_gv_trackbar.h》。TrackBar 属于使用较少的控件。

图4-14 TrackBar 图示



TrackBar 多用于鼠标操作，可以通过鼠标按下滑块后拖动改变刻度，也可以通过按键左右控制滑块移动。

4.12.1 TrackBar 私有属性

- Style：滑动风格，垂直或水平。
- 滑块：
  - Normaltrack：滑块普通状态图片资源。
  - Activetrack：滑块激活状态图片资源。
  - Mousedowntrack：滑块鼠标按下状态图片资源。
- 滑杆：
  - Slider：滑竿的背景图片资源。
  - Margin：滑竿的边距，滑竿的绘制区域。
- 滑动：
  - CurValue：当前刻度。
  - MaxValue：最大刻度。
  - MinValue：最小刻度。
  - Step：滑动步长。

4.12.2 TrackBar 的独特事件

刻度变化事件 onvaluechange，对应 HIGV_MSG_VALUEONCHANGE。

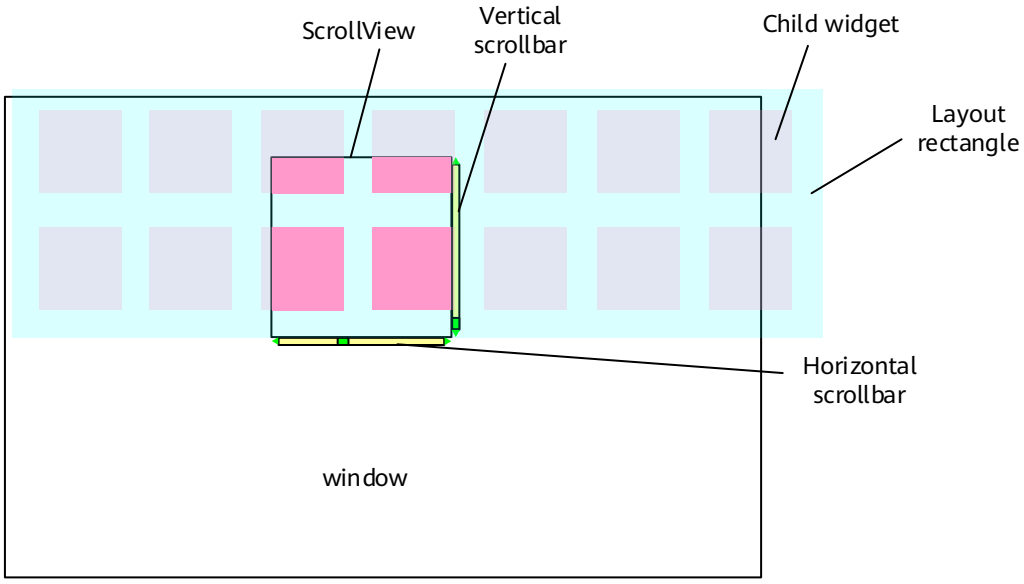
4.13 ScrollView 控件

ScrollView 是容器类控件，它可以放置除了窗口之外的其他控件。头文件为《hi_gv_scrollview.h》，其主要功能如下：

- 可排版布局的子控件总面积大于视口范围。
- 可绑定垂直滚动条。
- 可绑定水平滚动条。
- 支持鼠标滑轮控制垂直滚动条。
- 焦点切至未完全显示的子控件时自动移动视口。

- 视口移动事件通知。

图4-15 ScrollView 图示



4.13.1 ScrollView 私有属性

- 视口（可见区域）

  Margin 可以调节上、下、左、右四个边距来设置显示子控件的视口在 ScrollView 背景中的范围。

- 子控件的总范围：
  - ScrollContentWidth：子控件总范围的最大宽度，为零时不限制宽度。
  - ScrollContentHeight：子控件总范围的最大高度，为零时不限制高度。
- 滚动：
  - Step：视口滚动的步长，目前不支持平滑滚动，只刷新一次。
  - Interval：视口滚动的时间间隔，目前不支持平滑滚动，只刷新一次。
  - hVerticalScrollBar：绑定的垂直方向滚动条。
  - hHorizontalScrollbar：绑定的水平方向滚动条。
  - direction：控件内容滚动方向。

4.13.2 ScrollView 的独特事件

- ScrollView 视口移动事件 onviewmove，对应 HIGV_MSG_SCROLLVIEW_SCROLL。
- ScrollView fling 手势滑动结束事件 onfinishfling，对应 HIGV_MSG_SCROLLVIEW_FINISHFLING

4.14 SlideUnlock 控件

SlideUnlock 是通过移动滑块进行屏幕解锁的控件。可拖动滑块从起点移动到终点位置进行解锁，滑动过程中松开滑块，滑块会进行回弹操作，带有动画效果。

图4-16 SlideUnlock 图示

4.14.1 SlideUnlock 私有属性

- Style：滑动风格，垂直或水平。
- 滑块：
  - Normalthumb：滑块初始状态图片资源。
  - Touchthumb：滑块激活状态图片资源。
  - Donethumb：滑块完成状态图片资源。
- 滑杆：
  - Slider：滑竿的背景图片资源。
  - Margin：滑竿的边距，滑竿的绘制区域。

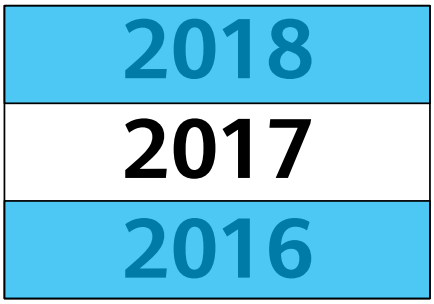
4.14.2 SlideUnlock 的独特事件

- 滑动解锁完成事件 onunlock，对应 HIGV_MSG_UNLOCK。
- 滑块移动事件 onmove，对应 HIGV_MSG_MOVE。
- 滑块回弹到原点事件 onkickback，对应 HIGV_MSG_KICKBACK。

4.15 WheelView 控件

WheelView 控件是滚动选择器，通过上下滑动操作来进行焦点切换，中间一行为焦点行，上下文本部分各覆盖半透明皮肤，实现滑动选择的效果。

图4-17 WheelView 图示



4.15.1 WheelView 私有属性

- Rownum：显示行数。
- Datamodel：数据模型句柄。
- Upcoverskin：上面覆盖文本皮肤句柄。
- Downcoverskin：下面覆盖文本皮肤句柄。
- Iscirclescroll：是否设置循环滚动。

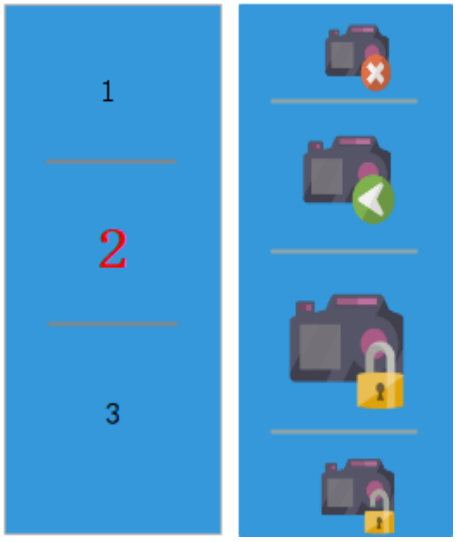
4.15.2 WheelView 的独特事件

滑动后焦点行上报事件 onfocusSelect，对应 HIGV_MSG_WHEELVIEW_ITEM。

4.16 ScaleView 控件

ScaleView 控件是滚动选择器，通过上下滑动操作改变中间行条目，点击条目来进行焦点切换并可自动搬移到中间行来使中间行为焦点行。每一行的条目内容的大小不同，中间一行为显示中间行。实现滑动改变条目内容大小以及点击条目进行条目内容变化同时伴随焦点切换的选择效果；该控件的数据来源于数据模型。

图4-18 ScaleView 图示



4.16.1 ScaleView 私有属性

- rowNum：显示行数。
- dataModel：数据模型句柄。
- hlineHeight：条目内容下水平线高度。
- hlineWidth：条目内容下水平线宽度。
- hlineColor：条目内容下水平线颜色。
- hlineDitance：条目内容下水平线到条目内容的距离。
- scaleFont：字体大小。
- tapTxtColor：点击选中字体条目的颜色。
- imgDecIndex：图片解码索引。
- type：条目内容形式：字体 0，图片 1。
- sizeGrain：字体大小放缩粒度。
- isNeedTransform：被点击图片条目后是否需要转换。注意替换的图片为被点击图片的上一级同名图片，需要保证有该资源。
- tapAutoMove：被点击条目后是否需要自动搬移。
- focusSelect：显示中间行行号上报事件索引。
- tapListenerIndex：Tap 监听，上报 tap 焦点行行号。
- imageSize：图片大小。

- imageSizeGrain：图片大小变化粒度。

4.16.2 ScaleView 的独特事件

- 滑动后显示中间行上报事件 onfocusselect，对应 MSG_SCALEVIEW_ITEM。
- 点击条目后焦点行上报事件 ontapaction，对应 MSG_SCALEVIEW_TAPLISTENER。

4.17 Edit 控件

Edit 用于编辑和显示简单字符，包括拼音、英文、数字及标点符号的编辑，不可编辑从右至左书写的复杂语言。编辑框的光标为竖光标，光标位置即为当前编辑字符的位置。Edit 的头文件为《hi_gv_edit.h》。

4.17.1 Edit 特殊风格

Eidt 除了最普通的编辑模式外，还支持替换风格、只读风格。

- 替换风格

  Eidt 可以设置替换风格，替换风格下所有输入的字符都显示为被替换字符。

  替换风格有自己独特的输入完成事件，对应 HIGV_MSG_EDIT_INPUTEND。
- 只读风格

  Edit 控件起来后，编辑框中预设置的文本内容只能读取，不支持编辑设置。

4.17.2 字符编码

目前 HiGV 字符编辑采用 UTF-8 字符编码，支持对应 1-4 个字节的字符编辑。常用的 ASCII 编码都是单字节字符，而中日韩文、阿拉伯文等复杂文字的一个字符对应 2 个以上字节。HiGV 的编辑是以字符为单位的。

为了方便操作编辑框内容，编辑框提供了设置、获取光标字节位置和字符位置的接口，详见《hi_gv_edit.h》。

4.18 ScrollText 控件

ScrollText 用于水平方向滚动显示图片和文本，ScrollText 头文件为《hi_gv_scrolltext.h》，其主要功能有：

- 支持一副图片和一段文字的滚动显示，也可以只有图片或文字。
- 设置图片及文字的相对位置。
- 左向右、右向左滚动。
- 可动态停止和重启滚动。
- 可设置滚动步长和滚动间隔。

ScrollText 默认只滚动一次，我们可以通过设置属性让文本一直处于滚动状态。图片和文字相对于控件左上角的距离都可以调用接口或在 xml 中直接设置。ScrollText 完成一次滚动后会产生滚动结束事件，对应 HIGV_MSG_ST_SCROLLONETIME。

滚动字幕注意事项：

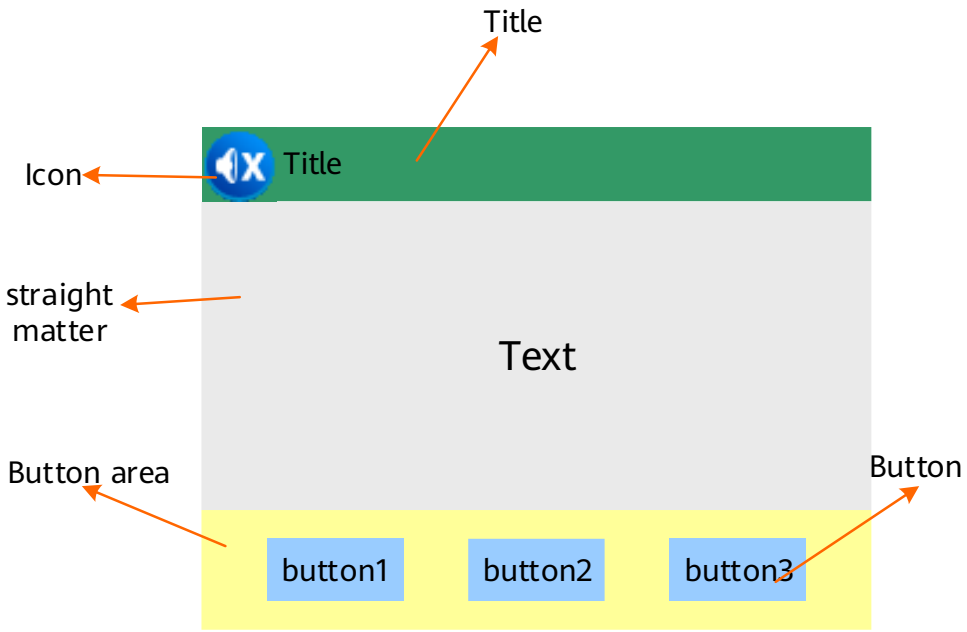
- 字符的长度有一定的限制，这和 HiGO 支持的 surface 最大长度有关。
- 滚动字幕的字体颜色也是由皮肤前景色控制，请注意将前景色与背景色设置为不同的值。
- 滚动的步长越小，间隔越短表现越细腻，但随之而来的是频繁刷新和搬移图片带来的性能影响；滚动步长过大，间隔过长可能会使滚动的视觉效果变差。我们应该根据视觉效果对滚动步长和间隔作适当微调。

4.19 MsgBox 控件

MsgBox 是弹出消息提醒用户或弹出选择窗口与用户交互选择信息的控件，MsgBox 的头文件为《hi_gv_msgbox.h》。其主要功能有：

- 标题栏设置。
- 显示提示正文。
- 可设置按钮选项。
- 自动隐藏。
- 可设置标题小图标。
- 选定按钮后回调注册事件。
- 提供阻塞式显示接口。

图4-19 MsgBox 图示



4.19.2 MsgBox 私有属性

- 窗口属性

  MsgBox 从 Window 继承而来，它具有窗口的所有属性。

- 风格：

  MsgBoxStyle 消息框风格分为七种，他表明弹出框拥有的按钮个数和按钮值：

  - MSGBOX_STYLE_NONE：无按钮提示框风格。
  - MSGBOX_STYLE_OK：单按钮风格，按钮 ID 为 MSG_BUTTONID_OK。
  - MSGBOX_STYLE_OKCANCEL：双按钮风格，按钮 ID 分别为：

    MSG_BUTTONID_OK

    MSG_BUTTONID_CANCEL
  - MSGBOX_STYLE_RETRYCANCEL：双按钮风格，按钮 ID 分别为：

    MSG_BUTTONID_RETRY

    MSG_BUTTONID_CANCEL
  - MSGBOX_STYLE_ABORTRETRYIGNORE：三按钮风格，按钮 ID 分别为：

    MSG_BUTTONID_ABORT

    MSG_BUTTONID_RETRY

    MSG_BUTTONID_IGNORE

  - MSGBOX_STYLE_YESNOCANCEL：三按钮风格，按钮 ID 分别为：

    MSG_BUTTONID_YES
    MSG_BUTTONID_NO
    MSG_BUTTONID_CANCEL
  - MSGBOX_STYLE_USERDEF：用户自定义风格，可以自定义按钮个数（1-4）和按钮 ID。
- 标题栏：
  - titleHeight：标题栏高度。
  - titleAlignment：标题栏文本对齐方式。
  - titleSkin：标题栏皮肤。
  - titleFont：标题栏文本字体。
  - titleText：标题栏文本。
  - icon：标题栏的小图标图片资源。
- 正文

  正文的文本、字体、对齐方式、背景皮肤等所有内容来源于控件本身。需要注意如果是用接口创建的 MsgBox，要在调用 HI_GV_MSGBOX_Init 初始化消息框之前设置正文的文本对齐方式。
- 按钮：
  - buttonAreaSkin：按钮区域的皮肤
  - buttonAreaHeight：按钮区域的高度。
  - buttonHeight：按钮高度。
  - buttonWidth：按钮宽度。
  - buttonNormalSkin：按钮普通状态皮肤。
  - buttonActiveSkin：按钮激活状态皮肤。
  - buttonHighlightSkin：鼠标移入按钮状态皮肤。
  - buttonMouseDownSkin：鼠标按下按钮皮肤。
  - buttonFont：按钮字体。
  - buttonCount：自定义风格下按钮的总数。
  - buttonID：按钮对应的 ID。
  - initFocusButton：MsgBox 显示时焦点按钮。
- 自动隐藏

showTimes 自动隐藏时间，单位为毫秒，显示后到达隐藏时间会自动隐藏。

4.19.3 MsgBox 的独特事件

- 按钮按下事件 onselectbutton，对应 HIGV_MSG_MSGBOX_SELECT。
- 自动隐藏事件 onautohide，对应 HIGV_MSG_MSGBOX_TIMEROUT。

4.19.4 设置 MsgBox 按钮

接口 HI_GV_MSGBOX_GetButtonHandle 可以获取到 MsgBox 的按钮句柄，有了按钮句柄可以对按钮进行操作，比如动态设置文本。

4.19.5 阻塞式显示 MsgBox

MsgBox 提供了阻塞式调用接口的功能，当 HI_GV_MSGBOX_Show 接口被调用时，，MsgBox 控件将接管消息分发引擎，，UI 只会响应在 MsgBox 内部的操作。当按下 MsgBox 某按钮触发隐藏，接口 HI_GV_MSGBOX_Show 返回这个按下按钮的 ID，并停止阻塞，将消息分发引擎再交给 Higv 主线程，进而继续执行接其他业务逻辑。

4.20 Charts 控件

Charts 是根据用户提供的相关数据，通过折线图、柱状图或两者兼有模式，向用户展示相关数据信息的控件，Charts 的头文件为《hi_gv_charts.h》。其主要功能有：

- 选配不同类型的图表：折线图、单柱状图、条目簇图。
- 标题显示可设置。
- 折线图、条目簇类型时，图例配置显示。
- X 轴刻度标签可替换文本。
- 条目簇类型，可同时显示折线与多簇柱状图。
- 可选择是否显示网格线。
- 可选择是否显示标准线。

4.20.1 Charts 私有属性

- 窗口属性

  Charts 从 Window 继承而来，它具有窗口的所有属性。

- 类型

  Charts 控件可用类型分为三种，并定义在 HigvChartsTypes 结构体中：

```
typedef enum {
    HIGV_CHARTS_TYPE_INVAILD = 0,
    HIGV_CHARTS_TYPE_LINE = 1,    /* line chart.[CN]:折线图 */
    HIGV_CHARTS_TYPE_BAR = 2,     /* histogram. [CN]:柱状图 */
    HIGV_CHARTS_TYPE_CLUSTER = 3,  /* histogram(cluster). [CN]:柱状条目簇图 */
    HIGV_CHARTS_TYPE_BUFFER
} HigvChartsTypes;
```

  注意：目前不支持折线图规格。

- 坐标系置属性

  通过设置 axisLeftSpace、axisRightSpace、axisTopSpace、axisBottonSpace 四个属性，即可将图表坐标系在当前 Window 窗口下的显示位置确定下来。

- 标题

  标题主要有：字体，颜色，内容三个属性。其显示位置与“坐标轴与边框的顶距离：axistopspace”数量有关。

- 图例：

  图例主要有：字体，颜色，内容三个属性。其显示位置与“坐标轴与边框的顶距离：axistopspace”数量有关，并位于标题之下。

  图例仅在折线图、条目簇图中显示。

- 坐标轴缩进：
  - indentationx：X 轴方向的缩进数值；

    indentationx 与 X 轴标签显示有关，X 轴显示标签的字体大小不应大于 indentationx 的数值；
  - indentationy：Y 轴方向的缩进数值。

- 坐标系刻度值：
  - axisYTickNum：Y 轴细分数值，即根据 Y 轴的长度，将其 axisYTickNum 等分；
  - axisXTickNum：X 轴细分数值，即根据 X 轴的长度，将其 axisXTickNum 等分；
  - axisXMinValue: X 轴刻度值最小值；
  - axisXMaxValue: X 轴刻度值最大值；
  - axisYMinValue: Y 轴刻度值最小值；
  - axisYMaxValue: Y 轴刻度值最大值；

- axisTickLength：刻度值长度；若选择网格线时，此属性无效。

- 数据绑定：

  在展示数据时，需先通过 API 接口（详见《HiGV API 参考》），将外部数据与图表控件相绑定。

4.21 MultiEdit 控件

MultiEdit 用于多行编辑和显示简单字符，包括拼音、英文、数字及标点符号的编辑，不可编辑从右至左书写的复杂语言。编辑框的光标为竖光标，光标位置即为当前编辑字符的位置。MultiEdit 的头文件为《hi_gv_multiedit.h》。

4.21.1 MultiEdit 边界

MultiEidt 有四个边界：左边界、右边界、上边界、下边界。光标位置在控件左边时向左滚，选择坐标即为控件屏幕坐标；光标位置在控件右边时向右滚，选择坐标为控件屏幕坐标加上控件宽度；鼠标滚动到边界后不再滚动；

4.21.2 字符编码

目前 HiGV 字符编辑采用 UTF-8 字符编码，支持对应 1-4 个字节的字符编辑。常用的 ASCII 编码都是单字节字符，而中日韩文、阿拉伯文等复杂文字的一个字符对应 2 个以上字节。HiGV 的编辑是以字符为单位的。

为了方便操作多行编辑框内容，多行编辑框提供了设置、获取光标字节位置和字符位置的接口，详见《hi_gv_multiedit.h》。

4.22 Cursor 鼠标控件

Cursor 控件用于在不支持触摸屏场景情况下，实现类似触摸屏的相关操作，包括鼠标按下，鼠标移动、鼠标中间滚轮移动等鼠标事件。cursor 的头文件为《hi_gv_cursor.h》。

4.22.1 Cursor 种类

Cursor 分为硬鼠标、软鼠标类型，主要的区别是：硬鼠标依赖芯片支持独立的物理鼠标图层，而软鼠标功能在仅支持单个图层的芯片上就可以使用。

4.22.2 鼠标相关属性

Cursor 主要属性有：

- 热点图标：鼠标显示时的图标；
- 显示范围：鼠标移动或显示的坐标范围。

4.22.3 参考代码

```
HI_S32 Mouse_Init()
{
    HIGV_HANDLE imageHandle;
    HigvCursorInfo cursorInfo = {0};
     HIGV_HANDLE redId= 0;
    (HI_VOID)HI_GV_CURSOR_Init(HIGV_CURSOR_SW);

    HI_S32 ret = HI_GV_Res_CreateID("./res/pic/mouse.png", HIGV_RESTYPE_IMG, &redId);
    if (ret != HI_SUCCESS) {
        printf("HI_GV_CURSOR_SetImage failed! Return: 0x%x\n", ret);
        return ret;
    }
    ret = HI_GV_Res_GetResInfo(redId, &imageHandle);
    if (ret != HI_SUCCESS) {
        printf("HI_GV_CURSOR_SetImage failed! Return: 0x%x\n", ret);
        return ret;
    }
    cursorInfo.imageHandle = imageHandle;

    (HI_VOID)HI_GV_CURSOR_SetCursorRegion(0, 0, SCREEN_HEIGHT, SCREEN_WIDTH);

    ret = HI_GV_CURSOR_SetImage(&cursorInfo);
    if (ret != HI_SUCCESS) {
        printf("HI_GV_CURSOR_SetImage failed! Return: 0x%x\n", ret);
        return ret;
    }
    (HI_VOID)HI_GV_CURSOR_Show();

    return HI_SUCCESS;
```



```
}
```


5 HiGV 编程技术

5.1 自定义绘制

HiGV 为用户提供了一套自定义绘制图形的接口，方便用户在已有控件的基础上定制化特殊的图形表现，这套接口包括绘制线条、填充背景、绘制文字、解码图片等功能。

5.1.1 创建绘制环境

说明
下文将创建临时绘制环境所依赖的控件称作**依赖控件**。

自定义绘制的原理是根据控件创建一个临时的绘制环境，通过调用 HiGV 封装的绘制接口，在控件的 surface 上完成绘制。

在开始自定义绘制之前，要创建临时绘制环境，接口 HI_GV_GraphicContext_Create(HI_HANDLE hWidget, HI_HANDLE *phGC)根据控件句柄得到一个临时的绘制环境句柄 phGC。

如果不会再使用临时绘制环境，应该调用接口 HI_GV_GraphicContext_Destroy(HI_HANDLE hGC)将临时绘制环境销毁，以免造成内存泄露。

5.1.2 自定义绘制接口

- HI_GV_GraphicContext_DecodeImg 和 HI_GV_GraphicContext_DecodeMemImg：解码图片接口。可以将指定文件或内存图片解码成 HiGV、HiGO 能够识别的图形 surface，解码图片并不依赖临时绘制环境，可以广泛使用于图形业务。

- HI_GV_GraphicContext_FreeImageSurface：释放解码的图片 surface 接口。解码的图片不再使用时应该释放，以免造成内存泄露。
- HI_GV_GraphicContext_SetFgColor：设置前景色接口。为临时绘制环境设置前景色，前景色作用于线条和文字，默认值为依赖控件的前景色。
- HI_GV_GraphicContext_SetBgColor：设置背景色接口。为临时绘制环境设置背景色，背景色作用于背景，默认值为依赖控件的背景色。
- HI_GV_GraphicContext_DrawLine：绘制线条接口。可以绘制**前景色**线条。
- HI_GV_GraphicContext_DrawImage：绘制图片接口。可以将图片资源 ID 或解码生成的图片 surface 搬移到临时绘制环境的 surface 上。
- HI_GV_GraphicContext_DrawText 和 HI_GV_GraphicContext_DrawTextByID：绘制文本接口。可以绘制字符串，字符串颜色为前景色。
- HI_GV_GraphicContext_SetFont：设置文字使用的字体。
- HI_GV_GraphicContext_FillRect：填充背景接口。为一块区域填充背景色。

5.1.3 自定义绘制流程

流程

自定义绘制流程如下：

步骤 1 调用接口 HI_GV_GraphicContext_Begin 开始绘制。

步骤 2 为绘制环境添加剪切矩形，剪切矩形决定了绘制范围。

步骤 3 调用绘制接口绘制图形。

步骤 4 调用接口 HI_GV_GraphicContext_End 结束绘制。

----结束

绘制时机

以控件为基础进行自定义绘制，需要考虑界面控件显示、隐藏状态、控件间的叠加关系，如果这些操作和管理都由应用层实现，那么自定义绘制会非常复杂。

自定义绘制的最佳时机是：控件内部已经设置好了控件的绘制状态和内部数据，开始进行控件本身的绘制前，或刚刚完成控件绘制还未清除绘制状态数据的时候。

前文的绘制章节介绍过 HIGV_MSG_PAINT 事件，该事件是控件绘制事件，可以调用接口 HI_GV_Widget_SetMsgProc 注册一个事件回调函数，在回调函数中完成自定义绘制。

- 如果回调函数处理优先级为 HIGV_PROCORDER_BEFORE，在控件绘制事件之前，回调函数可以影响到控件本身的绘制。
  - 返回值为 HIGV_PROC_GOON 时控件绘制会继续进行；
  - 返回值为 HIGV_PROC_STOP 则会中断控件本身绘制。

  此时的剪切矩形默认为控件的绘制区域，设置剪切矩形会改变控件的绘制区域。
- 如果回调函数处理优先级为 HIGV_PROCORDER_AFTER，在控件绘制事件之后，回调函数不会影响控件本身的绘制，此时的剪切矩形默认为控件的绘制区域。

在控件绘制完成之后进行自定义绘制更加安全准确，不需要设置剪切矩形，其绘制区域和控件的绘制区域相同。

5.1.4 参考代码

```
static HI_HANDLE s_ImgHandle = 0;
static HI_HANDLE s_WinGC = 0;

static HI_S32 TestPaintProc(HI_HANDLE hWidget, HI_U32 wParam, HI_U32 lParam)
{
    HI_S32 Ret;
    HI_RECT Rect = {50,150,207,100};

    HI_GV_GraphicContext_Begin(s_WinGC);
    HI_GV_GraphicContext_DrawImage(s_WinGC, &Rect, s_ImgHandle, 0, 0);
    HI_GV_GraphicContext_SetFgColor(s_WinGC, 0xFFFFF000);
    Rect.x += 120;
    HI_GV_GraphicContext_DrawText(s_WinGC, "Test graphiccontext.", &Rect, 0);

    HI_GV_GraphicContext_DrawLine(s_WinGC, 5, 5, 150, 5, 2);
    HI_GV_GraphicContext_DrawLine(s_WinGC, 5, 6, 150, 6, 2);
    HI_GV_GraphicContext_DrawLine(s_WinGC, 5, 7, 150, 7, 2);
    HI_GV_GraphicContext_End(s_WinGC);

    return HI_SUCCESS;
}
```



```
HI_S32 main(HI_S32 argc, HI_CHAR *argv[])
{
    HI_GV_Init();
    ...
    ...

    Ret = HI_GV_GraphicContext_Create(TestWin, &s_WinGC);
    if(HI_SUCCESS == Ret)
    {
        Ret = HI_GV_GraphicContext_DecodeImg("./res/test.png", 0,0, &s_ImgHandle);
        if(HI_SUCCESS != Ret)
        {
            HI_GV_GraphicContext_Destroy(s_WinGC);
        }
    }

    HI_GV_Widget_SetMsgProc(TestWin, HIGV_MSG_PAINT, TestPaintProc,
HIGV_PROCORDER_AFTER);
    ...
    HI_GV_App_Start(Gui_App);
    ...
    if(s_WinGC)
    {
        HI_GV_GraphicContext_Destroy(s_WinGC);
    }

    if(s_ImgHandle)
    {
        HI_GV_GraphicContext_FreeImageSurface(s_ImgHandle);
    }
    ...
    return HI_SUCCESS;
}
```


5.2 输入设备适配

HiGV 提供输入设备管理框架，各种输入设备可以集成到输入设备管理框架中。

5.2.1 适配输入事件

在 HiGV 中实现了输入设备的开发框架，用户将所有输入设备抽象成如下的接口，用户自定义的输入设备需要实现相应的接口，然后将定义的设备注册到系统，当系统运行时，会将用户定义的输入设备的消息发送给 GUI 系统。

```
/** Input devece structure*//** CNcomment:输入设备结构*/
typedef struct hiHIGV_INPUT_S
{
    HI_S32 (*InitInputDevice)(HI_VOID);
    HI_S32 (*DeinitInputDevice)(HI_VOID);
    /** timeout is us */
    HI_S32 (*GetKeyEvent)(HIGV_INPUTEVENT_S *pInputEvent, HI_U32 TimeOut);
    /** timeout is us */
HI_S32 (*GetMouseEvent)(HIGV_INPUTEVENT_S *pInputEvent, HI_U32 TimeOut);
    /** timeout is us */
    HI_S32 (*GetTouchEvent)(HIGV_TOUCH_GESTURE_EVENT_S* pInputEvent, HI_U32
TimeOut);
}HIGV_INPUT_S;
```

- InitInputDevice：指向用户实现的初始化输入设备接口。
- DeinitInputDevice：指向用户实现的去初始化输入设备接口。
- GetKeyEvent：指向用户实现的接收按键值事件接口。
- GetMouseEvent：指向用户实现的接收鼠标事件接口。
- GetTouchEvent：指向用户实现的接收触摸事件接口。

HiGV 会调用接口 HI_GV_InitInputSuite(HIGV_INPUT_S* pInputDevice)初始化设备管理，用户在应用层实现此接口，在接口中将 pInputDevice 的各项成员指向相应功能的函数。

5.2.2 触摸屏适配参考代码

```
#define    ONE_POINTER      0
#define    TWO_POINTER      1
```



```
#define    GESTURE_SCROLL_FIRST_TRIGGER    50
#define    GESTURE_SCROLL_TRIGGER                5
#define    GESTURE_FLING_TRIGGER                 500

static struct tsdev s_TsDevice;

static HI_BOOL s_AlwaysInTapReginon; //is always in down reginon

static HIGV_TOUCH_POINT_S   s_LatestDownEvent;
static HIGV_TOUCH_POINT_S   s_LatestMoveEvent;
static HIGV_TOUCH_POINT_S   s_LatestPointerEvent;
static HIGV_TOUCH_POINT_S   s_LatestTouchEvent;

static HI_S32 s_LastFocusX;
static HI_S32 s_LastFocusY;
static HI_S32 s_DownFocusX;
static HI_S32 s_DownFocusY;

static HI_S32 s_PointerMode;
static HI_BOOL s_PointerMove = HI_FALSE;

static HIGV_TOUCH_E GetTouchEventType(struct ts_sample* event)
{
    struct ts_sample currentInfo;
    HIGV_TOUCH_E type;

    /*save last one pointer pressure status*/
    static unsigned int lastOnePressure = 0;

    /*save last two pointer pressure status*/
    static unsigned int lastTwoPressure = 0;

    HIGV_MemSet(&currentInfo, 0x0, sizeof(struct ts_sample));
    HIGV_MemCopy(&currentInfo, event, sizeof(struct ts_sample));

    if (ONE_POINTER == currentInfo.id)
    {
        if (0 == lastOnePressure)
```



```
        {
            lastOnePressure = currentInfo.pressure;
            type = HIGV_TOUCH_START;
        }
        else if ( (1 == lastOnePressure) && (1 == currentInfo.pressure))
        {
            type = HIGV_TOUCH_MOVE;
        }
        else if ( (1 == lastOnePressure) && (0 == currentInfo.pressure))
        {
            lastOnePressure = currentInfo.pressure;
            type = HIGV_TOUCH_END;
        }
    }

    if (TWO_POINTER == currentInfo.id)
    {
        if (0 == lastTwoPressure)
        {
            lastTwoPressure = currentInfo.pressure;
            type = HIGV_TOUCH_POINTER_START;
        }
        else if ( (1 == lastTwoPressure) && (1 == currentInfo.pressure))
        {
            type = HIGV_TOUCH_MOVE;
        }
        else if ( (1 == lastTwoPressure) && (0 == currentInfo.pressure))
        {
            lastTwoPressure = currentInfo.pressure;
            type = HIGV_TOUCH_POINTER_END;
        }
    }

    return type;
}

static HI_S32 CalculateVelocity(HIGV_TOUCH_POINT_S* latestDownEvent,
HIGV_TOUCH_POINT_S* currEvent,
```



```
                                        HI_S32* velocityX, HI_S32* velocityY)
{
    HI_S32 timeInternal;

    if ((NULL == latestDownEvent) || (NULL == currEvent))
    {
        return HI_FAILURE;
    }

    HI_U32 latesttime = latestDownEvent->timeStamp;
    HI_U32 currtime = currEvent->timeStamp;

    timeInternal = currtime - latesttime;

    if (0 >= timeInternal)
    {
        return HI_FAILURE;
    }

    HI_S32 deltaX = abs(currEvent->x - latestDownEvent->x);
    HI_S32 deltaY = abs(currEvent->y - latestDownEvent->y);

    *velocityX = (deltaX * 1000) / timeInternal;
    *velocityY = (deltaY * 1000) / timeInternal;

    return HI_SUCCESS;
}

static HI_S32 CalculateDistance(HIGV_TOUCH_POINT_S* Event1, HIGV_TOUCH_POINT_S*
Event2,
                                        HI_S32* intervalX, HI_S32* intervalY)
{
    if ((NULL == Event1) || (NULL == Event2))
    {
        return HI_FAILURE;
    }

    *intervalX = abs(Event1->x - Event2->x);
```



```
    *intervalY = abs(Event1->y - Event2->y);

    return HI_SUCCESS;
}

static HI_S32 GestureDetector(HIGV_TOUCH_POINT_S* event,
HIGV_TOUCH_GESTURE_EVENT_S* callbackevent)
{
    int ret;
    HIGV_TOUCH_POINT_S currentEvent;
    HI_S32 focusX, focusY;
    HIGV_TOUCH_GESTURE_EVENT_S touch_gesture_event;
    HIGV_TOUCH_E type;

    if (NULL == event)
    {
        return HI_FAILURE;
    }

    HIGV_MemSet(&currentEvent, 0x0, sizeof(currentEvent));
    HIGV_MemCopy(&currentEvent, event, sizeof(currentEvent));

    HIGV_MemSet(&touch_gesture_event, 0x0, sizeof(HIGV_TOUCH_GESTURE_EVENT_S));

    focusX = currentEvent.x;
    focusY = currentEvent.y;
    type = currentEvent.type;

    switch (type)
    {
        case HIGV_TOUCH_POINTER_START:
        {
            s_PointerMode += 1;
            HIGV_MemCopy(&s_LatestPointerEvent, &currentEvent, sizeof(currentEvent));
            break;
        }

        case HIGV_TOUCH_POINTER_END:
```



```
        {
            s_PointerMode -= 1;
            break;
        }

        case HIGV_TOUCH_START:
        {
            s_DownFocusX = s_LastFocusX = focusX;
            s_DownFocusY = s_LastFocusY = focusY;

            s_AlwaysInTapReginon = HI_TRUE;
            s_PointerMode = 1;
            HIGV_MemCopy(&s_LatestDownEvent, &currentEvent, sizeof(currentEvent));
            break;
        }

        case HIGV_TOUCH_MOVE:
        {
            HI_S32 scrollX = s_LastFocusX - focusX;
            HI_S32 scrollY = s_LastFocusY - focusY;

            if (s_PointerMode >= 2)
            {
                HI_S32 intervalX = 0;
                HI_S32 intervalY = 0;

                /**handler pinch gesture*/
                if (TWO_POINTER == currentEvent.id)
                {
                    ret = CalculateDistance(&s_LatestTouchEvent, &currentEvent, &intervalX,
&intervalY);
                    HIGV_MemCopy(&s_LatestPointerEvent, &currentEvent,
sizeof(currentEvent));
                    s_PointerMove = HI_TRUE;
                    touch_gesture_event.gestureevent.gesture.pinch.pointer1 =
s_LatestTouchEvent;
                    touch_gesture_event.gestureevent.gesture.pinch.pointer2 = currentEvent;
                }
```



```
                else if (ONE_POINTER == currentEvent.id)
                {
                    ret = CalculateDistance(&s_LatestPointerEvent, &currentEvent, &intervalX,
&intervalY);
                    s_PointerMove = HI_FALSE;
                    touch_gesture_event.gestureevent.gesture.pinch.pointer1 = currentEvent;
                    touch_gesture_event.gestureevent.gesture.pinch.pointer2 =
s_LatestPointerEvent;
                }

                if (HI_SUCCESS != ret)
                {
                    HIGV_Printf("[Func: %s, Line: %d]\n", __FUNCTION__, __LINE__);
                    return HI_FAILURE;
                }

                touch_gesture_event.isgesture = HI_TRUE;
                touch_gesture_event.gesturetype = HIGV_GESTURE_PINCH;
                touch_gesture_event.gestureevent.gesture.pinch.intervalX = intervalX;
                touch_gesture_event.gestureevent.gesture.pinch.intervalY = intervalY;
            }

            if (HI_TRUE == s_AlwaysInTapReginon)
            {
                HI_S32 deltaX = focusX - s_DownFocusX;
                HI_S32 deltaY = focusY - s_DownFocusY;

                HI_S32 distance = (deltaX * deltaX) + (deltaY * deltaY);

                /**handler scroll gesture*/
                if (GESTURE_SCROLL_FIRST_TRIGGER < distance)
                {
                    s_LastFocusX = focusX;
                    s_LastFocusY = focusY;
                    s_AlwaysInTapReginon = HI_FALSE;

                    HIGV_MemCopy(&s_LatestMoveEvent, event, sizeof(struct ts_sample));
```



```
                touch_gesture_event.isgesture = HI_TRUE;
                touch_gesture_event.gesturetype = HIGV_GESTURE_SCROLL;
                touch_gesture_event.gestureevent.gesture.scroll.start = s_LatestDownEvent;
                touch_gesture_event.gestureevent.gesture.scroll.end = currentEvent;
                touch_gesture_event.gestureevent.gesture.scroll.distanceX = abs(scrollX);
                touch_gesture_event.gestureevent.gesture.scroll.distanceY = abs(scrollY);
            }
        }
        else if (abs(scrollX) > GESTURE_SCROLL_TRIGGER || abs(scrollY) >
GESTURE_SCROLL_TRIGGER)
        {
            s_LastFocusX = focusX;
            s_LastFocusY = focusY;

            touch_gesture_event.isgesture = HI_TRUE;
            touch_gesture_event.gesturetype = HIGV_GESTURE_SCROLL;
            touch_gesture_event.gestureevent.gesture.scroll.start = s_LatestMoveEvent;
            touch_gesture_event.gestureevent.gesture.scroll.end = currentEvent;
            touch_gesture_event.gestureevent.gesture.scroll.distanceX = abs(scrollX);
            touch_gesture_event.gestureevent.gesture.scroll.distanceY = abs(scrollY);

            HIGV_MemCopy(&s_LatestMoveEvent, &currentEvent, sizeof(currentEvent));
        }

        break;
    }

    case HIGV_TOUCH_END:
    {
        s_PointerMode = 0;

        /**handler tap gesture*/
        if (HI_TRUE == s_AlwaysInTapReginon)
        {
            touch_gesture_event.isgesture = HI_TRUE;
            touch_gesture_event.gesturetype = HIGV_GESTURE_TAP;
            touch_gesture_event.gestureevent.gesture.tap.pointer = currentEvent;
        }
```



```
            else
            {
                HI_S32 velocityX = 0;
                HI_S32 velocityY = 0;
                ret = CalculateVelocity(&s_LatestDownEvent, &currentEvent, &velocityX,
&velocityY);

                /**handler fling gesture*/
                if (GESTURE_FLING_TRIGGER < velocityX || GESTURE_FLING_TRIGGER <
velocityY)
                {
                    touch_gesture_event.isgesture = HI_TRUE;
                    touch_gesture_event.gesturetype = HIGV_GESTURE_FLING;
                    touch_gesture_event.gestureevent.gesture.fling.start = s_LatestDownEvent;
                    touch_gesture_event.gestureevent.gesture.fling.end   = currentEvent;
                    touch_gesture_event.gestureevent.gesture.fling.velocityX = velocityX;
                    touch_gesture_event.gestureevent.gesture.fling.velocityY = velocityY;
                }
            }

            break;
        }

    }

    if (HI_FALSE == s_PointerMove)
    {
        HIGV_MemCopy(&s_LatestTouchEvent, &currentEvent, sizeof(currentEvent));
    }

    touch_gesture_event.touchtype = type;
    touch_gesture_event.touchevent.last = currentEvent;

    *callbackevent = touch_gesture_event;

    return HI_SUCCESS;
}
```



```
HI_S32 GetTouchEvent(HIGV_TOUCH_GESTURE_EVENT_S* pTouchEvent, HI_U32 TimeOut)
{
    HI_S32 ret = HI_FAILURE;
    struct ts_sample samp = {0};
    fd_set readset;
    struct timeval timeout = {0};
    HIGV_TOUCH_GESTURE_EVENT_S currentEvent;
    HIGV_TOUCH_POINT_S pointEvent;

    HIGV_MemSet(&samp, 0x0, sizeof(struct ts_sample));
    HIGV_MemSet(&currentEvent, 0x0, sizeof(HIGV_TOUCH_GESTURE_EVENT_S));
    HIGV_MemSet(&pointEvent, 0x0, sizeof(HIGV_TOUCH_POINT_S));

    if (s_TsDevice.fd <= 0)
    {
        usleep(300 * 1000);
        return OpenTouchScreenDevice();
    }

    timeout.tv_sec = 0;
    timeout.tv_usec = TimeOut * 1000;

    FD_ZERO(&readset);
    FD_SET(s_TsDevice.fd, &readset);

    ret = select(s_TsDevice.fd + 1, &readset, 0, 0, &timeout);

    if (ret == 0)
    {
        ret = HI_EEMPTY;
        return ret;
    }
    else if (ret < 0)
    {
        HIGV_Printf("[Func: %s, Line: %d]   ret: %d\n", __FUNCTION__, __LINE__, ret);
        ret = HI_ERR_COMM_UNKNOWN;
        ts_close(&s_TsDevice);
        return ret;
```



```
}

if (FD_ISSET(s_TsDevice.fd, &readset))
{

    ret = ts_read(&s_TsDevice, &samp, 1);

    if (ret < 0)
    {
        HIGV_Printf("[Func: %s, Line: %d] \n", __FUNCTION__, __LINE__);
        return HI_FAILURE;
    }

    if ((0 == samp.tv.tv_sec) && (0 == samp.tv.tv_usec))
    {
        ret = HI_EEMPTY;
        return ret;
    }

    HIGV_TOUCH_E type = GetTouchEventType(&samp);

    pointEvent.id = samp.id;
    pointEvent.x = samp.x;
    pointEvent.y = samp.y;
    pointEvent.pressure = samp.pressure;
    pointEvent.type = type;
    pointEvent.timeStamp = (samp.tv.tv_sec) * 1000 + (samp.tv.tv_usec) / 1000;

    ret = GestureDetector(&pointEvent, &currentEvent);

    if (-1 == ret)
    {
        HIGV_Printf("[Func: %s, Line: %d] \n", __FUNCTION__, __LINE__);
        return ret;
    }

    *pTouchEvent = currentEvent;
```



```
            return HI_SUCCESS;
        }

        return ret;
}
```

5.3 多图层

HiGV 支持在多图层上建立系统窗口，HIGV 多图层主要支持以下特性：

- 支持基于图层来创建窗口。
- 支持输入法窗口在图层间自动移动。
- 支持设置窗口的缺省图层。
- 支持查询窗口所在图层。
- 支持图层创建与销毁。
- 支持显示/隐藏图层。
- 支持获取图层的活动窗口功能。
- 支持设置图层同源。

5.3.1 开发应用

开发应用步骤如下：

步骤 1 调用函数 HI_GV_Layer_CreateEx 创建图层，该函数需要传入初始化参数。

步骤 2 调用图层操作函数进行操作。可以对图层进行显示/隐藏，设置/获取默认图层，为窗口指定所属图层。

步骤 3 调用函数 HI_GV_Layer_Destroy 销毁图层。

----结束

5.3.2 图层创建参数解析

```
typedef struct
{
    HI_S32      ScreenWidth;
    HI_S32      ScreenHeight;
```



```
    HI_S32      CanvasWidth;
    HI_S32      CanvasHeight;
    HI_S32      DisplayWidth;
    HI_S32      DisplayHeight;
    HIGO_LAYER_FLUSHTYPE_E      LayerFlushType;
    HIGO_LAYER_DEFLICKER_E      AntiLevel;
    HIGO_PF_E       PixelFormat;
    HIGO_LAYER_E       LayerID;
} HIGO_LAYER_INFO_S;
```

- ScreenWidth、ScreenHeight：最终显示屏幕的宽高。
- CanvasWidth、CanvasHeight：图形绘制区域宽高，即创建的全屏窗口的大小。
- DisplayWidth、DisplayHeight：图形映射区域宽高，即全屏窗口映射到屏幕时的大小。
- LayerFlushType：图层的刷新方式，包括单缓冲、双缓冲、三缓冲刷新等方式。
- AntiLevel：抗闪烁级别，值为 LOW~HIGH,值越大抗闪烁效果越好，但显示越模糊。
- PixelFormat：图层的像素格式。
- LayerID：图层硬件 ID，能支持图层取决于芯片平台。

5.3.3 参考代码

```
#define SCREEN_WIDTH    1280
#define SCREEN_HEIGHT 720

HI_S32 main(HI_S32 argc, HI_CHAR *argv[])
{
    HI_S32 Ret;
    HI_S32 hApp;
    HI_S32 Num;
    HI_HANDLE hDDB = INVALID_HANDLE;
    HI_HANDLE Layer;
    HIGO_LAYER_INFO_S LayerInfo = {SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_WIDTH,
SCREEN_HEIGHT, SCREEN_WIDTH, SCREEN_HEIGHT,
                                   (HIGO_LAYER_FLUSHTYPE_E)(HIGO_LAYER_FLUSH_FLIP),
                                   HIGO_LAYER_DEFLICKER_AUTO,
                                   HIGO_PF_8888, HIGO_LAYER_HD_0};
```



```
    HIGV_WINCREATE_S WinCreate;

    if (argc < 2)
    {
        printf("Please input higvbin file.\n");
        return -1;
    }

    Ret = HI_GV_Init();
    if (HI_SUCCESS != Ret)
    {
        printf("HI_GV_Init fail Ret:%x\n", Ret);
        return -1;
    }
    Ret = HI_GV_PARSER_Init();
    if (HI_SUCCESS != Ret)
    {
        printf("HI_GV_PARSER_Init fail Ret:%x\n", Ret);
        return -1;
    }


    (HI_VOID)HI_GV_Font_SetSystemDefault(hFont);
    s_hLanTestFont = hFont;

    Ret = HI_GV_App_Create("Step2MainApp", (HI_HANDLE *)&hApp);
    if (HI_SUCCESS != Ret)
    {
        HI_GV_Deinit();
        HI_GV_PARSER_Deinit();
        return NULL;
    }
    HI_GV_PARSER_SetWidgetEventFunc(g_pfunHIGVAppEventFunc,
sizeof(g_pfunHIGVAppEventFunc)/sizeof(HI_CHAR *));
    printf("step6\n");

    Ret = HI_GV_PARSER_LoadFile(argv[1]);
    if (HI_SUCCESS != Ret)
```



```
{
    printf("HI_GV_PARSER_LoadFile fail Ret:%x\n", Ret);
    HI_GV_Deinit();
    HI_GV_PARSER_Deinit();
    return -1;
}
Ret = HI_GV_Layer_CreateEx(&LayerInfo, LAYER_0);
if (HI_SUCCESS != Ret)
{
    printf("failed to HI_GV_Layer_CreateEx! ret:0x%x\n", Ret);
    return Ret;
}

Ret = HI_GV_PARSER_LoadViewById(allwidget);
if (HI_SUCCESS != Ret)
{
    printf("HI_GV_PARSER_LoadViewById fail Ret:%x\n", Ret);
    HI_GV_Deinit();
    HI_GV_PARSER_Deinit();
    return -1;
}

HI_GV_Widget_Show(allwidget);
HI_GV_Widget_Active(allwidget);

Ret = HI_GV_App_Start(hApp);
if (HI_SUCCESS != Ret)
{
    HI_GV_PARSER_Deinit();
    (HI_VOID)HI_GV_App_Destroy(hApp);
    HI_GV_Deinit();
    return NULL;
}

HI_GV_PARSER_Deinit();
(HI_VOID)HI_GV_App_Destroy(hApp);
HI_GV_Deinit();
```



```
    return 0;
}
```

5.4 MXML

MXML（Multi XML）是满足图形层上的窗口内，控件在不同输出设备进行独立控制的应用场景。该功能主要是通过一套 XML 界面描述，在不同输出设备上克隆出相同的界面，且能对不同的输出设备上的界面进行独立操作。

为了实现相同 XML 界面描述在不同输出设备进行独立操作，需要对指定图层进行克隆相应的控件，其中包括克隆控件的共享事件操作函数和相同 DB。由于所有克隆控件共享事件处理函数，因此需要用户自己来区分克隆控件的操作状态，为了更方便用户操作克隆控件，可将相应克隆的操作状态保存到控件的私有数据中。

5.4.1 开发应用

开发应用步骤如下：

步骤 1 设置 LAYER 全局图层方法，通过 xml2bin 选项进行配置界面 XML 中窗口所在的图层。

如：xml2bin –L "LAYER_0；LAYER_1；LAYER_2"。表示有三个图层，它们分别为 LAYER_0，LAYER_1，LAYER_2。

步骤 2 在 XML 窗口的属性 layer 制定图层。

步骤 3 初始化时，创建图层并显示相应图层上的窗口。

步骤 4 在应用程序中，通过查询并获取相应的图层的控件进行操作。

----结束

调用函数 HI_GV_Widget_GetPrivate 可以获取到克隆控件的私有属性，调用函数 HI_GV_Widget_GetLayer 来查询当前克隆控件的图层信息，根据图层信息就可以查询其他关联在该图层的克隆控件。

5.4.2 参考代码

```
HI_VOID main()
{
    HI_S32 Ret;
    HI_S32 Num;
```



```
    HI_HANDLE Layer;
    HIGV_WINCREATE_S WinCreate;
    HIGO_LAYER_INFO_S LayerInfo1= {SCREEN_WIDTH, SCREEN_HEIGHT,
                                   SCREEN_WIDTH,SCREEN_HEIGHT,
                                   SCREEN_WIDTH, SCREEN_HEIGHT,
(HIGO_LAYER_FLUSHTYPE_E)(HIGO_LAYER_FLUSH_DOUBBUFER),
                                  HIGO_LAYER_DEFLICKER_AUTO,
                                  HIGO_PF_1555, HIGO_LAYER_SD_0};

HIGO_LAYER_INFO_S LayerInfo2 = {SCREEN_WIDTHHD, SCREEN_HEIGHTHD,
SCREEN_WIDTHHD, SCREEN_HEIGHTHD, SCREEN_WIDTHHD, SCREEN_HEIGHTHD,
(HIGO_LAYER_FLUSHTYPE_E)(HIGO_LAYER_FLUSH_DOUBBUFER),
                                  HIGO_LAYER_DEFLICKER_AUTO,
                                  HIGO_PF_1555, HIGO_LAYER_HD_0};

    Ret = HI_GV_Init();
    assert (Ret == HI_SUCCESS);
    Ret = HI_GV_PARSER_Init();
    assert(Ret == HI_SUCCESS);
    Ret = AppCreateSysFont(&hFont);
    assert(Ret == HI_SUCCESS);

    Ret = HI_GV_App_Create("Step2MainApp", (HI_HANDLE *)&g_hMXMLApp);
    assert(Ret == HI_SUCCESS);

    Num = sizeof(g_pfunHIGVAppEventFunc);
    HI_GV_PARSER_SetWidgetEventFunc(g_pfunHIGVAppEventFunc,
sizeof(g_pfunHIGVAppEventFunc)/sizeof(HI_CHAR *));

    Ret = HI_GV_PARSER_LoadFile("./higv.bin");
    assert(Ret == HI_SUCCESS);

    Ret = HI_GV_Layer_CreateEx(&LayerInfo, LAYER_0);
    assert(Ret == HI_SUCCESS);

    Ret = HI_GV_Layer_CreateEx(&LayerInfo2, LAYER_1);
    assert(Ret == HI_SUCCESS);
...
```



```
#if   SHOW_CLONE_LAYER_WIN
      HI_HANDLE hCloneWin = 0, hCloneWidget = 0;
      HI_GV_Widget_GetCloneHandle(allwidget, LAYER_1, &hCloneWin);

      Ret = HI_GV_Widget_Show(hCloneWin);
      assert(Ret == HI_SUCCESS);
#endif

      Ret = HI_GV_Widget_Show(allwidget);
     assert(Ret == HI_SUCCESS);
      Ret = HI_GV_Widget_Active(allwidget);
     assert(Ret == HI_SUCCESS);

...
}
```

5.5 CLUT8 图层格式

HiGV 可以设置图片解码输出格式设置为 CLUT8 格式，CLUT8 格式需要设置调试板，为了减少内存，整个系统的调色板使用一个公共的调色板，包括图片绘制，图层显示等。

5.5.1 开发应用

开发应用步骤如下：

步骤 1 创建图层并设置像素格式。

步骤 2 设置解码图层信息.

步骤 3 创建控件。

----结束

5.5.2 注意事项

通过设置图层的像素格式和公共调色板就可以轻松使用 CLUT8，但需要注意以下限制：

- 不支持半透明窗口。
- 所有的图片和元素使用相同的 CLUT8 像素格式。
- 不支持界面中有任何缩放的功能。
- 在用户界面设计开始的时候就定下调色板的格式。包括图片元素颜色，透明色等。

5.5.3 参考代码

```
    HI_S32 Ret;
    HIGV_WCREATE_S WinInfo = {HIGV_WIDGET_WINDOW, {80, 50, 600, 450},
INVALID_HANDLE, 0, 0};
    HI_HANDLE hWin0, hWidget = INVALID_HANDLE;
    HI_HANDLE hWinSkin;
    HIGV_WCREATE_S info;
    HIGV_DEC_SUFINFO_S DecSufinfo;
    HIGV_WINCREATE_S WinCreate;
    HIGV_STYLE_S WinStyle;
    HI_HANDLE hSkin;

    /* set layer info */
    HIGO_LAYER_INFO_S LayerInfo= {SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_WIDTH,
SCREEN_HEIGHT, SCREEN_WIDTH, SCREEN_HEIGHT,

(HIGO_LAYER_FLUSHTYPE_E)(HIGO_LAYER_FLUSH_DOUBBUFER),
                                    HIGO_LAYER_DEFLICKER_NONE,
                                    HIGO_PF_CLUT8, HIGO_LAYER_HD_0};

    HI_GV_Deinit();

    Ret = HI_GV_Init();
    assert(Ret == HI_SUCCESS);


   /* create layer */
    Ret = HI_GV_Layer_Create(&LayerInfo, &g_StLayer);
    if (HI_SUCCESS != Ret)
    {
```



```
        printf("failed to HI_GV_Layer_Create! ret:0x%x\n", Ret);
        assert(0);
    }

    Ret = HI_GV_App_Create("test_drawclip", &s_hDrawClipApp);
    assert(Ret == HI_SUCCESS);

    Ret = HI_GO_SetWindowMode(HIGO_WNDMEM_SHARED);
    assert(Ret == HI_SUCCESS);

     /* set decode surface pixel format is clut8 */
    DecSufinfo.PixFormat = HIGO_PF_CLUT8;
    DecSufinfo.MemType = HIGO_MEMTYPE_MMZ;
    DecSufinfo.IsPubPalette = HI_TRUE;
    Ret = HI_GV_Resm_SetDecSurfInfo(&DecSufinfo);
    assert(Ret == HI_SUCCESS);

     /* create window */
    WinCreate.hLayer = g_StLayer;
    WinCreate.PixelFormat = HIGO_PF_BUTT;
    WinInfo.pPrivate = &WinCreate;
    Ret = HI_GV_Widget_Create(&WinInfo, &hWin0);
    assert(Ret == HI_SUCCESS);

    memset(&WinStyle, 0x0, sizeof(WinStyle));
    WinStyle.StyleType = HIGV_STYLETYPE_COLOR;
    WinStyle.BackGround.Color = 0xff;
    WinStyle.FontColor = 0x00;
    Ret = HI_GV_Res_CreateStyle(&WinStyle,&hSkin);
    Ret = HI_GV_Widget_SetSkin(hWin0, HIGV_SKIN_NORMAL, hSkin);
  assert(Ret == HI_SUCCESS);

    memset(&info, 0x0, sizeof(info));
    info.rect.x = 50;
    info.rect.y = 50;
    info.rect.w = 260;
    info.rect.h = 260;
    info.hParent = hWin0;
```



```
    info.type = HIGV_WIDGET_IMAGE;

    Ret = HI_GV_Widget_Create(&info,  &hWidget);
    assert(Ret == HI_SUCCESS);

     /* set gif on image */
    HI_RESID ResID;
    Ret = HI_GV_Res_CreateID("./res/test.gif", HIGV_RESTYPE_IMG, &ResID);
    assert(Ret == HI_SUCCESS);
    HI_GV_Image_SetImage(hWidget, ResID);

    memset(&WinStyle, 0x0, sizeof(WinStyle));
    WinStyle.StyleType = HIGV_STYLETYPE_COLOR;
    WinStyle.BackGround.Color = 0x55;
    Ret = HI_GV_Res_CreateStyle(&WinStyle,&hSkin);
    Ret = HI_GV_Widget_SetSkin(hWidget, HIGV_SKIN_NORMAL, hSkin);

    s_QuitHandle = hWin0;
    Ret = HI_GV_Widget_SetMsgProc(hWin0, HIGV_MSG_KEYDOWN, DrawClip_TestQuit,
HIGV_PROCORDER_AFTER);

    Ret = HI_GV_Widget_Show(hWin0);
    HI_GV_Widget_Active(hWin0);

    Ret = HI_GV_App_Start(s_hDrawClipApp);

    HI_GV_App_Destroy(s_hDrawClipApp);
    HI_GV_Deinit();
```

5.6 RLE 格式使用

RLE（run-length encoding）游程编码，是一种无损的图片编码格式。该格式不仅可以节约内存空间，而且又是一种无损失的压缩方案，不损失任何图像数据。

RLE 作为一种图片无损压缩的 clut8 格式，RLE 格式的图片可以通过将 gif 格式图片转换而来。在使用这种格式时，需要将 RLE 资源格式文件和调色板生成出来。

RLE 格式图片是由 GIF 图片通过 RLE 工具生成，同时也可以使用该工具提取调色板。

5.6.1 开发应用

RLE 开发应用步骤如下：

步骤 1 通过 RLE 工具生成调色板 pal 文件和资源文件 rle 文件。

步骤 2 创建图层和调色板。

步骤 3 设置解码图层信息。

步骤 4 加载视图，显示窗口。

----结束

5.6.2 参考代码

```
HI_S32 main(HI_S32 argc, HI_CHAR *argv[])
{
    HI_S32 Ret, hApp;
    HI_HANDLE hFont;
    HIGV_DEC_SUFINFO_S DecSufinfo;

    HIGO_LAYER_INFO_S LayerInfo = {SCREEN_WIDTH,
                                   SCREEN_HEIGHT,
                                   SCREEN_WIDTH,
                                   SCREEN_HEIGHT,
                                   SCREEN_WIDTH,
                                   SCREEN_HEIGHT,
                          (HIGO_LAYER_FLUSHTYPE_E)(HIGO_LAYER_FLUSH_FLIP),
                                   HIGO_LAYER_DEFLICKER_NONE,
                                   HIGO_PF_CLUT8,
                                   HIGO_LAYER_HD_0};

    /* check args */
    if (argc < 2)
    {
        printf("Please input higvbin file.\n");
        return -1;
    }
```



```
/* init higv module */
Ret = HI_GV_Init();
if (HI_SUCCESS != Ret)
{
    printf("HI_GV_Init fail Ret:%x\n", Ret);
    return -1;
}

Ret = HI_GV_PARSER_Init();
if (HI_SUCCESS != Ret)
{
    printf("HI_GV_Init fail Ret:%x\n", Ret);
    return -1;
}

/* set window memory mode */
Ret = HI_GO_SetWindowMode(HIGO_WNDMEM_SHARED);
assert(Ret == HI_SUCCESS);

/* load higv.bin file for parser */
Ret = HI_GV_PARSER_LoadFile(argv[1]);
if (HI_SUCCESS != Ret)
{
    printf("HI_GV_PARSER_LoadFile fail Ret:%x\n", Ret);
    HI_GV_Deinit();
    HI_GV_PARSER_Deinit();
    return -1;
}

 HI_HANDLE HD_0 = HI_NULL;
/* create layer */
Ret = HI_GV_Layer_Create(&LayerInfo, &HD_0);
if (HI_SUCCESS != Ret)
{
    printf("failed to HI_GV_Layer_CreateEx! ret:0x%x\n", Ret);
    return Ret;
}
```



```
/* set public palette, once clut to rgb */
Ret = HI_GO_SetPubPalette(gs_Palette);
assert(Ret == HI_SUCCESS);

/* set layer palette */
HI_HANDLE    HigoLayer;
HI_PALETTE Palette={0};

Ret = RLE_CreatePalette("./res/pub.pal", Palette);
assert(Ret == HI_SUCCESS);

HI_GV_Layer_GetHigoLayer(HD_0, &HigoLayer);
Ret = HI_GO_SetLayerPalette(HigoLayer, Palette);
assert(Ret == HI_SUCCESS);

/* set decode surface info in resource manager */
DecSufinfo.PixFormat = HIGO_PF_CLUT8;
DecSufinfo.MemType = HIGO_MEMTYPE_OS;
/* clut to rgb active using public palatte, after decode */
DecSufinfo.IsPubPalette = HI_TRUE;
Ret = HI_GV_Resm_SetDecSurfInfo(&DecSufinfo);
assert(Ret == HI_SUCCESS);

Ret = HI_GV_PARSER_LoadViewById(allwidget);
if (HI_SUCCESS != Ret)
{
    printf("HI_GV_PARSER_LoadViewById fail Ret:%x\n", Ret);
    HI_GV_Deinit();
    HI_GV_PARSER_Deinit();
    return -1;
}

Ret = HI_GV_App_Create("MainApp", (HI_HANDLE *)&hApp);
if (HI_SUCCESS != Ret)
{
    HI_GV_Deinit();
    HI_GV_PARSER_Deinit();
```



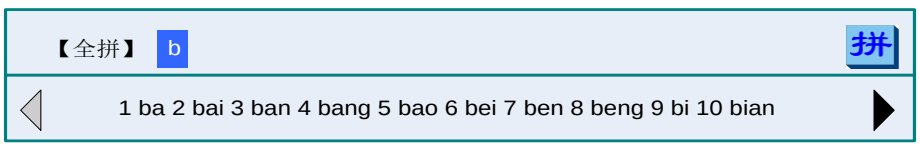
```
        return 0;
    }

    HI_GV_Widget_Show(allwidget);
    HI_GV_Widget_Active(allwidget);

    Ret = HI_GV_App_Start(hApp);
    if (HI_SUCCESS != Ret)
    {
        HI_GV_PARSER_Deinit();
        (HI_VOID)HI_GV_App_Destroy(hApp);
        HI_GV_Deinit();
        return 0;
    }

    printf("step 5:Destroy!\n");
    HI_GV_PARSER_Deinit();
    (HI_VOID)HI_GV_App_Destroy(hApp);
    HI_GV_Deinit();

    return 0;
}
```

5.7 输入法

HiGV 实现了拼音、英文大小写、数字、标点符号、自定义字符等输入法，用户可以打开或关闭这些输入法。

输入法还提供了输入法面板，这主要是对于复杂的输入法需要提供选择操作，特别是使用拼音输入法输入时，需要两级候选码进行输入，拼音输入法面板结构如图 5-1 所示。

图5-1 拼音输入法图示



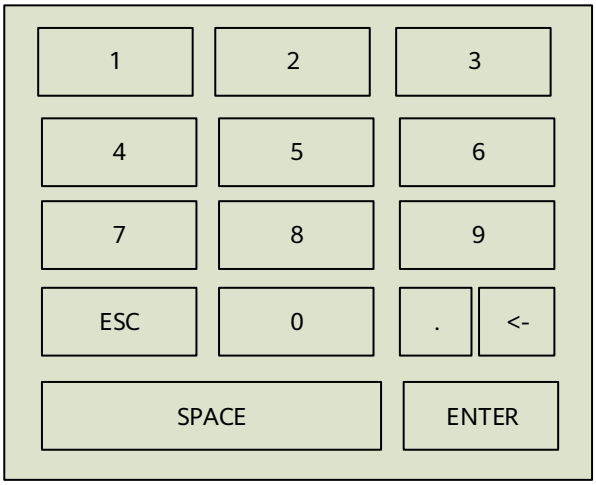
其中：

- 第一级候选码为键值组合的直接映射，比如拼音输入法中已输入键值所对应的候选拼音。
- 第二级候选码为一级候选码的映射，如用户所选拼音对应的汉字。

另外，为了适应有鼠标或触摸设备进行输入，HiGV 还增加软键盘的功能，用户可以通过接口或 XML 来创建软键盘及输入法切换窗口，其生成过程和普通界面一样，目前用户可以使用的键值是《hi_gv_msg.h》中所定义的虚拟键值。

软键盘的显示主要依赖当前的输入法，目前输入法已经支持拼音、数值、符号、大写字母、小写字母、自定义字符软键盘。

图5-2 数字软键盘图示



5.7.1 开发应用

输入法创建

输入法创建步骤如下：

步骤 1 在应用程序初始化过程中调用 HI_GV_IMEWINDOW_Create（）创建输入法，该函数输入参数需要设置候选码数目和输入法窗口外观图片资源。

步骤 2 为输入法窗口设置外观皮肤。

----结束

软键盘创建

软键盘创建步骤如下：

步骤 1 使用接口或在 xml 中定义软键盘的窗口、按钮等控件以及切换输入法的按钮窗口。

步骤 2 在应用程序中调用函数 HI_GV_IMEWINDOW_SetSoftKB（）注册软键盘和输入法。

----结束

5.7.2 参考代码

```
#define STD_KEY_NUM 64
#define NUM_KEY_NUM 15
#define EN_KEY_NUM 32
#define CAPEN_KEY_NUM 32

static HigvKeyMap g_keyValueMap[STD_KEY_NUM] = {{softkey_esc, HIGV_KEY_ESC, "Esc"},
{softkey_0, HIGV_KEY_0, "0"},
{softkey_1, HIGV_KEY_1, "1"}, {softkey_2, HIGV_KEY_2, "2"}, {softkey_3, HIGV_KEY_3, "3"},
{softkey_4, HIGV_KEY_4, "4"},
{softkey_5, HIGV_KEY_5, "5"}, {softkey_6, HIGV_KEY_6, "6"}, {softkey_7, HIGV_KEY_7, "7"},
{softkey_8, HIGV_KEY_8, "8"},
{softkey_9, HIGV_KEY_9, "9"}, {softkey_q, HIGV_KEY_q, "q"}, {softkey_w, HIGV_KEY_w, "w"},
{softkey_e, HIGV_KEY_e, "e"},
{softkey_r, HIGV_KEY_r, "r"}, {softkey_t, HIGV_KEY_t, "t"}, {softkey_y, HIGV_KEY_y, "y"},
{softkey_u, HIGV_KEY_u, "u"},
{softkey_i, HIGV_KEY_i, "i"}, {softkey_o, HIGV_KEY_o, "o"}, {softkey_p, HIGV_KEY_p, "p"},
{softkey_a, HIGV_KEY_a, "a"},
{softkey_s, HIGV_KEY_s, "s"}, {softkey_d, HIGV_KEY_d, "d"}, {softkey_f, HIGV_KEY_f, "f"},
{softkey_g, HIGV_KEY_g, "g"},
{softkey_h, HIGV_KEY_h, "h"}, {softkey_j, HIGV_KEY_j, "j"}, {softkey_k, HIGV_KEY_k, "k"},
{softkey_l, HIGV_KEY_l, "l"},
{softkey_dot, HIGV_KEY_DOT, "."}, {softkey_z, HIGV_KEY_z, "z"}, {softkey_x, HIGV_KEY_x, "x"},
{softkey_c, HIGV_KEY_c, "c"}, {softkey_v, HIGV_KEY_v, "v"}, {softkey_b, HIGV_KEY_b, "b"},
{softkey_n, HIGV_KEY_n, "n"},
{softkey_m, HIGV_KEY_m, "m"}, {softkey_del, HIGV_KEY_DEL, "Del"}, {softkey_space,
HIGV_KEY_SPACE, "Space"},
{softkey_left, HIGV_KEY_LEFT, "<--"}, {softkey_right, HIGV_KEY_RIGHT, "-->"}, {softkey_sy1, '!',
```



```
"!"},
{softkey_sy2, '"', "\""}, {softkey_sy3, '#', "#"}, {softkey_sy4, '$', "$"}, {softkey_sy5, '%', "%"},
{softkey_sy6, '&', "&"},{softkey_sy7, '\'', "'"}, {softkey_sy8, '(', "("}, {softkey_sy9, ')', ")"},
{softkey_sy10, '*', "*"}, {softkey_sy11, '+', "+"}, {softkey_sy12, ',', ","}, {softkey_sy13, '-', "-"},
{softkey_sy14, '/', "/"}, {softkey_sy15, ':', ":"}, {softkey_sy16, ';', ";"}, {softkey_sy17, '<', "<"},
{softkey_sy18, '=', "="}, {softkey_sy19, '>', ">"}, {softkey_sy20, '?', "?"}, {softkey_sy21, '@', "@"},
{softkey_sy22, '~', "~"}
};

static HigvKeyMap g_numberValueMap[NUM_KEY_NUM] = {{num_softkey_0, HIGV_KEY_0, "0"},
{num_softkey_1, HIGV_KEY_1, "1"},
{num_softkey_2, HIGV_KEY_2, "2"}, {num_softkey_3, HIGV_KEY_3, "3"}, {num_softkey_4,
HIGV_KEY_4, "4"},
{num_softkey_5, HIGV_KEY_5, "5"}, {num_softkey_6, HIGV_KEY_6, "6"}, {num_softkey_7,
HIGV_KEY_7, "7"},
{num_softkey_8, HIGV_KEY_8, "8"}, {num_softkey_9, HIGV_KEY_9, "9"}, {num_softkey_esc,
HIGV_KEY_ESC, "ESC"},
{num_softkey_del, HIGV_KEY_DEL, "Del"}, {num_softkey_space, HIGV_KEY_SPACE, "SPACE"},
{num_softkey_left, HIGV_KEY_LEFT, "<--"}, {num_softkey_right, HIGV_KEY_RIGHT, "-->"}
};

static HigvKeyMap g_enValueMap[EN_KEY_NUM] = {{en_softkey_a, HIGV_KEY_a, "a"},
{en_softkey_b, HIGV_KEY_b, "b"},
{en_softkey_c, HIGV_KEY_c, "c"}, {en_softkey_d, HIGV_KEY_d, "d"}, {en_softkey_e, HIGV_KEY_e,
"e"},
{en_softkey_f, HIGV_KEY_f, "f"}, {en_softkey_g, HIGV_KEY_g, "g"}, {en_softkey_h, HIGV_KEY_h,
"h"},
{en_softkey_i, HIGV_KEY_i, "i"}, {en_softkey_j, HIGV_KEY_j, "j"}, {en_softkey_k, HIGV_KEY_k, "k"},
{en_softkey_l, HIGV_KEY_l, "l"}, {en_softkey_m, HIGV_KEY_m, "m"}, {en_softkey_n, HIGV_KEY_n,
"n"},
{en_softkey_o, HIGV_KEY_o, "o"}, {en_softkey_p, HIGV_KEY_p, "p"}, {en_softkey_q, HIGV_KEY_q,
"q"},
{en_softkey_r, HIGV_KEY_r, "r"}, {en_softkey_s, HIGV_KEY_s, "s"}, {en_softkey_t, HIGV_KEY_t, "t"},
{en_softkey_u, HIGV_KEY_u, "u"}, {en_softkey_v, HIGV_KEY_v, "v"}, {en_softkey_w, HIGV_KEY_w,
"w"},
{en_softkey_x, HIGV_KEY_x, "x"}, {en_softkey_y, HIGV_KEY_y, "y"}, {en_softkey_z, HIGV_KEY_z,
"z"},
{en_softkey_esc, HIGV_KEY_ESC, "ESC"}, {en_softkey_dot, HIGV_KEY_DOT, "."}, {en_softkey_del,
```



```
HIGV_KEY_DEL, "Del"},
{en_softkey_space, HIGV_KEY_SPACE, "SPACE"}, {en_softkey_left, HIGV_KEY_LEFT, "<--"},
{en_softkey_right, HIGV_KEY_RIGHT, "-->"}
};

static HigvKeyMap g_capenValueMap[CAPEN_KEY_NUM] = {{capen_softkey_a, HIGV_KEY_A,
"A"},
{capen_softkey_b, HIGV_KEY_B, "B"}, {capen_softkey_c, HIGV_KEY_C, "C"}, {capen_softkey_d,
HIGV_KEY_D, "D"},
{capen_softkey_e, HIGV_KEY_e, "E"}, {capen_softkey_f, HIGV_KEY_F, "F"}, {capen_softkey_g,
HIGV_KEY_G, "G"},
{capen_softkey_h, HIGV_KEY_H, "H"}, {capen_softkey_i, HIGV_KEY_I, "I"}, {capen_softkey_j,
HIGV_KEY_J, "J"},
{capen_softkey_k, HIGV_KEY_K, "K"}, {capen_softkey_l, HIGV_KEY_L, "L"}, {capen_softkey_m,
HIGV_KEY_M, "M"},
{capen_softkey_n, HIGV_KEY_N, "N"}, {capen_softkey_o, HIGV_KEY_O, "O"}, {capen_softkey_p,
HIGV_KEY_P, "P"},
{capen_softkey_q, HIGV_KEY_Q, "Q"}, {capen_softkey_r, HIGV_KEY_R, "R"}, {capen_softkey_s,
HIGV_KEY_S, "S"},
{capen_softkey_t, HIGV_KEY_T, "T"}, {capen_softkey_u, HIGV_KEY_U, "U"}, {capen_softkey_v,
HIGV_KEY_V, "V"},
{capen_softkey_w, HIGV_KEY_W, "W"}, {capen_softkey_x, HIGV_KEY_X, "X"}, {capen_softkey_y,
HIGV_KEY_Y, "Y"},
{capen_softkey_z, HIGV_KEY_Z, "Z"}, {capen_softkey_esc, HIGV_KEY_ESC, "ESC"},
{capen_softkey_dot, HIGV_KEY_DOT, "."},
{capen_softkey_del, HIGV_KEY_DEL, "Del"}, {capen_softkey_space, HIGV_KEY_SPACE, "SPACE"},
{capen_softkey_left, HIGV_KEY_LEFT, "<--"}, {capen_softkey_right, HIGV_KEY_RIGHT, "-->"}
};

HI_S32 InputMethod_Init()
{
    HigvIMEWindowInit initdata;
    HIGV_STYLE_S winStyle;
    HI_RESID normalSkin = 0;
    HI_S32 ret;
    const HI_S32 sofbKBcnt = 4;
    const HI_S32 frameColor = 0xFFFFFF00;
    ret = memset_s(&winStyle, sizeof(winStyle), 0x00, sizeof(winStyle));
```



```
if (ret != EOK) {
    return ret;
}
winStyle.StyleType = HIGV_STYLETYPE_COLOR;
winStyle.BackGround.Color = 0xFF000080;
winStyle.FontColor = 0xFFFFFFFF;
winStyle.Top.Color = frameColor;
winStyle.Bottom.Color = frameColor;
winStyle.Left.Color = frameColor;
winStyle.Right.Color = frameColor;
winStyle.bNoDrawBg = HI_FALSE;
winStyle.LineWidth = 5;   // 5 pixel

ret = HI_GV_Res_CreateStyle(&winStyle, &normalSkin);

ret = memset_s(&initdata, sizeof(initdata), 0x00, sizeof(initdata));
if (ret != EOK) {
    return ret;
}
initdata.pixelFormat = HIGO_PF_BUTT;
initdata.capEnglishLogoIndex = (HI_PARAM)"./res/pic/ime/capenglishlogo.PNG";
initdata.englishLogoIndex = (HI_PARAM)"./res/pic/ime/englishlogo.PNG";
initdata.leftArrowPicIndex = (HI_PARAM)"./res/pic/ime/leftarrowlogo.PNG";
initdata.numberLogoIndex = (HI_PARAM)"./res/pic/ime/numberlogo.PNG";
initdata.pinyinLogoIndex = (HI_PARAM)"./res/pic/ime/pinyinlogo.PNG";
initdata.standardALogoIndex = (HI_PARAM)"./res/pic/ime/englishlogo.PNG";
initdata.standardBLogoIndex = (HI_PARAM)"./res/pic/ime/englishlogo.PNG";
initdata.rightArrowPicIndex = (HI_PARAM)"./res/pic/ime/rightarrowlogo.PNG";
initdata.symbolLogoIndex = (HI_PARAM)"./res/pic/ime/symbollogo.PNG";
initdata.unActiveLeftArrowPicIndex = (HI_PARAM)"./res/pic/ime/unactiveleft.PNG";
initdata.unActiveRightArrowPicIndex = (HI_PARAM)"./res/pic/ime/unactiveright.PNG";
initdata.pinYinTablePath = (HI_PARAM)"./res/pymb.bin";
initdata.pinYinTablePathLen = 14;
ret = HI_GV_IMEWINDOW_Create(HI_NULL, &initdata, &g_imeWinHandle);
if (ret != HI_SUCCESS) {
    printf("Failed to create the inputmethod window!\n");
    return ret;
}
```



```
HI_GV_Widget_SetSkin(g_imeWinHandle, HIGV_SKIN_NORMAL, normalSkin);

/* SoftKey init */
HigvIMEWindowSoftKB softKBVec[sofbKBcnt];

/* Standard soft keyboard,number and english HI_CHAR map */
HigvIMEWindowKeyValuePair keyValueMap[STD_KEY_NUM];
for (HI_S32 j = 0; j < STD_KEY_NUM; j++) {
    keyValueMap[j].keyWidget = g_keyValueMap[j].widgetHandle;
    keyValueMap[j].keyValue = g_keyValueMap[j].keyValue;
    HI_GV_Widget_SetText(keyValueMap[j].keyWidget, g_keyValueMap[j].text);
}

/* Number soft keyboard */
HigvIMEWindowKeyValuePair numkeyValueMap[NUM_KEY_NUM];
for (HI_S32 j = 0; j < NUM_KEY_NUM; j++) {
    numkeyValueMap[j].keyWidget = g_numberValueMap[j].widgetHandle;
    numkeyValueMap[j].keyValue = g_numberValueMap[j].keyValue;
    HI_GV_Widget_SetText(numkeyValueMap[j].keyWidget, g_numberValueMap[j].text);
}

/* Lowercase soft keyboard */
HigvIMEWindowKeyValuePair enkeyValueMap[EN_KEY_NUM];
for (HI_S32 j = 0; j < EN_KEY_NUM; j++) {
    enkeyValueMap[j].keyWidget = g_enValueMap[j].widgetHandle;
    enkeyValueMap[j].keyValue = g_enValueMap[j].keyValue;
    HI_GV_Widget_SetText(enkeyValueMap[j].keyWidget, g_enValueMap[j].text);
}

/* Capital soft keyboard */
HigvIMEWindowKeyValuePair capenkeyValueMap[CAPEN_KEY_NUM];
for (HI_S32 j = 0; j < CAPEN_KEY_NUM; j++) {
    capenkeyValueMap[j].keyWidget = g_capenValueMap[j].widgetHandle;
    capenkeyValueMap[j].keyValue = g_capenValueMap[j].keyValue;
    HI_GV_Widget_SetText(capenkeyValueMap[j].keyWidget, g_capenValueMap[j].text);
}
```



```
    /* Register soft keyboard in the imewindow */
    ret = memset_s(softKBVec, sizeof(softKBVec), 0x0, sizeof(softKBVec));
    if (ret != EOK) {
        return ret;
    }

    softKBVec[HI_GV_IMEWindow_PINYIN].isNoDispIMW = HI_FALSE;
    softKBVec[HI_GV_IMEWindow_PINYIN].softKB = im_softkb;
    softKBVec[HI_GV_IMEWindow_PINYIN].isDispSoftKB = HI_TRUE;
    softKBVec[HI_GV_IMEWindow_PINYIN].keyValueMap = keyValueMap;
    softKBVec[HI_GV_IMEWindow_PINYIN].keyNum = STD_KEY_NUM;

    softKBVec[HI_GV_IMEWindow_CAPENGLISH].isNoDispIMW = HI_TRUE;
    softKBVec[HI_GV_IMEWindow_CAPENGLISH].softKB = im_softkb_capen;
    softKBVec[HI_GV_IMEWindow_CAPENGLISH].isDispSoftKB = HI_TRUE;
    softKBVec[HI_GV_IMEWindow_CAPENGLISH].keyValueMap = capenkeyValueMap;
    softKBVec[HI_GV_IMEWindow_CAPENGLISH].keyNum = CAPEN_KEY_NUM;

    softKBVec[HI_GV_IMEWindow_ENGLISH].isNoDispIMW = HI_TRUE;
    softKBVec[HI_GV_IMEWindow_ENGLISH].softKB = im_softkb_en;
    softKBVec[HI_GV_IMEWindow_ENGLISH].isDispSoftKB = HI_TRUE;
    softKBVec[HI_GV_IMEWindow_ENGLISH].keyValueMap = enkeyValueMap;
    softKBVec[HI_GV_IMEWindow_ENGLISH].keyNum = EN_KEY_NUM;

    softKBVec[HI_GV_IMEWindow_NUMBER].isNoDispIMW = HI_TRUE;
    softKBVec[HI_GV_IMEWindow_NUMBER].softKB = im_softkb_num;
    softKBVec[HI_GV_IMEWindow_NUMBER].isDispSoftKB = HI_TRUE;
    softKBVec[HI_GV_IMEWindow_NUMBER].keyValueMap = numkeyValueMap;
    softKBVec[HI_GV_IMEWindow_NUMBER].keyNum = NUM_KEY_NUM;

    ret = HI_GV_IMEWINDOW_SetSoftKB(g_imeWinHandle, softKBVec, sofbKBcnt);
    if (ret != HI_SUCCESS) {
        printf("HI_GV_IMEWINDOW_SetSoftKB failed! Return: %d\n", ret);
        return ret;
    }

    return HI_SUCCESS;
}
```


5.8 线程安全

当工程中存在多个线程同时运行，在其他线程调度 HiGV 的功能接口改变控件数据是不安全的，我们需要借助发送消息的方式在其他线程完成对 HiGV 的行为改变。

步骤 1 注册消息回调函数。

HiGV 消息注册有两种方式：

- 将业务实现封装成函数并通过接口 HI_GV_Widget_SetMsgProc 注册成为控件消息事件回调函数。
- 在布局 xml 文件中注册消息回调函数，消息回调函数类型可参考《HiGV 标签使用指南》文档。

步骤 2 在其他线程向 HiGV 发送消息。

步骤 3 HiGV 接收到消息执行注册回调函数。

----结束

说明
消息发送接口请参考头文件《hi_gv_msg.h》，消息事件请参考头文件《hi_gv_widget.h》。

5.9 动画

5.9.1 创建动画方法 1

动画组件是一个实现 UI 界面动画效果的 API，HiGV 目前只支持补间动画（Tween 动画），该动画主要是通过对动画对象（包括控件或图片等）的动画属性（包括位置属性，Alpha 等）不断变化而产生的动画效果，而动画在不同时间点上运行速度是通过 Easing 函数来控制。对于 Easing 函数的解释可参考网站 http://easings.net/zh-cn。

目前 HiGV 部分控件内部已经支持动画效果，如果用户需要定制动画效果也可以调用动画组件的接口进行控件级别的动画。

步骤 1 调用接口 HI_GV_TweenAnimCreate 创建动画实例；

步骤 2 调用接口 HI_GV_TweenAnimSetDuration 设置动画运行时长 Duration；

步骤 3 调用接口 HI_GV_TweenAnimAddTween 为动画实例设置 Easing 类型（包括 Transition 和 Ease）和动画属性的开始和结束范围；

步骤 4 注册动画监听回调函数，包括动画启动，动画运行，动画结束的回调函数，其中动画效果的实现主要依赖于运行回调函数的实现，该回调函数需要用户进行更新动画对象属性和刷新界面，从而产生动画效果。动画属性对象可以通过接口 HI_GV_TweenAnimGetTweenValue 获取 Easing 函数计算后的属性值；

步骤 5 调用接口 HI_GV_TweenAnimStart 启动动画。

----结束

5.9.2 创建动画方法 2

HiGV 提供了部分默认的补间动画效果（平移、Alpha 渐变、卷帘等），通过 XML 配置即可实现。也可通过调用接口动态创建动画（具体参考 3.5.2 “资源的使用”中的动画信息）

步骤 1 在 res/xml/anim.xml 中定义动画信息；如定义平移动画：

```
<translate
    id="ANIM_BTN_MOVE"
    duration_ms=500
    repeatcount=3
    delaystart=20
    animtype=0
    fromx=20
    fromy=6
    tox=252
    toy=260
/>
```

步骤 2 在控件中关联动画

```
<button
id="ANIM_BUTTON"
...
Anim="ANIM_BTN_MOVE"
/>
```

按钮"ANIM_BUTTON"所在窗口显示时，根据按钮关联的动画"ANIM_BTN_MOVE"，会进行动画显示。

----结束

5.9.3 列表控件回弹效果

HiGV 针对 ListBox、ScrollView、ScrollGrid、WheelView 提供默认的回弹效果。

创建步骤如下：

在列表框 XML 资源定义时增加“reboundmax”属性，设置回弹最大值即可。

5.10 性能优化指导

HiGV 开发过程中，从资源到具体控件，都需要设置许多的参数配置，这些配置影响的整个图形系统的功能和性能，本章节主要讲述各配置对界面性能及功能的影响。

5.10.1 控件/资源实例

- 控件和资源实例越多，所须的创建和加载时间就越长。控制控件和资源的实例数量，尽可能复用属性相近的控件（如弹出提示框）和资源（多个控件可以使用同一个资源），减少不必要的实例创建。
- 一般情况下，固定且常用的控件通过 xml 来创建，在某些特定场景控件的属性实时变化，可以采用接口动态创建，用完即刻销毁的方式来实现。
- 频繁显示、刷新控件会影响性能，控制控件的刷新次数，避免多余刷新。
- 通过接口创建的资源实例不再使用时须销毁，多次重复创建一个资源实例不会造成内存泄露，HiGV 会检查创建的资源是否已经存在，HiGV 去初始化时会释放所有资源。

5.10.2 图片型皮肤优化

- 利用九宫格皮肤的特点，将资源拆分为多个小图片，组合重复利用。
- HiGV 绘制图片时，如果非缩放模式会平铺图片。因此可以将大块的单色图片裁剪为 1 像素的小图片；水平方向颜色无变化的图片裁剪至宽度为 1 像素的线条图片。

5.10.3 数据模型合理使用

- 在控件新绑定数据模型或数据发生变化时才同步数据，同步数据需要的时间和数据量有关。

- 多使用 HI_GV_Widget_SyncDB 同步数据，只有特定场景需要同步数据模型绑定的所有数据才使用 HI_GV_ADM_Sync。
- UserDB 类型的数据模型合理分配缓冲行数 BufferRows，xml 属性为 cacherows。缓冲行数越多一次获取的数据就越多，可以减少调用回调函数获取数据的次数，但是数据模型创建时所需要的内存就越多。

5.10.4 隐藏释放风格和窗口切换性能

- 为了提高开机加载速度，HiGV 所有界面的皮肤资源加载是在控件第一次显示的时候进行的，加载图片资源会申请内存。
- 当控件风格为隐藏释放资源风格时（xml 属性 isrelease），控件隐藏后会释放掉图片、内部定时器等资源，但下次显示时需要消耗时间重新加载这些资源。
- 当控件风格为隐藏不释放资源风格时，可以省去控件隐藏后再次显示加载资源的步骤。考虑到内存管理希望开机就加载这些不释放的资源，可以调用接口 HI_GV_Win_LoadSkin()，该接口只对隐藏不释放的控件有效。
- 页面切换时**先显示（HI_GV_Widget_Show）并激活（HI_GV_Widget_Active）下一个界面，再隐藏（HI_GV_Widget_Hide）上一个界面**，因为 HiGV 不允许焦点处于隐藏控件，如果直接隐藏焦点控件，HiGV 会去寻找一个可以激活的控件，这可能会浪费时间。另外，显示控件使用到的图片资源不会被释放，不会再花时间加载已利用的资源。
- 在物理内存（MMZ）特别紧张的情况下，为了对降低物理内存的消耗，在页面切换时可以考虑使用先隐藏（HI_GV_Widget_Hide），再显示（HI_GV_Widget_Show）并激活（HI_GV_Widget_Active）的调用流程，功能上不会有影响，但这个不是推荐的操作。

