GUIslice API

Developer Guide

Ver: 0.16.0

Publication date and software version

Published XXXXXX, 2021. Based on GUIslice API Library 0.16.0

Chapter 1 Introduction

1.1 Introduction

GUIslice is User Interface platform designed for embedded microcontroller systems, most without any operating system. The design is such that it trys to minimize overhead. Achieving this goal by being a pure C library with no dynamic memory allocation.

It's not so much designed to be to be easy to use as much as it's meant to be efficient. Whatever difficulty in defining storage declarations and other essential defines, enums, and UI coordinates is mitigated by another tool called the GUIsliceBuilder which will not be further discussed here.

This document explains the main structure and design of the GUIslice library. It's purpose is to give a deeper understanding than one would get by simply knowing the API. While GUIslice comes with excellent documentation for the API with many examples there are times when someone needs to view below the covers. Say, you have to extend GUIslice with new UI Elements or create moveable windows. Hopefully this document will supply enough information for anyone maintaining or extending the package.

Certain topics that could cloud the explanations of logic flow such as frame rate tracking, or routines that are very small and that have self-describing names (such as gslc_PageFlipSet) will be either avoided or confined to the Appendix. If you need more information you will need to examine the actual source code.

Chapter 2 Architecture

2.1 Phases

2.1.1 C Language definition phase

The GUIslice API storage requirements must be laid out in globals. In addition, various enums for pages (menus), fonts, UI Elements, and images must be defined at the outset.

We start with defining `MAX_PAGE` as the total number of pages to be used. Followed by an enum giving the number of UI Elements (widgets) to be placed on each page and an enum for the number of Fonts accessed, if any. Optionally, depending upon usage, enum for each font and/or images used must also be provided.

These definitions must be followed by actual storage allocations for pages, fonts, and all UI Elements.

Source code must also be provided for all callbacks.

The important data structures that will be discussed in this document are:

- gslc_tsGui
- gslc_tsDriver
- gslc_tsPage
- gslc_tsFont
- gslc_tsElem
- gslc_tsElemRef

Lesser structures will also be exposed as they come up.

2.1.2 Setup

For Micro-Controllers using Arduino's IDE for development the Application will have been provided with a `Setup()` and `Loop()` function. For Linux initialization will happen in the `Main()` function.

At the start of an Application, GUIslice API will need to initialize API storage structs and link them together. The Application will also need to provide to the GUIslice API any details required by the various UI elements, like coordinates, sizes, colors, fonts, and callback addresses.

It all starts with an optional call to `gslc_InitDebug()` to setup debugging messaging. This call can be removed when you deploy your App. The first routine that must be called is `gslc_Init()` which is responsible for initializing the global gslc_tsGui struct and starting up the third party TFT driver and optionally touch interface.

After that `gslc_PageAdd()` must be called for each page used in the application, followed by whatever API calls are needed to create the UI Elements to be placed on each page.

The details of these routines will be shown in *Chapter 4 API Internals*.

2.1.3 Loop

Once the API and the Application's requirements for initialization have been completed the Loop phase is entered. Arduino like apps will have an actual `Loop()` function while Linux users must provide a `while()` loop.

GUIslice is an event loop non-preemptive driven API which does not depend upon multi-threading support. What this means is that the Application needs to call a function periodically to detect any user or device interactions.

The routine GUIslice provides for this is called `gslc_Update()` and breaking this down requires it's own section so see *Chapter 4 gslc_Update* for a complete discussion. Of course, `Loop()` is also where you provide your own Application Logic just be sure to periodically call `gslc_Update()` or you won't properly handle your UI.

2.2 Layers

There are three basic layers to GUIslice API.

Layer one is exposed API meant for users to call directly.

Layer two is the driver interface layer. The specific driver is choosen by the user of the API by uncommenting the desired driver configuration file inside `GUIslice_config.h`.

Layer three contains actual third party drivers that are called from layer two. This would be for the TFT graphics, Touch and File access support.

2.3 Hierarchical view of data structures

2.4 Elements vs Element References

2.5 Singular vs Compound Elements

Chapter 3 Driver Modules

3.1 Display drivers

While we have discussed the existence of the driver layer where files are of the form `GUIslice_drv_xxxx.<h,cpp>` ; They're two additional files we should mention, `GUIslice_config_ard.h` and `GUIslice_config_linux.h`.

These are holdovers from the early days of GUIslice API. You generally don't need to deal with them unless you are adding a new driver pair say, `GUIslice_drv_mydriver.<h,cpp>`. In this case you need to add your include file to either of these files to allow GUIslice to use your new driver.

In the future one or both may be removed but for now you need to be aware of them.

One more point, while GUIslice prefixes the driver files with "*drv*" they are not in fact drivers. In fact, It might best be renamed the Graphics layer. If you open, `GUIslice_drv_adagfx.cpp` you will soon see why. It's littered with `#if defined` , `elseif` , and `#endif` statements to deal with individual TFT and Touch drivers. Having a fourth layer that handles the interfaces to real drivers would make the code much easier to follow and maintain. Again this is an artifact of how GUIslice evolved over time.

This isn't included as a criticism as much as a warning of what you will face if you need to extend an existing driver file.

For many of these the calls translate to a simple one to one calls to say, Adafruit_GFX routines or whatever Graphics package is being used. The complex part however is in font handling where each TFT DRiver seems to have a different approach and with Touch drivers that also need special handling.

You will need to study the source code to appreciate whats involved.

3.2 Touch drivers

3.2.1 Debouncing

3.2.2 Filtering

3.2.3 Calibration scaling

3.2.4 Rotation

3.2.5 gslc_InitTouchHandler()

If you need special handling of the touch interface or to support a new touch chip there is an alternative approach you can take rather than modifying the driver files. As briefly discussed in an earlier section GUIslice API has provided the call `gslc_InitTouchHandler()` which allows to you inteface touch through your own code. You can see *GUIslice/examples/arduino/ex16_ard_touch_hnd* for a demonstration.

3.3 Identifying What features the Driver supports

GUIslice provides default implementations for certain API calls such as, `gslc_DrvDrawPoints()`, and `gslc_DrvDrawFrameRect()`, among others. Now most graphic drivers have optimized versions of these routines. So how does GUIslice know to use them? If you open `GUIslice_drv_adagfx.h` you will see a group of `#defines` specifying whats supported by the driver and what GUIslice must supply instead.

```
#define DRV_HAS_DRAW_POINT             1 ///< Support gslc_DrvDrawPoint()

#define DRV_HAS_DRAW_POINTS            0 ///< Support gslc_DrvDrawPoints()
#define DRV_HAS_DRAW_LINE              1 ///< Support gslc_DrvDrawLine()
#define DRV_HAS_DRAW_RECT_FRAME        1 ///< Support gslc_DrvDrawFrameRect()
#define DRV_HAS_DRAW_RECT_FILL         1 ///< Support gslc_DrvDrawFillRect()
#define DRV_HAS_DRAW_RECT_ROUND_FRAME  1 ///< Support gslc_DrvDrawFrameRoundRect()
#define DRV_HAS_DRAW_RECT_ROUND_FILL   1 ///< Support gslc_DrvDrawFillRoundRect()
#define DRV_HAS_DRAW_CIRCLE_FRAME      1 ///< Support gslc_DrvDrawFrameCircle()
#define DRV_HAS_DRAW_CIRCLE_FILL       1 ///< Support gslc_DrvDrawFillCircle()
#define DRV_HAS_DRAW_TRI_FRAME         1 ///< Support gslc_DrvDrawFrameTriangle()
#define DRV_HAS_DRAW_TRI_FILL          1 ///< Support gslc_DrvDrawFillTriangle()
#define DRV_HAS_DRAW_TEXT              1 ///< Support gslc_DrvDrawTxt()
#define DRV_HAS_DRAW_BMP_MEM           0 ///< Support gslc_DrvDrawBmp24FromMem()

#define DRV_OVERRIDE_TXT_ALIGN         0 ///< Driver provides text alignment
```

The Zero indicates not supported and one declares support.

Chapter 4 API Internals

Here the discussion will be about the more important API's. Even the actual UI Element creation API's will only be represented by a couple of UI Elements, `gslc_ElemCreateTxt()`, `gslc_ElemCreateBtnTxt()`, and `gslc_ElemCreateLine`. This is because the existing User API documentation is more than adequate for understanding. Instead the idea is to show the logic that applies to any such call within the API. What structures are involved, and how they are used by the lower level routines.

The API for `gslc_ElemCreateImg()` and `gslc_ElemCreateBtnImg()` will be explored in *Chapter 7 Special Features*.

4.1 gslc_Init()

```
bool gslc_Init(gslc_tsGui* pGui,void* pvDriver,gslc_tsPage* asPage,uint8_t nMaxPage,
  gslc_tsFont* asFont,uint8_t nMaxFont);
```

The pieces of gslc_tsGui struct to be explained in this document

Data Type	Field Name	Comment
gslc_tsPage*	asPage	Array of all pages defined in system
uint8_t	nPageMax	Maximum number of pages that can be defined
uint8_t	nPageCnt	Current number of pages defined
gslc_tsFont*	asFont	Collection of loaded fonts
uint8_t	nFontMax	Maximum number of fonts to allocate
uint8_t	nFontCnt	Number of fonts allocated
void*	pvDriver	Driver-specific members (gslc_tsDriver*)
gslc_tsElem	sElemTmpProg	Temporary element for Flash compatibility
gslc_tsPage*	apPageStack[GSLC_STACK__MAX]	Stack of displayed pages
bool	abPageStackActive[GSLC_STACK__MAX]	Flags pages that can receive touch events
bool	abPageStackDoDraw[GSLC_STACK__MAX]	Flags pages in stack that are actively drawn
gslc_tsPage*	apPageStack[GSLC_STACK__MAX]	Stack of displayed pages
bool	bInvalidateEn[GSLC_STACK__MAX]	A region of the display has been invalidated
gslc_tsRect	rInvalidateRect	The rect region that has been invalidated
gslc_tsElem	sElemTmp	Temporary element
gslc_tsElemRef	sElemRefTmp	Temporary element reference

This routine is responsible for setting up the gslc_tsGui, gslc_tsDriver, and gslc_tsFont structures. It will also fire up the TFT Display driver and any touch interface or even GPIO interface.

gslc_tsGui is the main container of data for GUIslice. Virtually everything needed is attached to this struct. That's the reason it's required in every call to the API. Most of what is in here is self-explanatory, like nDispW, and nDispH. So for these the source code comments are sufficient for understanding.

The items in yellow will be addressed in this chapter while the ones in GREEN outline will be discussed in *Chapter 4 gslc_Update()*. Thoses in BLUE will be in *Chapter 7 Extending GUIslice*.

By convention gslc_tsGui pGui will the address of `m_Gui`, pvDriver will be set to address of `m_drv`, asPage will be set to the address of the `m_asPage[MAX_PAGE]` array and asFont will set to address `m_asFont[MAX_FONT]`.

`gslc_Init()` will begin by Initializing state of the display values inside gslc_tsGui to zeroes, set `nPageMax=nMaxPage`, `nPageCnt=0`, `nFontMax=nMaxFont`, `nFontCnt=0`, default display orientation to `GSLC_ROTATE`. It will also init the page stack to NULL.

Diagram 4.1A gives the overall flow of this routine and it shows a driver specific function will be called to init the TFT driver and fill out pvDriver structure. The driver names will be of the form GUIslice_drv_xxxx.cpp where "xxxx" could be adagfx, m5stack, sdl, tft_espi, or any others that might be supported now or in the future. Chapter 6 Driver Modules will explore this deeper.

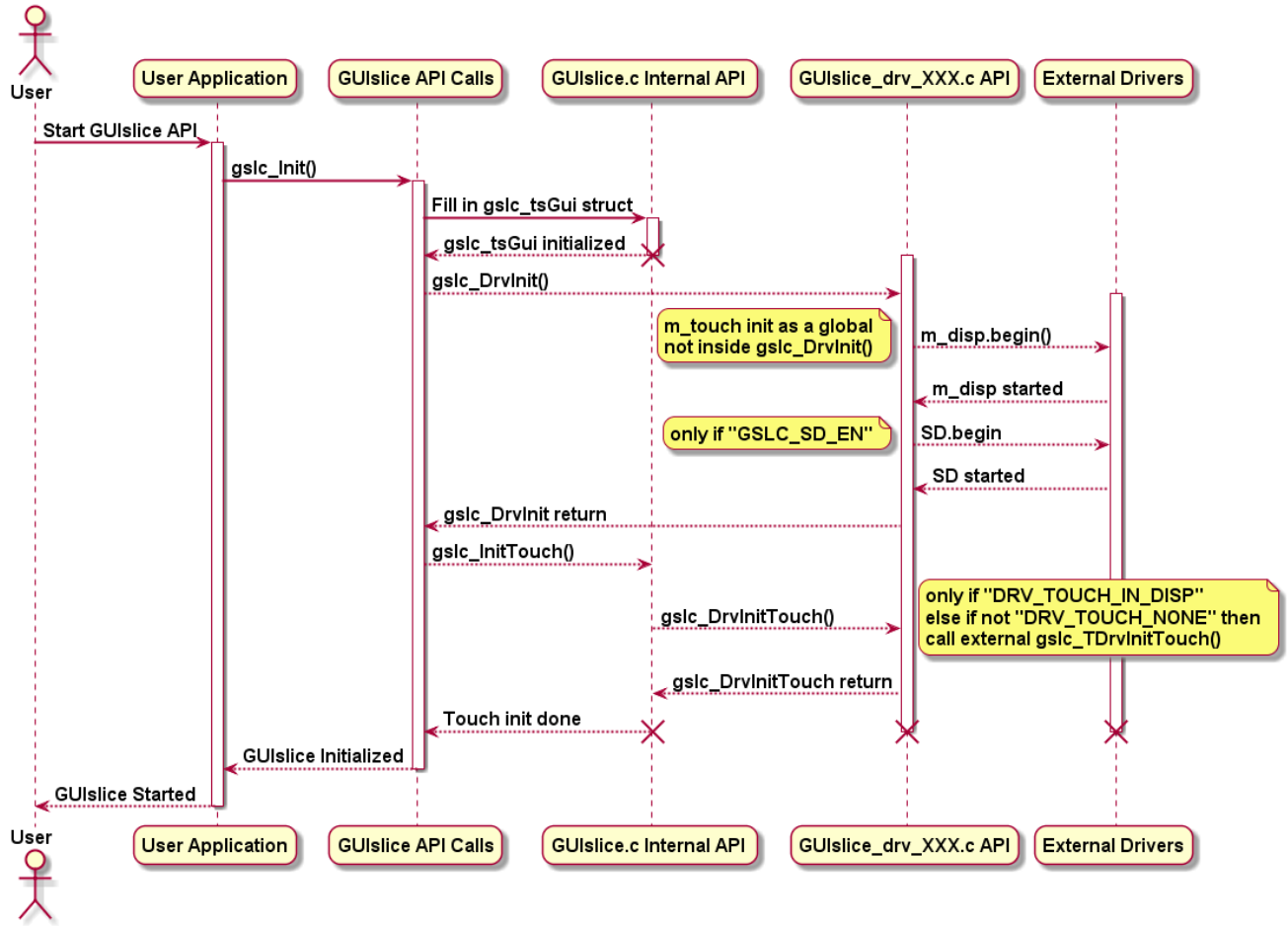

Diagram 4.1A gslc_Init() Flow

As you can see from the flow diagram we need to go to the GUIslice_drv_XXXX.cpp driver layer to access the touch hardware. Unless the user supplied a custom touch handler by registering it with a `gslc_InitTouchHandler()` call. In which case the user will have sub-classed TouchHandler and set their config file to use this implementation at the driver layer.

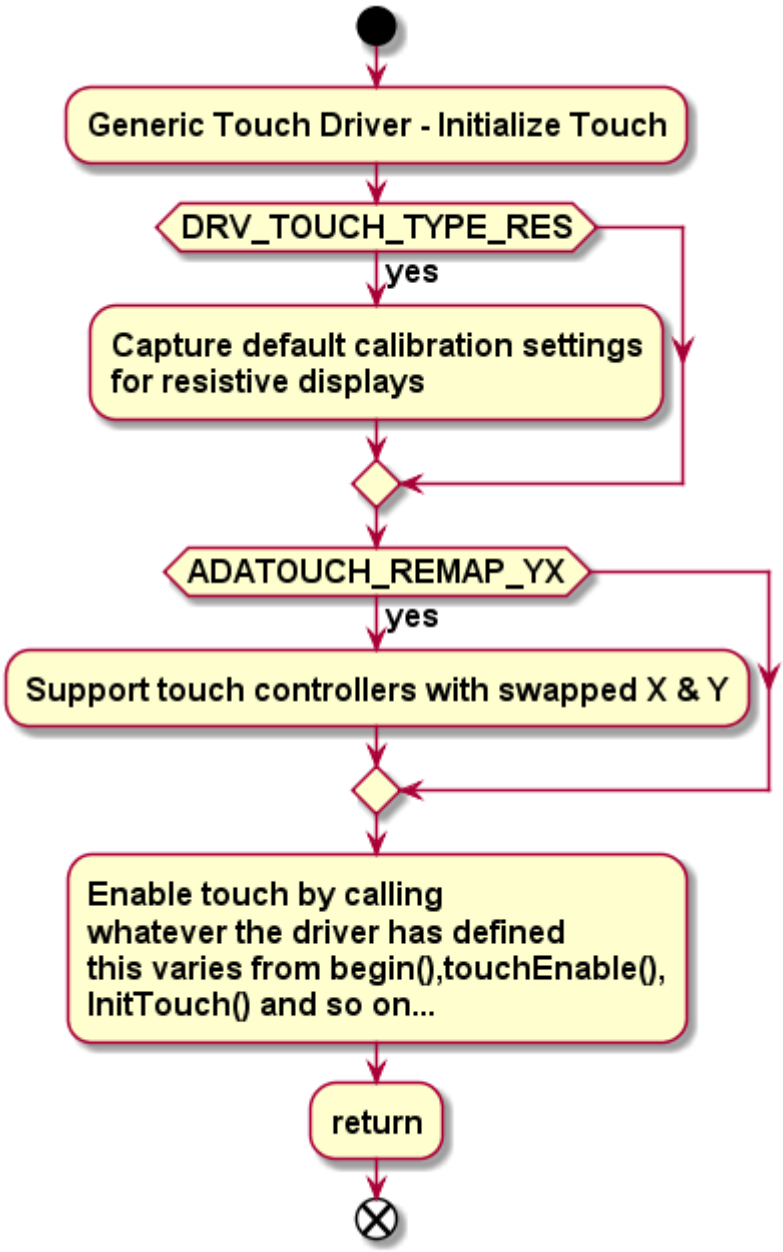

Diagram 4.1B Touch Initialization Flow

4.2 gslc_PageAdd()

```
void gslc_PageAdd(gslc_tsGui* pGui,int16_t nPageId,gslc_tsElem* psElem,
  uint16_t nMaxElem,gslc_tsElemRef* psElemRef,uint16_t nMaxElemRef)
```

gslc_PageAdd() fills in gslc_tsPage struct

Data Type	Field Name	Comment
gslc_tsCollect	sCollect	Collection of elements on page
int16_t	nPageId	Page identifier
gslc_tsRect	rBounds	Bounding rect for page elements

This routine adds a new page to the globally defined page array that was passed into `gslc_Init()` function (`&m_asPage[]`).

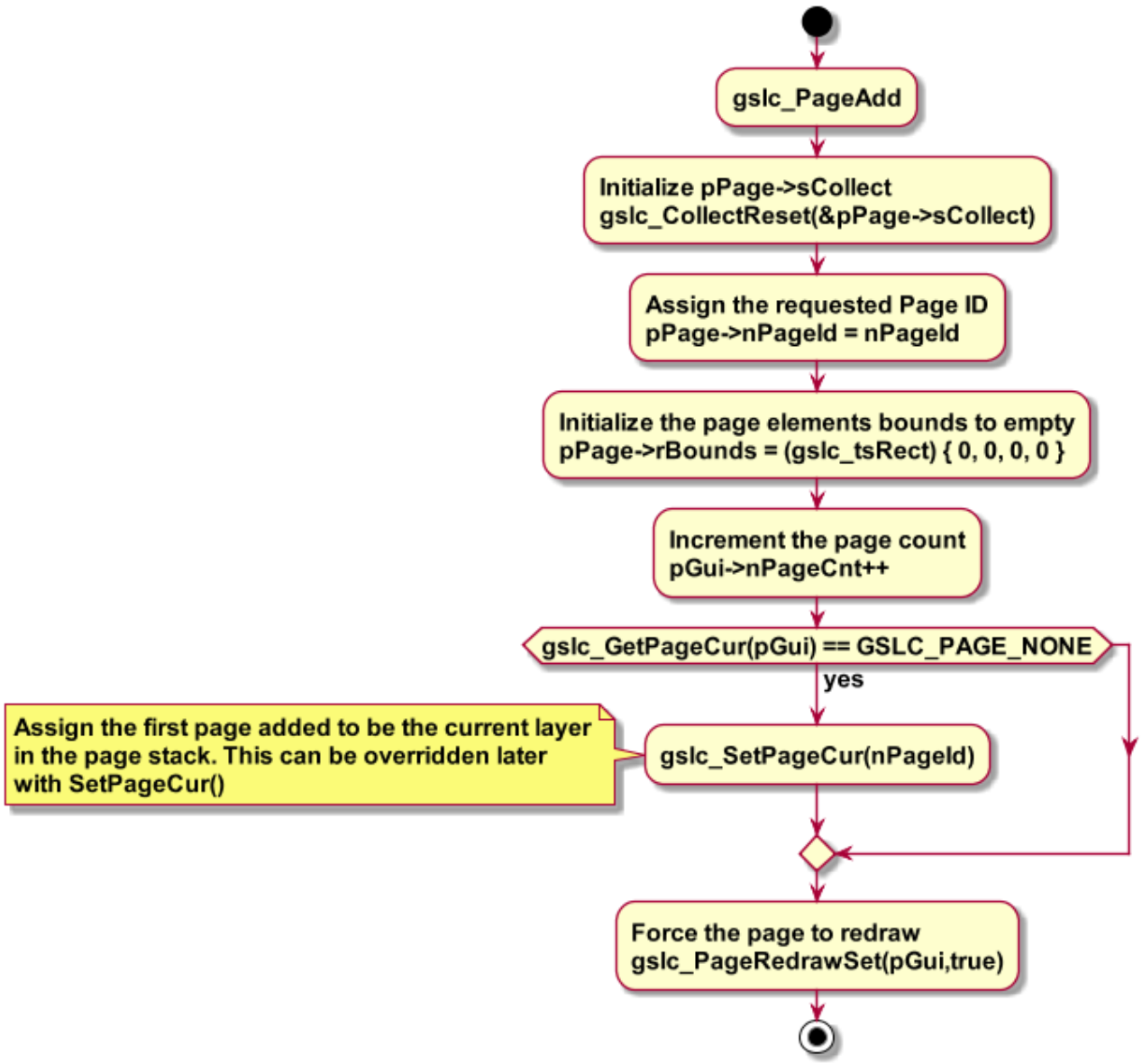

Diagram 4.2 gslc_PageAdd() Flow

Chapter 5 gslc_Update()

```
void gslc_Update(gslc_tsGui* pGui)
```

The polling loop for GUIslice. Performs GUIslice handling functions for any touch events and screen drawing. Note that nothing appears on screen until this routine is called.

Diagram 5.A gslc_Update() Flow

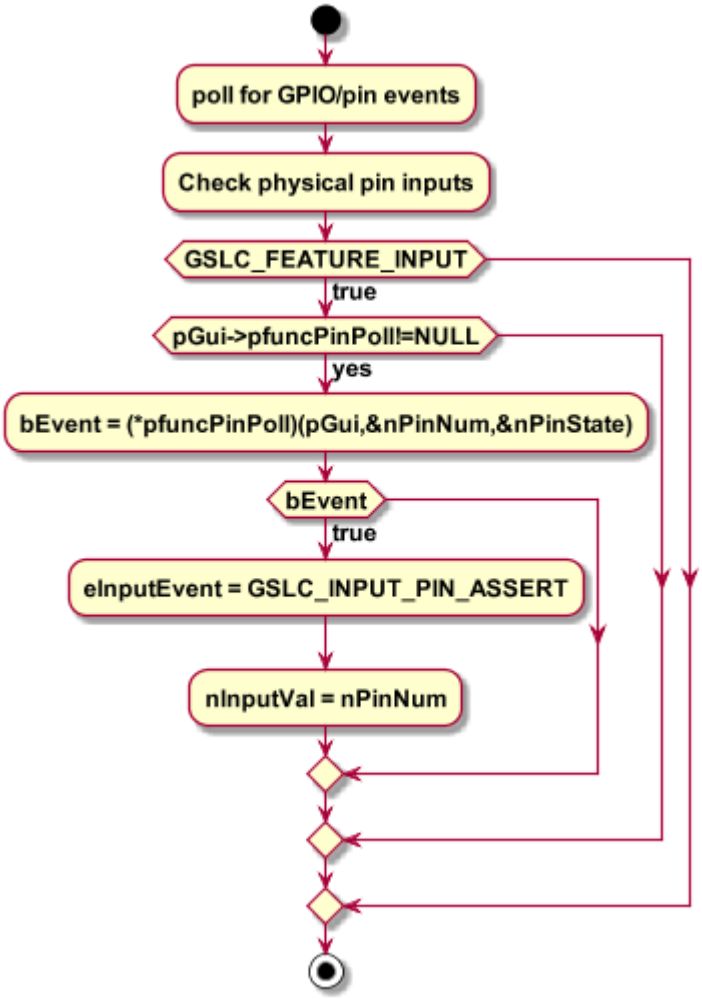

Diagram 5.B Poll Input GPIO/pin Flow

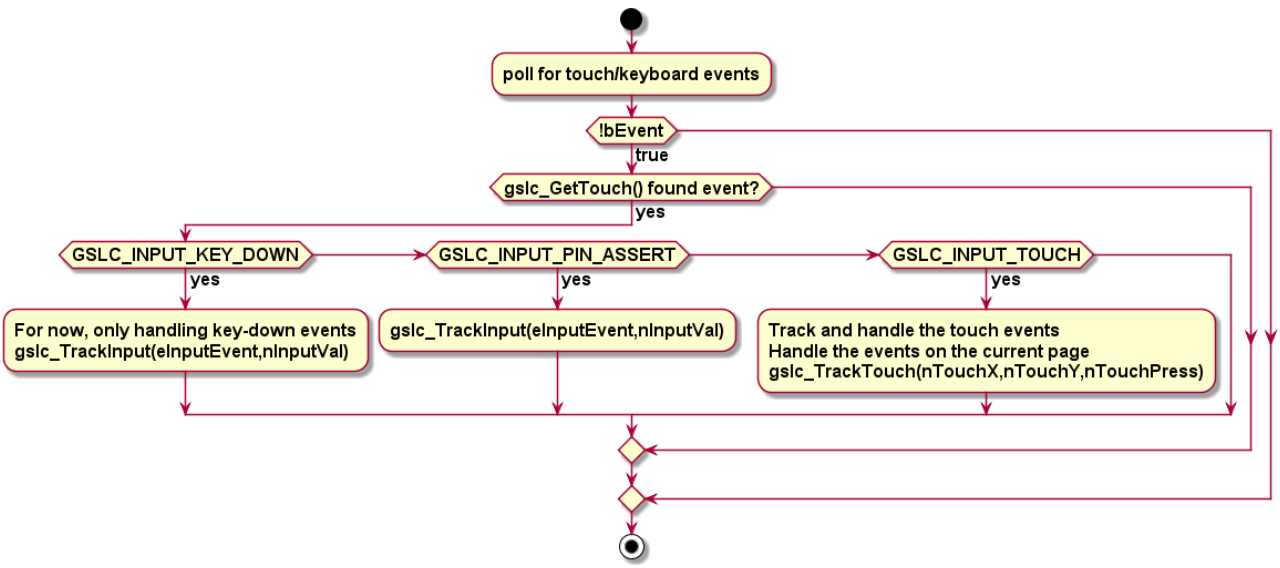

Diagram 5.C Poll Touch Flow

5.1 gslc_GetTouch()

```
bool gslc_GetTouch(gslc_tsGui* pGui, int16_t* pnX, int16_t* pnY, uint16_t* pnPress,
gslc_teInputRawEvent* peInputEvent, int16_t* pnInputVal);
```

The touch handling logic is used by both the touchscreen handler as well as the GPIO/pin/keyboard input controller. It should be mentioned that while the source code uses the term "Keyboard" GUIslice API really isn't providing full keyboard support, unlike the TFT Keypad Extended UI Element. It's more of lower level a key has been pressed support. Also, no key debouncing is provided.

Get the last touch event from the Touch Driver handler.

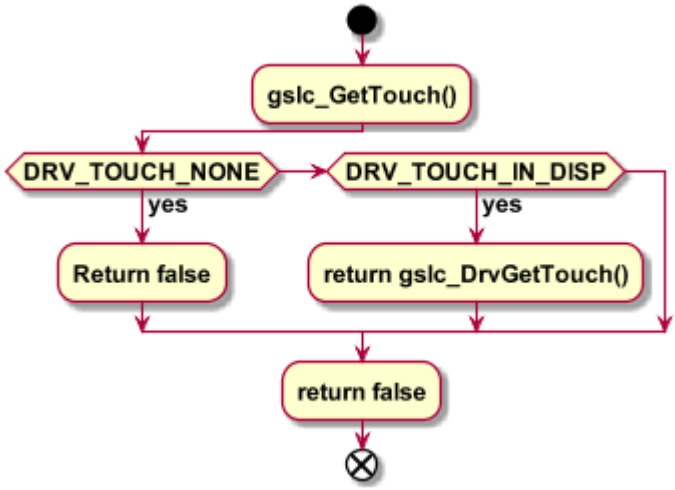

Diagram 5.1A gslc_GetTouch() Flow

The code for grabbing a touch event is quite complex so the source code within the driver of interest is the best way to see specifics. Independent of the actual implementation the overall flow is like this:

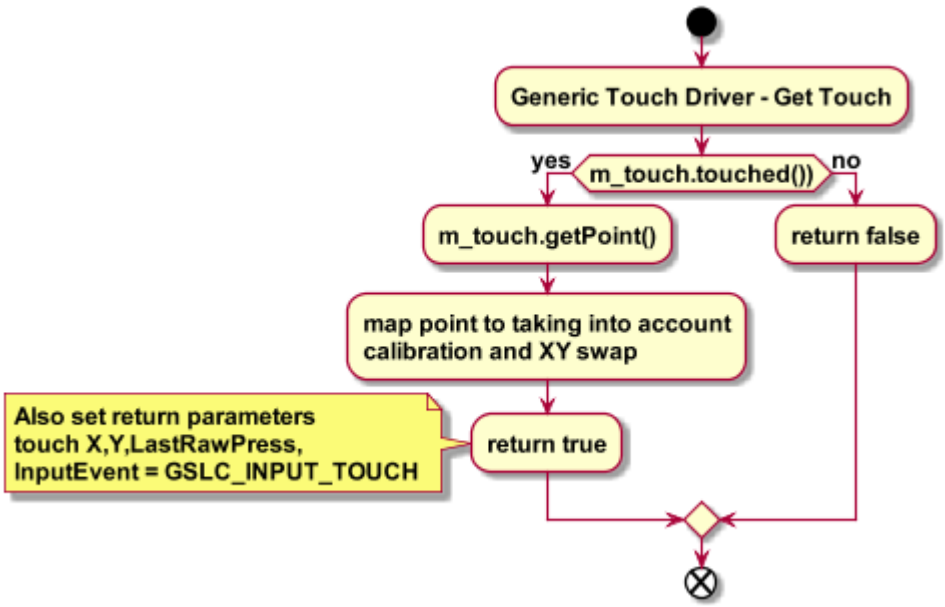

Diagram 5.1B Generic Touch Driver Flow

5.2 gslc_TrackTouch()

```
void gslc_TrackTouch(gslc_tsGui* pGui,gslc_tsPage* pPage,int16_t nX,int16_t nY,uint16_t
nPress)
```

Handles a touch event and performs the necessary tracking, glowing and selection actions depending on the press state.

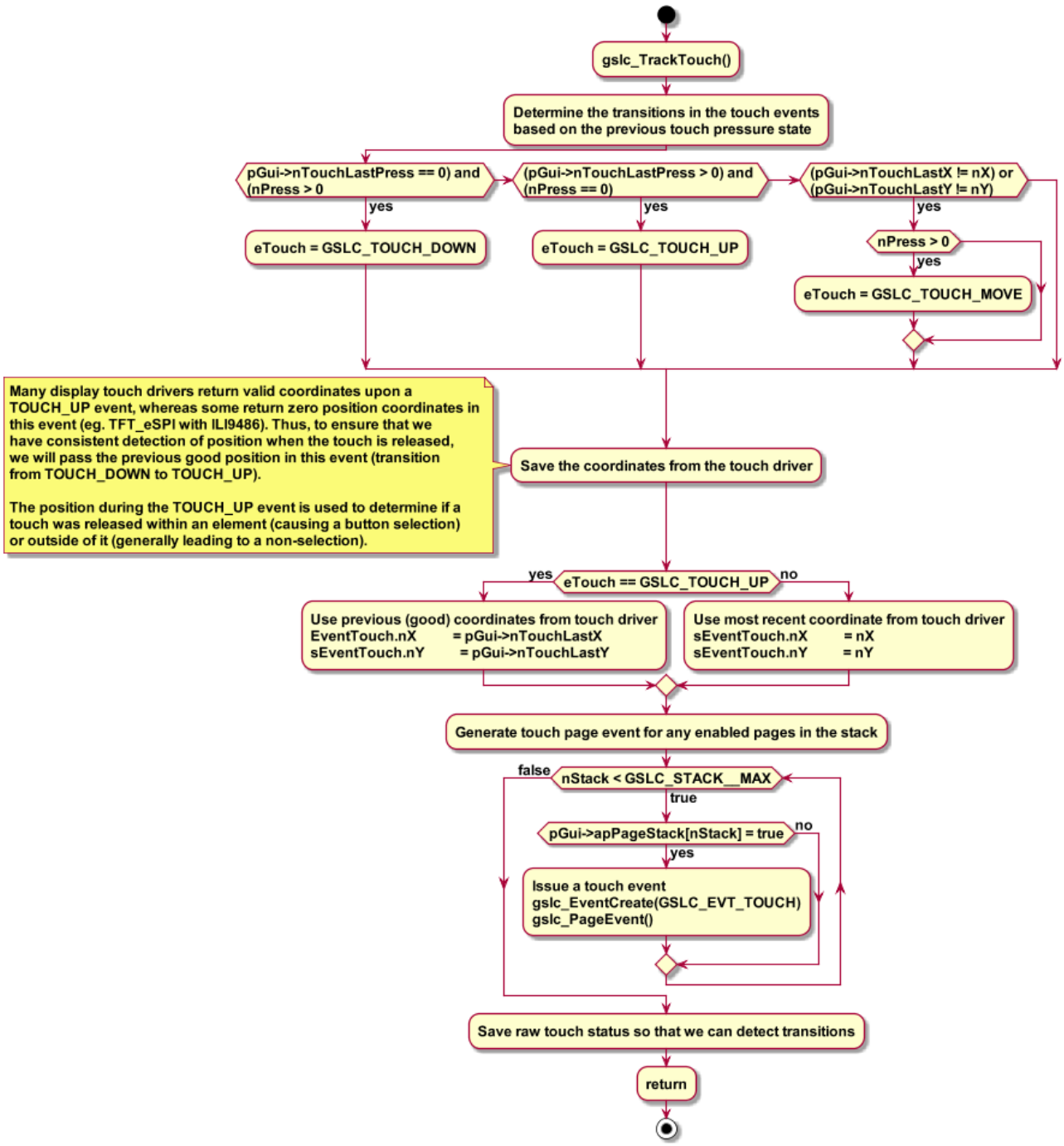

Diagram 5.2 gslc_TrackTouch() Flow

5.3 gslc_TrackInput()

```
void gslc_TrackInput(gslc_tsGui* pGui,gslc_tsPage* pPage,gslc_teInputRawEvent
eInputEvent,int16_t nInputVal);
```

Handles a direct input event from a keyboard or GPIO pin and performs the necessary tracking, glowing and selection actions depending on the state.

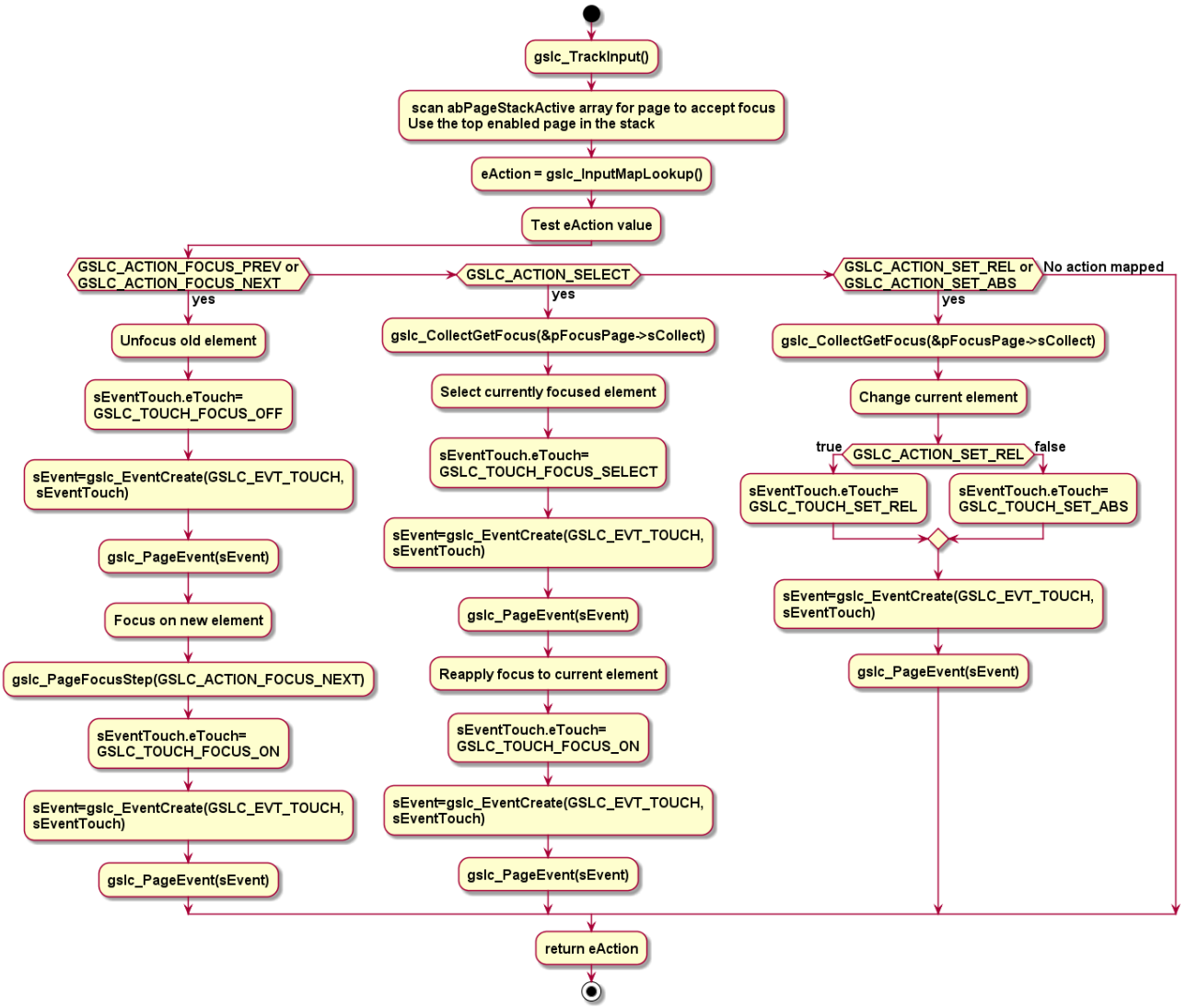

Diagram 5.3 gslc_TrackInput() Flow

5.4 gslc_EventCreate

```
gslc_tsEvent  gslc_EventCreate(gslc_tsGui* pGui,gslc_teEventType eType,uint8_t
nSubType,void* pvScope,void* pvData)
```

This routine uses the passed in parameters to fill in an event structure. This structure will guide the lower routines logic and also the overall handling of the UI.

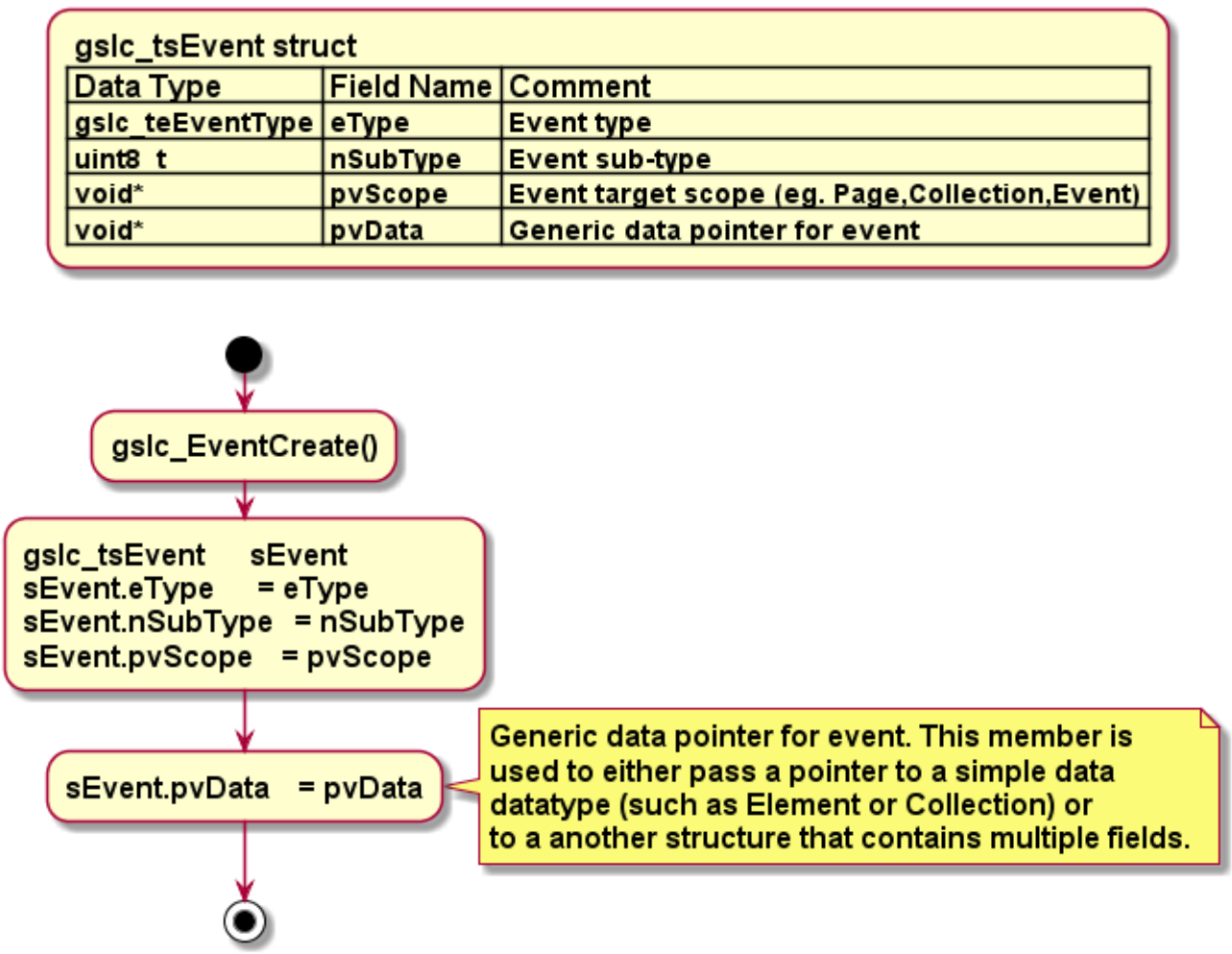

Diagram 5.4 gslc_EventCreate() Flow

5.5 gslc_PageEvent()

```
bool gslc_PageEvent(void* pvGui,gslc_tsEvent sEvent);
```

Common event handler function for a page.

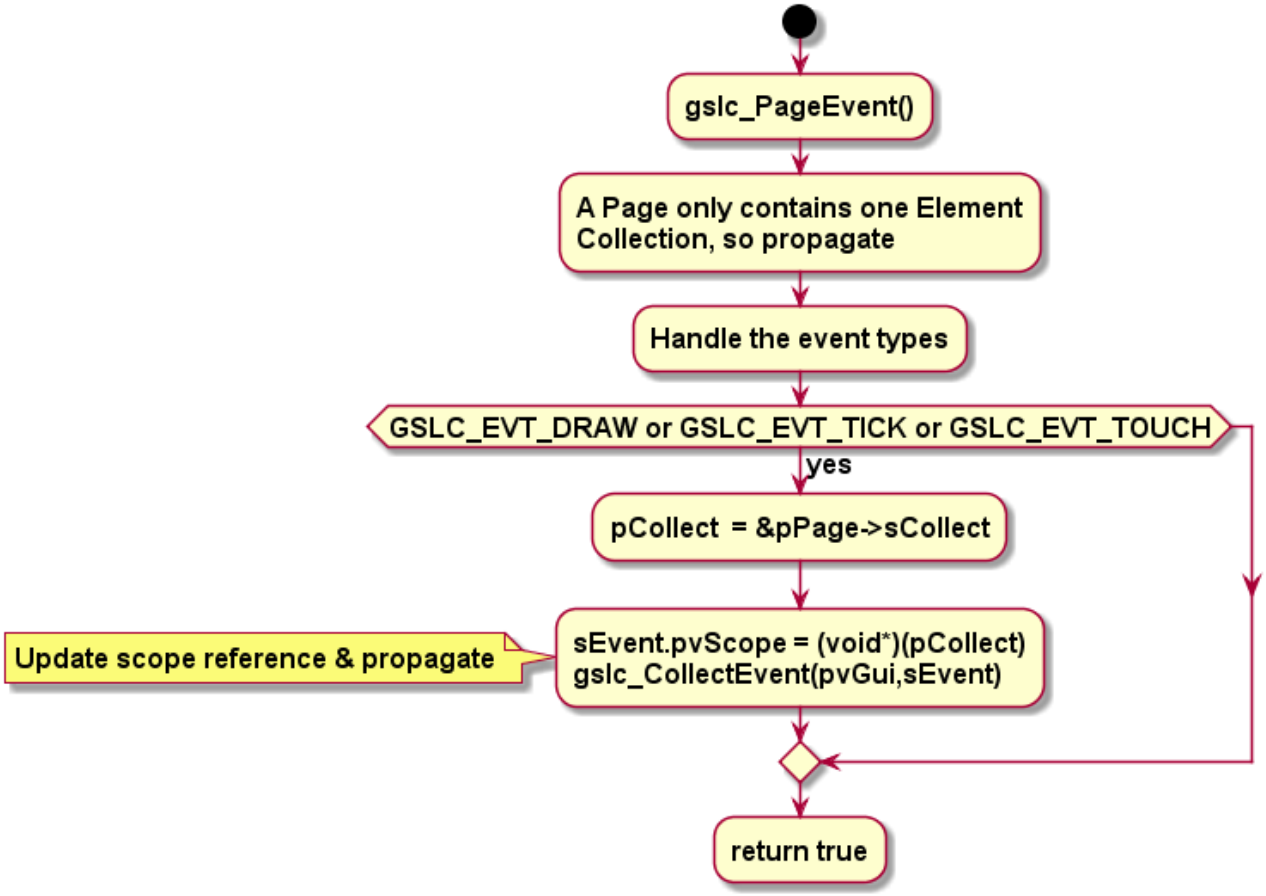

Diagram 5.5 gslc_PageEvent() Flow

5.5.1 gslc_CollectEvent()

```
bool gslc_CollectEvent(void* pvGui,gslc_tsEvent sEvent);
```

Common event handler function for an element collection.

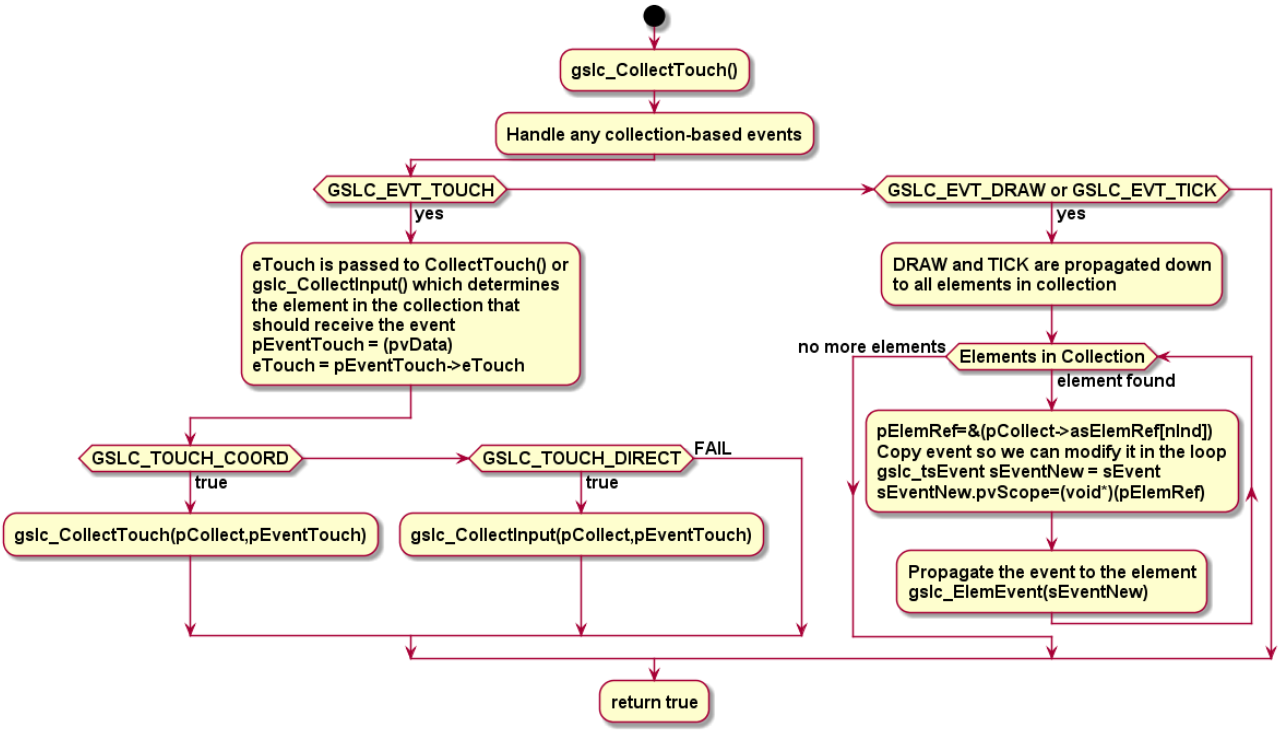

Diagram 5.5.1 gslc_CollectEvent() Flow

5.5.1.1 gslc_CollectTouch()

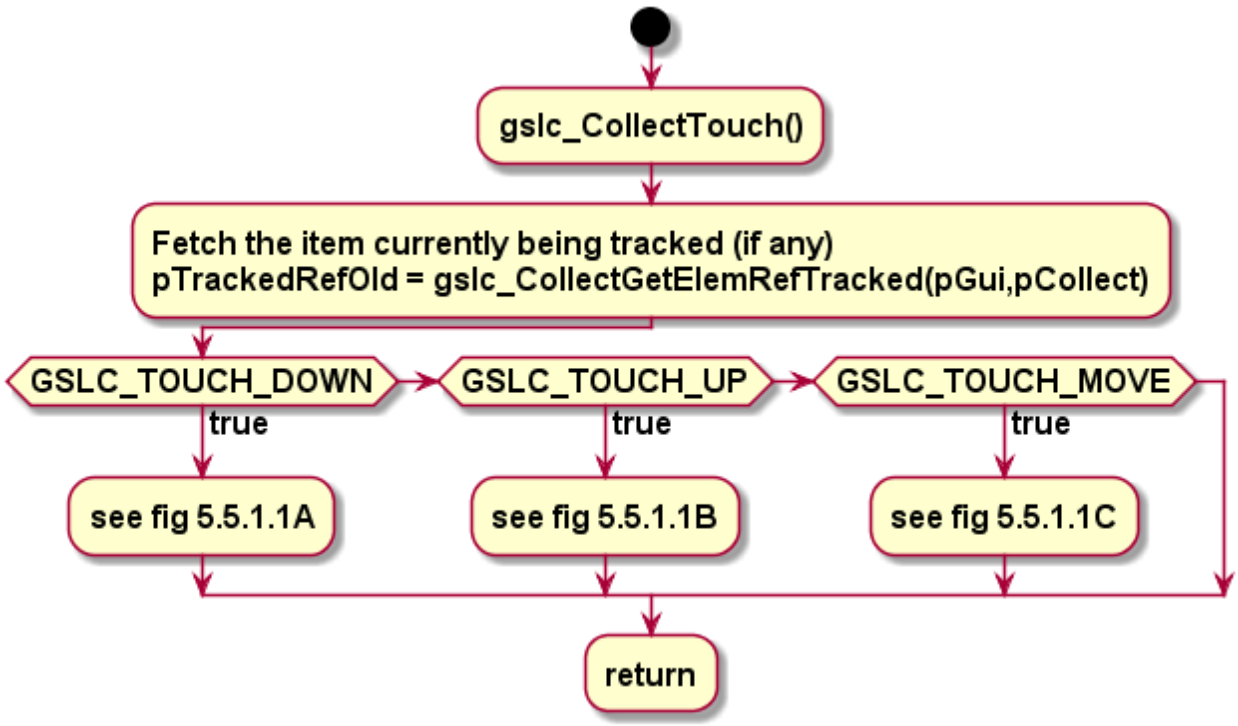

Diagram 5.5.1.1 gslc_CollectTouch() Flow

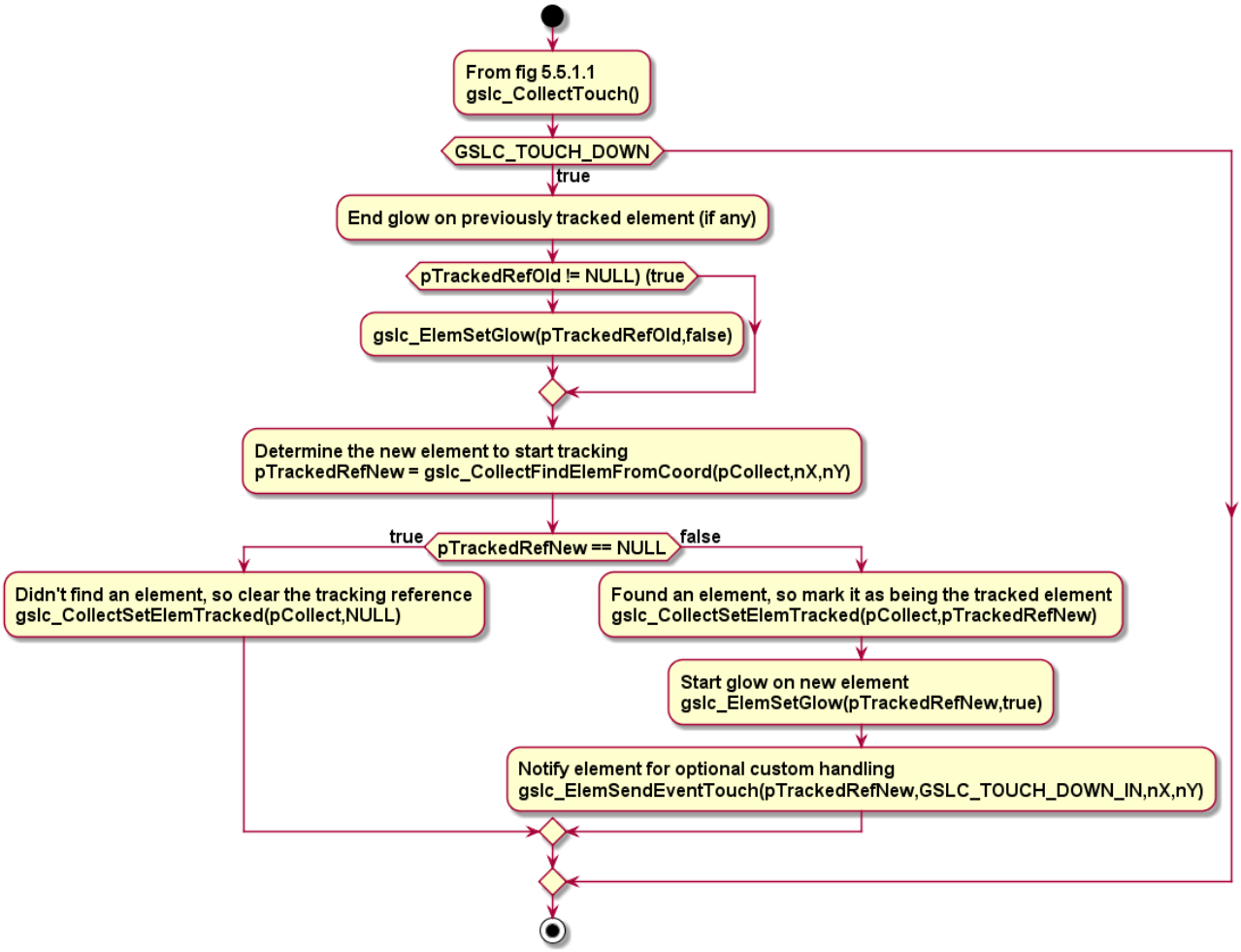

Diagram 5.5.1.1A GSLC_TOUCH_DOWN Flow

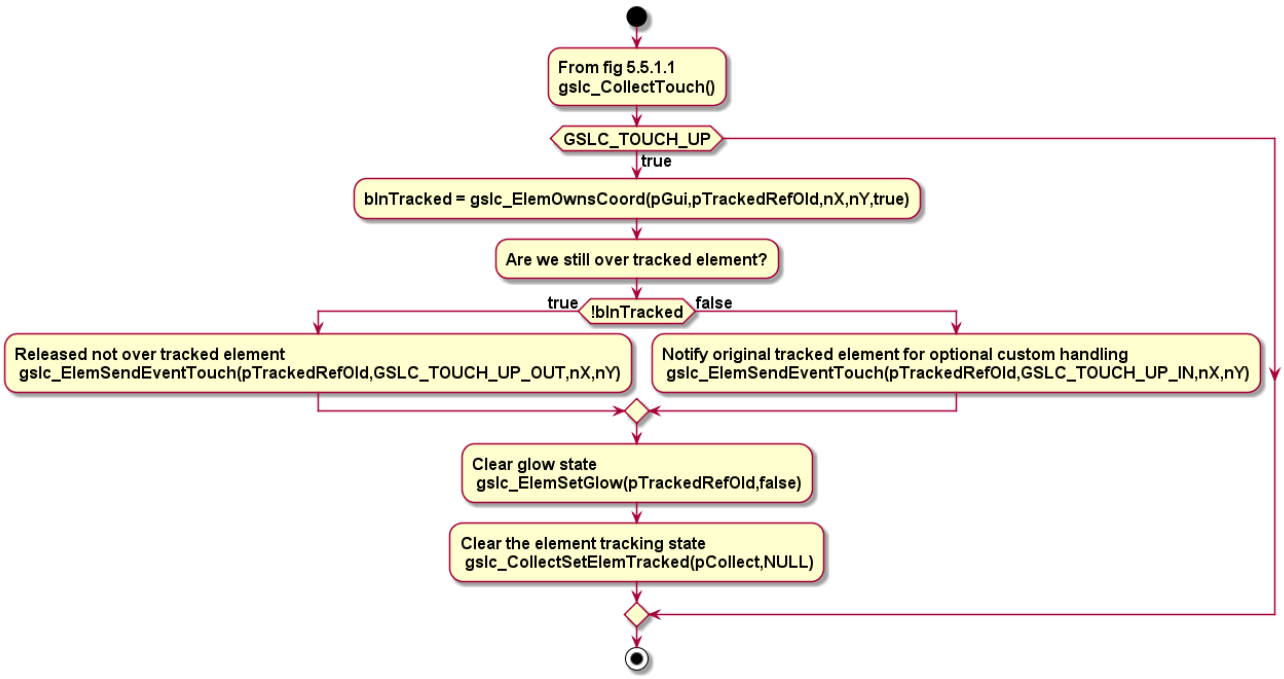

Diagram 5.5.1.1B GSLC_TOUCH_UP Flow

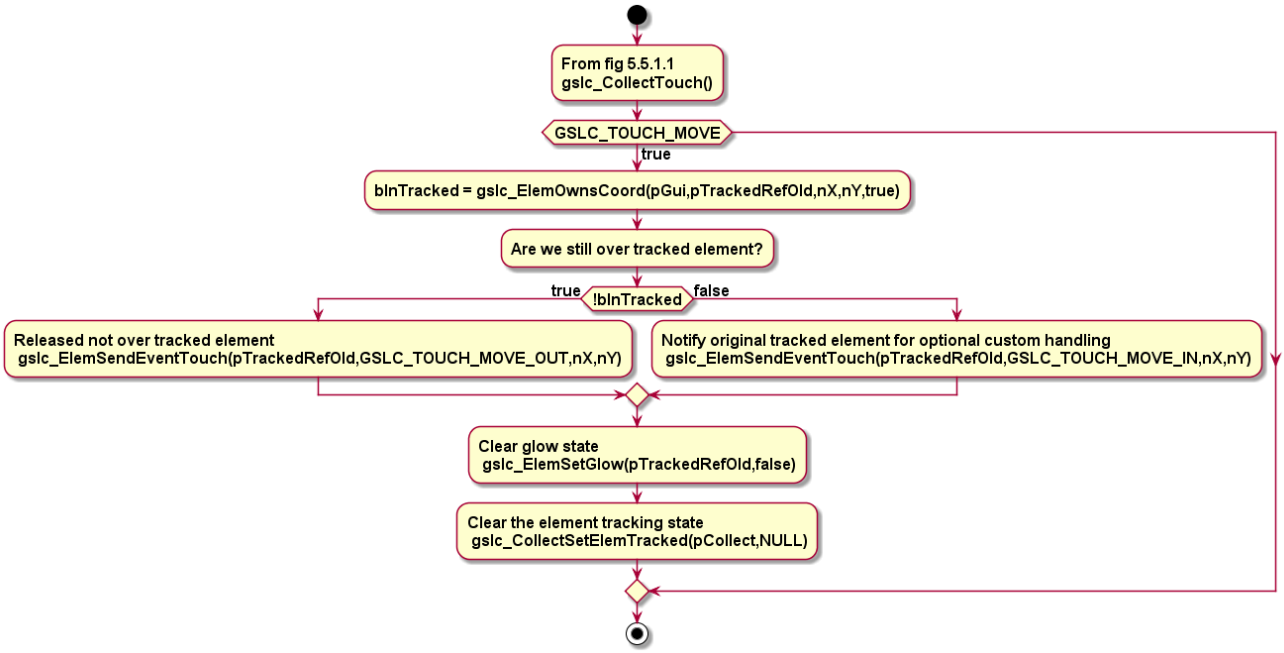

Diagram 5.5.1.1C GSLC_TOUCH_MOVE Flow

5.5.1.2 gslc_CollectInput()

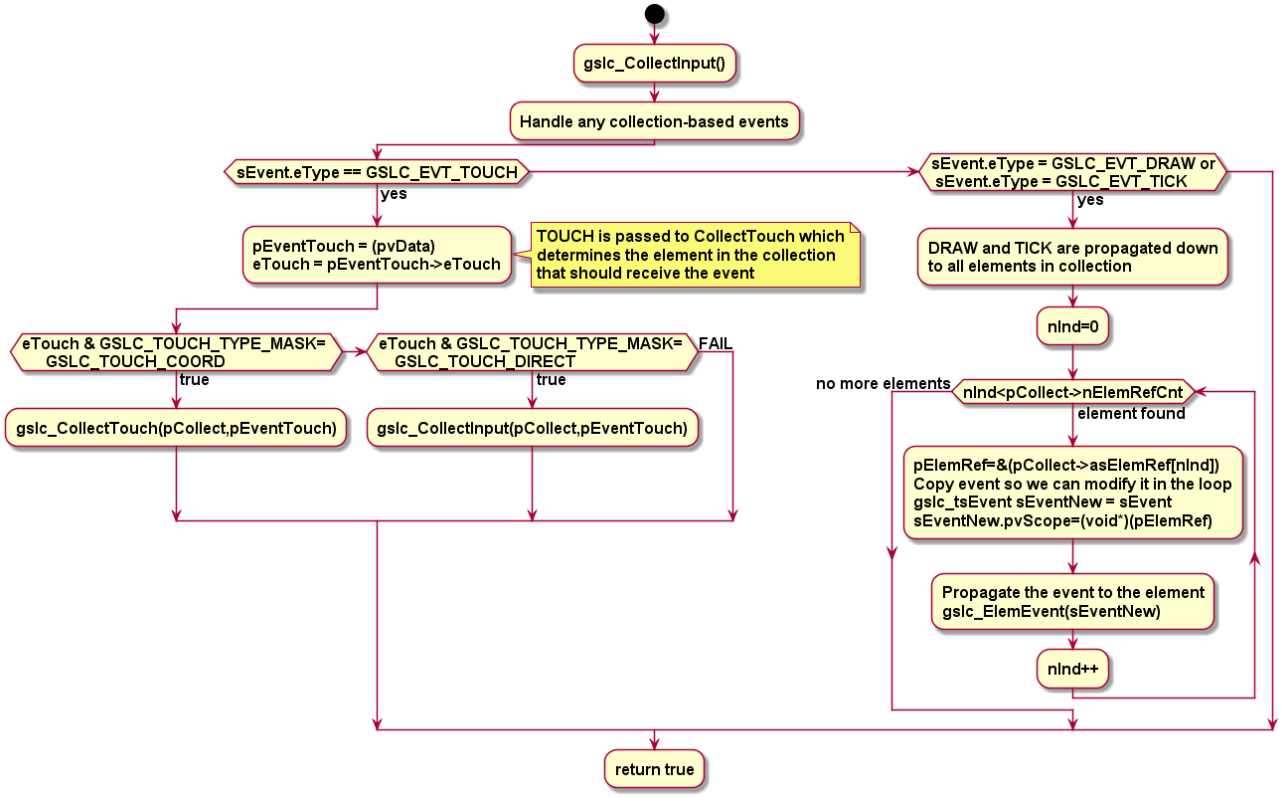

Diagram 5.5.1.2 gslc_CollectInput() Flow

5.5.1.3 gslc_ElemSendEventTouch()

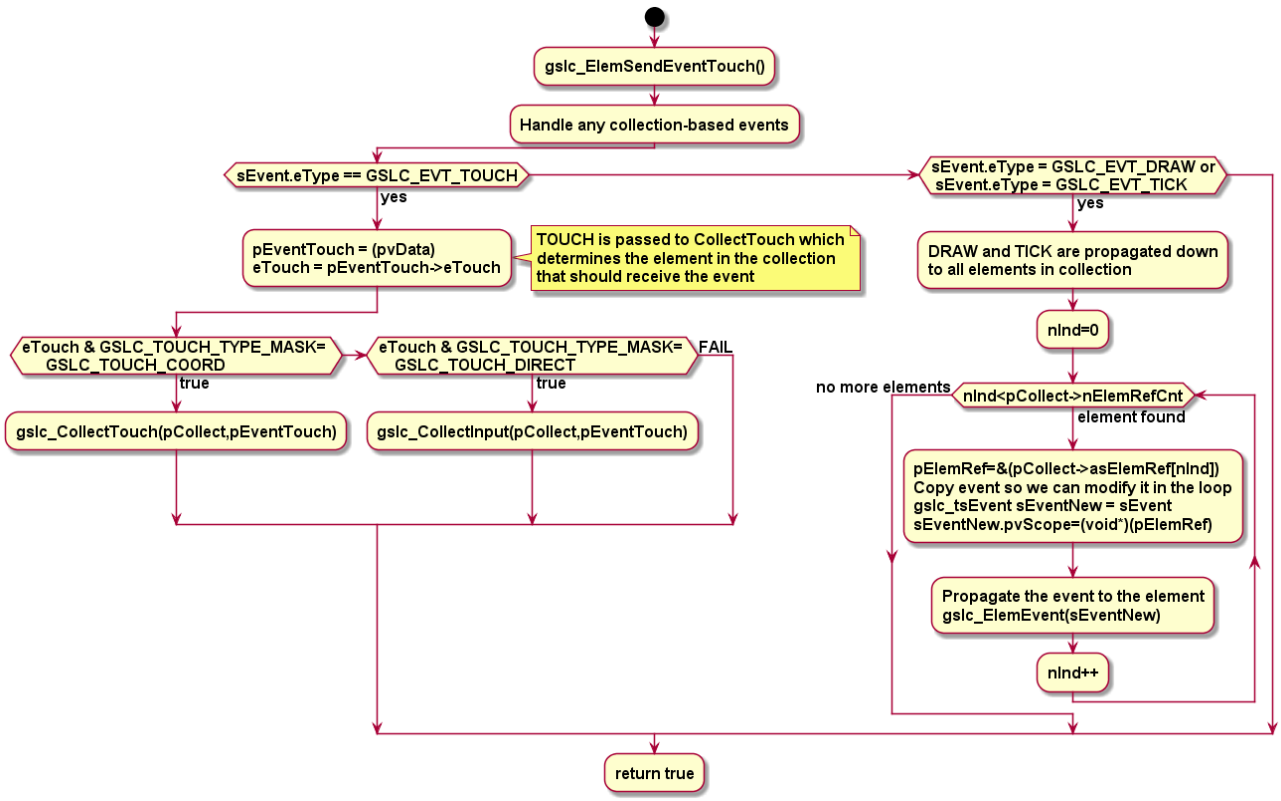

Diagram 5.5.1.3 gslc_ElemSendEventTouch() Flow

5.6 gslc_PageRedrawGo()

Redraw the active page

- If the page has been marked as needing redraw, then all elements are rendered
- If the page has not been marked as needing redraw then onlythe elements that have been marked as needing redraw are rendered.

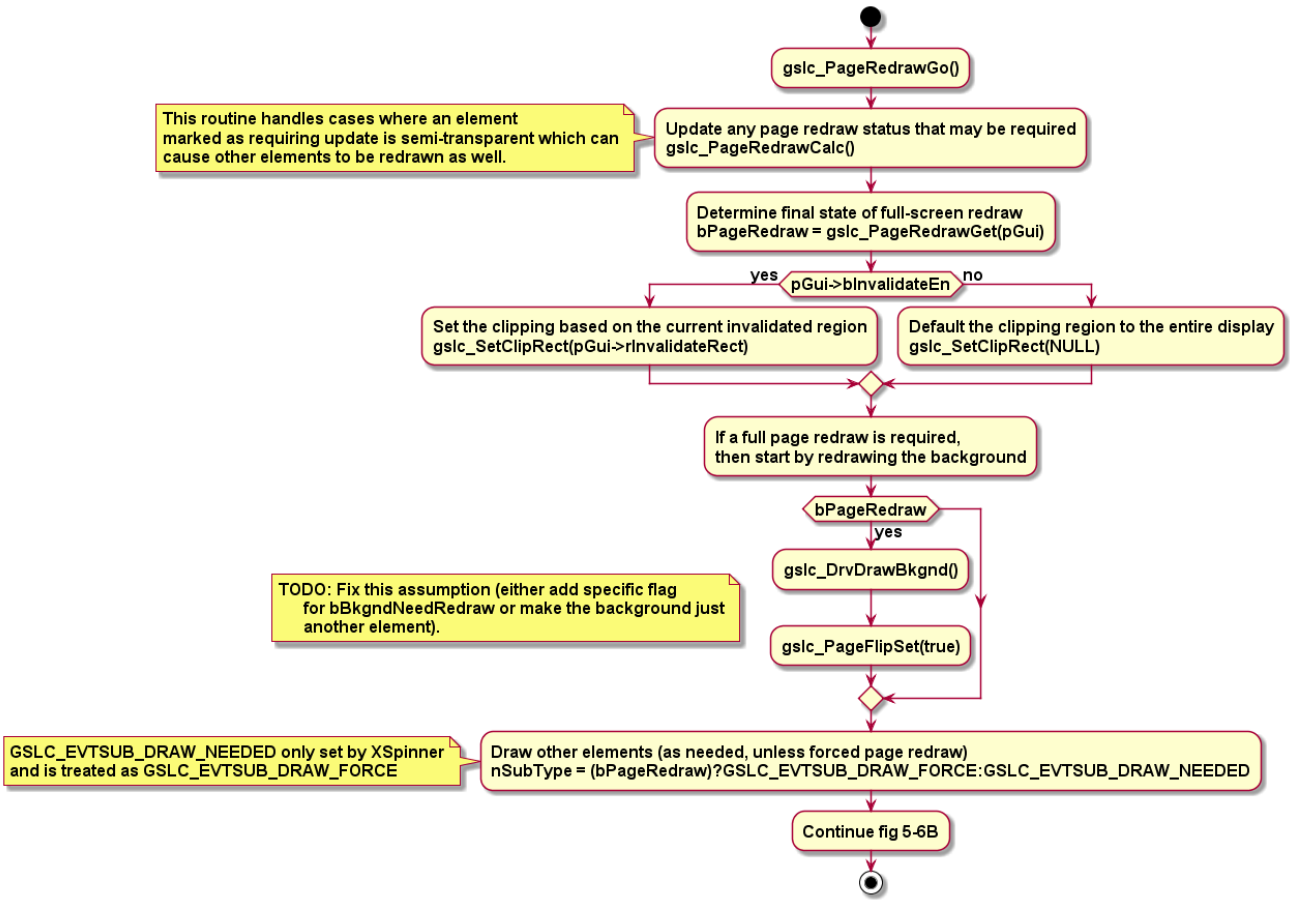

Diagram 5.6A gslc_PageRedrawGo() Flow

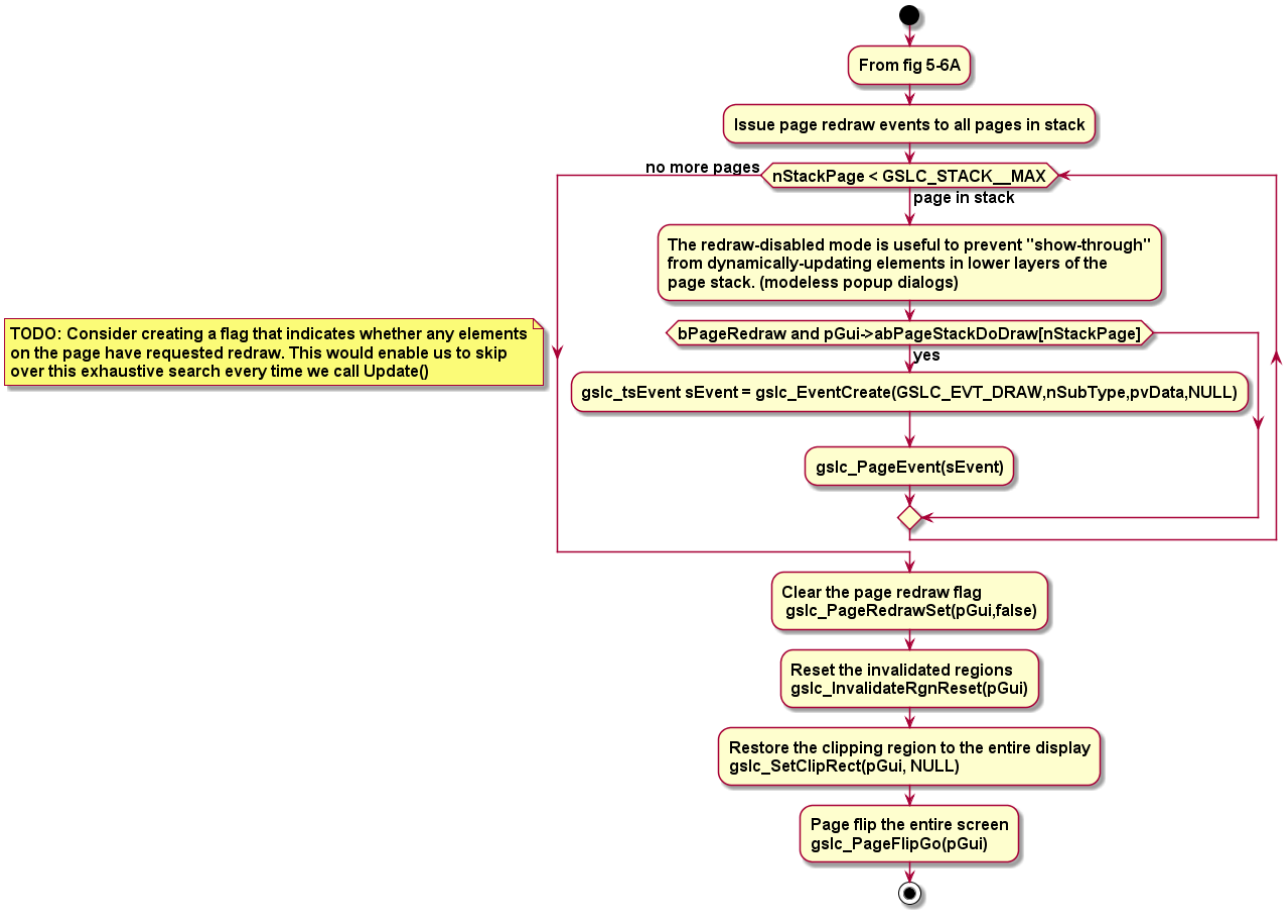

Diagram 5.6B gslc_PageRedrawGo() Flow

Note that you will see various calls to deal with Flip pages. This is to support double buffering of displays that support this feature. Very few drivers support this and since the names of these routines describes them sufficiently there is no reason to go into a deep explanation or flowchart.

5.6.1 gslc_PageRedrawCalc()

Check the redraw flag on all elements on the current page and update the redraw status if additional redraws are required (or the entire page should be marked as requiring redraw).

- The typical case for this being required is when an element requires redraw but it is marked as being transparent. Therefore, the lower level elements should be redrawn.
- For now, just mark the entire page as requiring redraw.

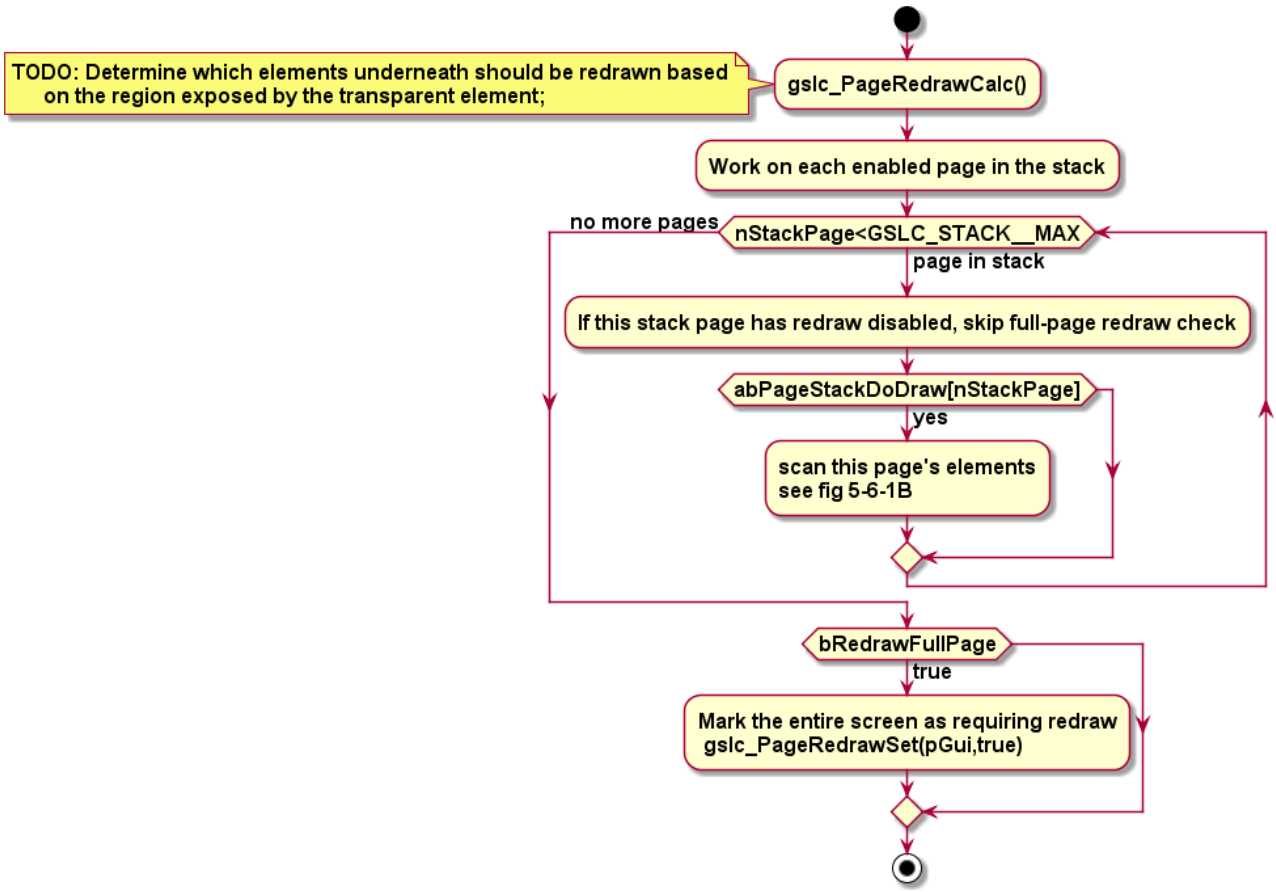

Diagram 5.6.1A gslc_PageRedrawCalc() Flow

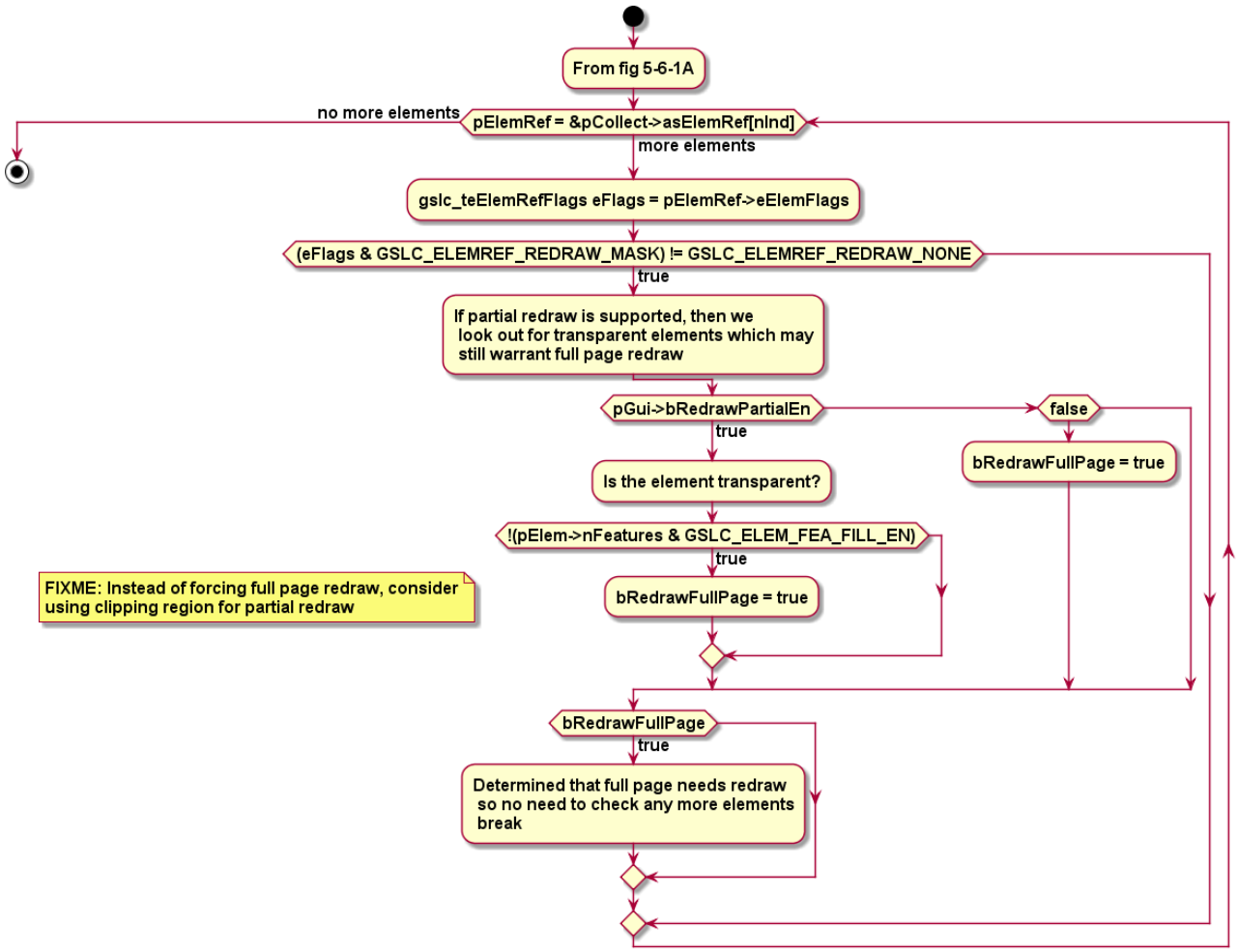

Diagram 5.6.1B gslc_PageRedrawCalc() Flow

5.6.2 gslc_SetClipRect()

Update the drawing clip rectangle.

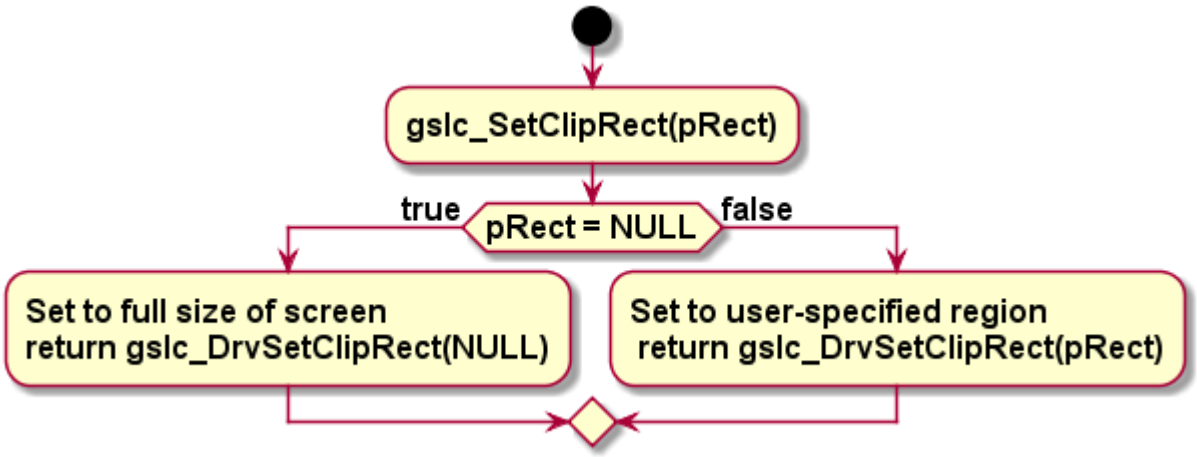

Diagram 5.6.2 gslc_SetClipRect() Flow

5.6.3 gslc_PageRedrawSet()

Adjust the flag that indicates whether the entire page requires a redraw.

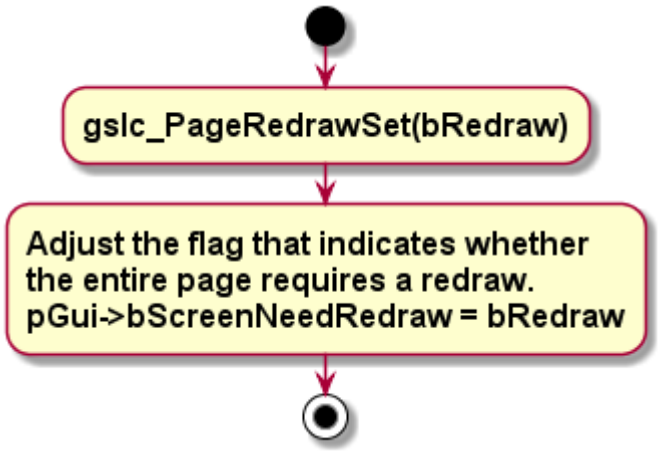

Diagram 5.6.3 gslc_PageRedrawSet() Flow

5.6.4 gslc_InvalidateRgnReset()

Clear our regions.

Diagram 5.6.4 gslc_InvalidateRgnReset() Flow

Chapter 6 Fonts

Chapter 7 Special Features

7.1 Page Layers

7.1.1 Pages, switching between

7.1.2 Popups

7.2 Images

7.2.1 Image Format Support

7.2.2 gslc_ElemCreateImg()

7.2.3 gslc_ElemCreateBtnImg()

7.3 Elements in FLASH

7.4 Element aliases

Chapter 8 Extending GUIslice's UI

8.1 Singular vs Compound Elements

8.2 Modify existing elements

8.3 Creating new elements

8.4 Elements in FLASH

Appendix

Variable naming conventions