

**Politecnico di Torino**

[cas.polito.it](http://www.cas.polito.it)

**LandTiger Emulator API Documentation**

|  |  |
| --- | --- |
| **Professors**  Bernardi Paolo  Sanchez Sanchez Edgar Ernesto | **Authors**  Gabriele Filipponi  Federico Filipponi |

Contents

[Introduction 4](#_Toc65662277)

[Implementing your own module DLL 5](#_Toc65662278)

[LandTiger Emulator Function Description 7](#_Toc65662279)

[DefineAllSFR 8](#_Toc65662280)

[DefineAllVTREG 9](#_Toc65662281)

[DefineAllInterrupts 10](#_Toc65662282)

[DefineAllWatches 11](#_Toc65662283)

[ResetPeripheral 12](#_Toc65662284)

[DefineAllKeys 13](#_Toc65662285)

[DefineAllComponents 14](#_Toc65662286)

[DefineAllDialogPages 15](#_Toc65662287)

[UpdateUI 16](#_Toc65662288)

[ClosingUI 17](#_Toc65662289)

[ComponentsLoaded 18](#_Toc65662290)

[getAgsi 19](#_Toc65662291)

[DefineKey 20](#_Toc65662292)

[DefineNewComponent 21](#_Toc65662293)

[DefinePageDialog 22](#_Toc65662294)

[getBoardChipWithID 23](#_Toc65662295)

[isDialogForeMost 24](#_Toc65662296)

[isBoardPowered 25](#_Toc65662297)

[LTBase 26](#_Toc65662298)

[LTLed Interface Example 28](#_Toc65662299)

[LTLed Implementation Example 29](#_Toc65662300)

[Bundle 32](#_Toc65662301)

[Bundle LoadAfterID 33](#_Toc65662302)

[Re-using Components 34](#_Toc65662303)

# Introduction

This project is based on the µVision Debugger, which supports a simulator interface, called AGSI, for implementing user-defined peripherals. **The documentation needed to understand how Keil handles it and how modules works can be found** [**here**](https://www.keil.com/appnotes/files/apnt_154.pdf).

The LandTiger Emulator is a fixed window, loaded by Keil, which offers a way for dynamically loading modules into a single page. The available modules are:

* Buttons.dll, for the switches KEY1, KEY2, INT0, RESET.
* GLCD.dll, for the LCD driver.
* TP.dll, for the TouchPanel driver.
* Joystick.dll, for the Joystick driver.
* Jumpers.dll, which exports the jumper class for other components.
* LandTigerBase.dll, which is the base image of the board.
* Leds.dll, which exports the led class for other components and for the LD4 through LD11 leds.
* Potmeter.dll, for the potmeter driver.
* Power.dll, for the power switch.
* Speaker.dll, for the speaker driver.

# Implementing your own DLL module

In order to develop a DLL module you should perform the following steps:

* Install µVision Debugger on your PC.
* Download and install the LandTiger Emulator following your Professor instructions.
* Download and install the [Visual Studio Installer](https://visualstudio.microsoft.com/it/downloads/).
* Open the installer and on the Workloads tab select “Desktop Development with *C++*”, under the Individual Components tab select “*MFC for the most recent Build Tools v142 (x86 and x64)*”.
* When the installation is completed, open Visual Studio and create a new project using the template “*MFC dynamic link library*”.
* Now right click on your project from the Solution Explorer and select those properties:
  + *“General C++ Language”* select “*Standard C++ 17 ISO (/std:c++ 17)*”.
  + “Debug *Command*” write “*<your\_Keil\_installation\_path\UV4\UV4.exe>*” (by default it should be “*C:\Keil\_v5\UV4\UV4.exe*”).
  + “*C/C++ General Additional inclusion directories*” write “<you\_Keil\_installation\_path\ARM\BIN\DOCS\Documentation\Libs>” (by default is should be “C:\Keil\_v5\ARM\BIN\DOCS\Documentation\Libs”).
  + “*C/C++ General Code Generation*” disable “Spectre Mitigation”.
  + “*Linker Input* ” add *“<you\_Keil\_installation\_path\ARM\BIN\DOCS\Documentation\Libs\LandTiger.lib>*” (by default is should be “*C:\Keil\_v5\ARM\BIN\DOCS\Documentation\Libs\LandTiger.lib*”).
  + This can be skipped as it is only for convenience, “*Compilation Events Post-Compilation Event Command Line*” add “*copy $(OutDir)$(ProjectName).dll <your\_Keil\_instllation\_path\ARM\BIN\DOCS\<you\_module\_dir> \$(ProjectName).dll*”.

# LandTiger Emulator Function Description

There are required functions that are directly called from the LandTiger Emulator:

* BOOL DefineAllSFR(void);
* BOOL DefineAllVTREG(void);
* BOOL DefineAllInterrupts(void);
* BOOL DefineAllWatches(void);
* BOOL ResetPeripheral(void);
* BOOL DefineAllKeys(void);
* BOOL DefineAllComponents(CWnd \* dialog, CRect board\_rect, Bundle\* bundle);
* BOOL DefineAllDialogPages(CWnd \* parent);
* void UpdateUI(void);

and optional functions:

* void ClosingUI(void);
* void ComponentsLoaded(void);

There are functions to initialize your module and can be called **only during initialization**:

* BOOL DefineKey(KeyRAI\* key);
* BOOL DefineNewComponent(LTBase \*component, CWnd\* parent, CRect rect, Bundle\* bundle);
* BOOL DefinePageDialog(CDialog\* page, CWnd\* parent);

There are functions to retrieve information from the Emulator:

* const struct AGSIFUNCS\* getAgsi(void);
* LTBase \* getBoardChipWithID(const std::string & ID);
* BOOL isDialogForeMost(void);
* BOOL isBoardPowered(void);

# DefineAllSFR

**Summary:**

extern "C" LTEXPORT BOOL DefineAllSFR(void);

**Parameters:**

None

**Return value:**

This function should return TRUE() if completed successfully or FALSE() if an error occurred.

**Description:**

This function is called by the Emulator and in its body you should define all of yours SFR.

**Note:**

None

**Example:**

extern "C" LTEXPORT BOOL DefineAllSFR(void)

{

//Define SFRs here using DefineSFR

}

# DefineAllVTREG

**Summary:**

extern "C" LTEXPORT BOOL DefineAllVTREG(void);

**Parameters:**

None

**Return value:**

This function should return TRUE() if completed successfully or FALSE() if an error occurred.

**Description:**

This function is called by the Emulator and in its body you should define all of yours VTREG.

**Note:**

None

**Example:**

std::vector<struct vtrlist> VTREG =

{

//Define VTRs here

};

extern "C" LTEXPORT BOOL DefineAllVTREG(void)

{

BOOL ret = TRUE;

UINT i;

for (i = 0; i < VTREG.size(); i++) {

VTREG[i].hVTR = getAgsi()->DefineVTR(VTREG[i].pName, VTREG[i].Type, VTREG[i].Value);

if (!VTREG[i].hVTR) ret = FALSE;

}

return ret;

}

# DefineAllInterrupts

**Summary:**

extern "C" LTEXPORT BOOL DefineAllInterrupts(void);

**Parameters:**

None

**Return value:**

This function should return TRUE() if completed successfully or FALSE() if an error occurred.

**Description:**

This function is called by the Emulator and in its body you should define all of yours Interrupts.

**Note:**

None

**Example:**

extern "C" LTEXPORT BOOL DefineAllInterrupts(void)

{

//Define Interrupts

}

# DefineAllWatches

**Summary:**

extern "C" LTEXPORT BOOL DefineAllWatches(void);

**Parameters:**

None

**Return value:**

This function should return TRUE() if completed successfully or FALSE() if an error occurred.

**Description:**

This function is called by the Emulator and in its body you should define all of yours Watches.

**Note:**

None

**Example:**

extern "C" LTEXPORT BOOL DefineAllWatches(void)

{

//Define All Watches here

}

# ResetPeripheral

**Summary:**

extern "C" LTEXPORT BOOL ResetPeripheral(void);

**Parameters:**

None

**Return value:**

This function should return TRUE() if completed successfully or FALSE() if an error occurred.

**Description:**

This function is called by the Emulator when the debugger is resetted (RESET button).

**Note:**

None

**Example:**

extern "C" LTEXPORT BOOL ResetPeripheral (void)

{

//Take actions here

}

# DefineAllKeys

**Summary:**

extern "C" LTEXPORT BOOL DefineAllKeys(void);

**Parameters:**

None

**Return value:**

This function should return TRUE() if completed successfully or FALSE() if an error occurred.

**Description:**

This function is called by the Emulator during initialization to define key handlers.

**Note:**

None

**Example:**

extern "C" LTEXPORT BOOL DefineAllKeys (void)

{

//Define Keys here

}

# DefineAllComponents

**Summary:**

extern "C" LTEXPORT BOOL DefineAllComponents(CWnd \* dialog, CRect board\_rect, Bundle\* bundle);

**Parameters:**

CWnd \*dialog this is the base dialog box for the emulator (you can ignore it).

CRect board\_rect this is the rectangle of the board (not the dialog).

Bundle \*bundle the bundle associated to the call made by the emulator. This parameter needs to be used with the “DefineNewComponent” function.

**Return value:**

This function should return TRUE() if completed successfully or FALSE() if an error occurred.

**Description:**

This function is called by the Emulator during initialization to define the components of your module.

**Note:**

None

**Example:**

extern "C" LTEXPORT BOOL DefineAllComponents (void)

{

//Define Components here

}

# DefineAllDialogPages

**Summary:**

extern "C" LTEXPORT BOOL DefineAllDialogPages(CWnd \* parent);

**Parameters:**

CWnd \*parent this is the parent of your defined page dialogs. This needs to be passed as argument when using “DefinePageDialog”.

**Return value:**

This function should return TRUE() if completed successfully or FALSE() if an error occurred.

**Description:**

This function is called by the Emulator to define additional page dialogs when the user right click on any components of your module.

**Note:**

None

**Example:**

extern "C" LTEXPORT BOOL DefineAllDialogPages(void)

{

//Define Page Dialogs here

}

# UpdateUI

**Summary:**

extern "C" LTEXPORT void UpdateUI(void);

**Parameters:**

None

**Return value:**

None

**Description:**

This function is called by the Emulator when the UI needs to be updated. Please NOTE that for performance reasoning this function is NOT called on every change of your monitored watches.

**Note:**

None

**Example:**

extern "C" LTEXPORT void UpdateUI(void)

{

//Update your UI here

}

# ClosingUI

**Summary:**

extern "C" LTEXPORT void ClosingUI(void);

**Parameters:**

None

**Return value:**

None

**Description:**

This function is called by the Emulator when the UI is closed (not on emulation end). If you need to release UI elements you can do it here.

**Note:**

None

**Example:**

extern "C" LTEXPORT void ClosingUI(void)

{

//Update your UI here

}

# ComponentsLoaded

**Summary:**

extern "C" LTEXPORT void ComponentsLoaded(void);

**Parameters:**

None

**Return value:**

None

**Description:**

This function is called by the Emulator when all the modules are loaded. Hence can be useful to check if the board is powered to update your components.

**Note:**

None

**Example:**

extern "C" LTEXPORT void ComponentsLoaded(void)

{

//Do something here

}

# getAgsi

**Summary:**

extern "C" LTIMPORT const struct AGSIFUNCS\* getAgsi(void);

**Parameters:**

None

**Return value:**

This functions should return AGSI struct pointer.

**Description:**

None.

**Note:**

None

**Example:**

getAgsi()->DefineSFR("DACR", DACR, AGSILONG, 0);

getAgsi()->SetWatchOnSFR(DACR, callback, AGSIWRITE);

# DefineKey

**Summary:**

extern "C" LTIMPORT BOOL DefineKey(KeyRAI\* key);

**Parameters:**

KeyRAI \*key key handler to define.

**Return value:**

This function returns TRUE() if completed successfully or FALSE() if an error occurred.

**Description:**

Define a new key handler.

**Note:**

The KeyRAI class follows the RAII pattern.

**Example:**

extern "C" LTEXPORT BOOL DefineAllKeys(void)

{

BOOL ret = TRUE;

ret &= DefineKey(new KeyRAI('1', [](UINT key, BOOL state)

{

//Do something here

}

));

return ret;

}

# DefineNewComponent

**Summary:**

extern "C" LTIMPORT BOOL DefineNewComponent(LTBase \*component, CWnd\* parent, CRect rect, Bundle\* bundle);

**Parameters:**

LTBase \*component the component pointer to be defined.

CWnd \*parent the dialog box (ignore this).

CRect rect the rect of the component.

Bundle \*bundle bundle parameter.

**Return value:**

This function returns TRUE() if completed successfully or FALSE() if an error occurred.

**Description:**

Define a new component for your module.

**Note:**

Every component inherits from the abstract class LTBase, which is discussed later.

**Example:**

extern "C" LTEXPORT BOOL DefineAllComponents(CWnd \* dialog, CRect board\_rect, Bundle \* bundle)

{

LTBase \*parent = getBoardChipWithID("LandTiger");

return DefineNewComponent(new MyAwesomeComponent("MAC"), parent, CRect(50, 50, 50 + 10, 50 + 20), bundle);

}

# DefinePageDialog

**Summary:**

extern "C" LTIMPORT BOOL DefinePageDialog(CDialog\* page, CWnd\* parent);

**Parameters:**

CDialog \*page the page pointer to be defined.

CWnd \*parent the dialog parent box.

**Return value:**

This function returns TRUE() if completed successfully or FALSE() if an error occurred.

**Description:**

Define a new page dialog when the user right click on your module.

**Note:**

The title of the Dialog will be the title of the Tab-Page-Dialog.

**Example:**

extern "C" LTEXPORT BOOL DefineAllDialogPages(CWnd \* parent)

{

return DefinePageDialog(new MyAwesomePage(parent), parent);

}

# getBoardChipWithID

**Summary:**

extern "C" LTIMPORT LTBase \* getBoardChipWithID(const std::string & ID);

**Parameters:**

const std::string &ID the ID of the component to retrieve.

**Return value:**

This function returns the component pointer if found, otherwise nullptr.

**Description:**

None

**Note:**

You can get a component ID by right-clicking it during the emulation. The ID will be the title of the window.

**Example:**

LTLed \*ld4 = static\_cast<LTLed\*>getBoardChipWithID("LD4");

# isDialogForeMost

**Summary:**

extern "C" LTIMPORT BOOL isDialogForeMost(void);

**Parameters:**

None

**Return value:**

This function returns the TRUE(1) if the Emulator is the foremost window, FALSE(0) otherwise.

**Description:**

None

**Note:**

None

**Example:**

if (isDialogForeMost())

{

//Do something here

}

# isBoardPowered

**Summary:**

extern "C" LTIMPORT BOOL isBoardPowered(void);

**Parameters:**

None

**Return value:**

This function returns TRUE(1) if the power is ON, FALSE(0) otherwise.

**Description:**

None

**Note:**

For power here we intend the “emulated” power (the power switch).

**Example:**

if (isDialogForeMost())

{

//Do something here

}

# LTBase

LTBase is the abstract base class of every component on the board. It was designed to offer re-usability of component classes if necessary. The important point is that every component can be seen as a layer where we draw into. The shape we want is defined, and drew, on the “OnPaint()” method, which is called by the MFC library.

Its interface is:

class LTIMPORT LTBase : public CWnd

{

private:

DECLARE\_DYNAMIC(LTBase)

BOOL enabled;

std::string bundleID;

ChipDialog\* info;

LTBase() = delete;

std::string ID;

protected:

DECLARE\_MESSAGE\_MAP();

static INT OffsetForRotatedRectOfWidth(double width);

ChipOrientation orientation;

void OnDraw(CDC& dc, BaseType type = BaseType::Rect, BOOL draw\_chip = TRUE, COLORREF back\_color = CHIP\_COLOR);

BOOL loadSettings(float& value);

BOOL saveSettings(float value);

public:

LTBase(const std::string& ID, ChipOrientation orientation);

void SetBundleID(const std::string& ID);

std::string GetID(void) const;

virtual ~LTBase();

virtual void setEnabled(BOOL enabled);

virtual BOOL isEnabled() const;

virtual void setOrientation(ChipOrientation orientation) = 0;

afx\_msg void OnRButtonDown(UINT nFlags, CPoint point);

};

The class offers a method to draw the background of a component with ease:

void OnDraw(CDC& dc, BaseType type = BaseType::Rect, BOOL draw\_chip = TRUE, COLORREF back\_color = CHIP\_COLOR);

Where the arguments are:

CDC& dc CDC object of the drawing class.

BaseType type is the type of background that you want to draw. You can choose between “Rect”, “Rotated\_Rect” (45 degree) and “Ellipse”.

BOOL draw\_chip a flag to draw the metallic chip.

COLORREF back\_color the background color to fill the rectangle.

# LTLed Interface Example

“LTLed” is an example of a component class that inherits from “LTBase”. It represents a Led component. Here is the interface:

class LTEXPORT LTLed : public LTBase

{

private:

DECLARE\_DYNAMIC(LTLed);

CRect diode;

LTLed() = delete;

protected:

DECLARE\_MESSAGE\_MAP();

BOOL on;

void OnDraw(CPaintDC& dc);

BOOL init;

public:

static CSize Size(ChipOrientation orientation = ChipOrientation::TOP);

LTLed(const std::string& ID, BOOL on = FALSE, ChipOrientation orientation = ChipOrientation::TOP);

virtual ~LTLed();

void setON(BOOL flg);

void setOrientation(ChipOrientation orientation) override;

void setEnabled(BOOL enabled) override;

BOOL isON(void) const;

void toggle(void);

afx\_msg void OnPaint();

afx\_msg BOOL OnEraseBkgnd(CDC\* pDC);

};

Following the re-usability thoughts mentioned earlier, every class that I wrote follows the “static CSize Size(…)” pattern, but you can ignore it, as it is easier to create new components:

extern "C" LTEXPORT BOOL DefineAllComponents(CWnd \* dialog, CRect board\_rect, Bundle \* bundle)

{

BOOL ret = TRUE;

INT x = 443;

INT y = 518;

INT w = LTLed::Size().cx;

INT h = LTLed::Size().cy;

LTBase\* parent = getBoardChipWithID("LandTiger");

ret &= DefineNewComponent(new LTLed("LD4"), parent, CRect(x, y, x + w, y + h), bundle);

return ret;

}

# LTLed Implementation Example

LTLed implementation is pretty self-explanatory if you know a little of WIN32 programming and you read what’s before this chapter.

Here is the implementation:

CSize LTLed::Size(ChipOrientation orientation)

{

if (orientation == ChipOrientation::TOP || orientation == ChipOrientation::BOTTOM)

{

return { 13, 24 };

}

else

{

return { 24, 13 };

}

}

LTLed::LTLed(const std::string& ID, BOOL on, ChipOrientation orientation) : LTBase::LTBase(ID, orientation), on(on), init(FALSE)

{

}

LTLed::~LTLed()

{

}

void LTLed::setON(BOOL flg)

{

if (flg != on)

{

on = flg;

if (init && isEnabled())

{

InvalidateRect(diode, FALSE);

}

}

}

void LTLed::setOrientation(ChipOrientation orientation)

{

if (orientation != this->orientation)

{

this->orientation = orientation;

if (init)

{

Invalidate();

}

}

}

BOOL LTLed::isON(void) const

{

return (on && isEnabled());

}

void LTLed::setEnabled(BOOL enabled)

{

if (enabled != isEnabled())

{

LTBase::setEnabled(enabled);

if (init)

{

InvalidateRect(diode, FALSE);

}

}

}

void LTLed::toggle(void)

{

setON(!on);

}

void LTLed::OnDraw(CPaintDC& dc)

{

CRect rect;

GetClientRect(rect);

rect.DeflateRect(CHIP\_RECT\_DEFLATE, CHIP\_RECT\_DEFLATE);

CBrush g\_brush;

g\_brush.CreateSolidBrush(CHIP\_COLOR);

CBrush\* old\_brush = dc.SelectObject(&g\_brush);

CRect first, second;

switch (orientation)

{

case ChipOrientation::BOTTOM:

case ChipOrientation::TOP:

{

INT dim = (INT)std::floor(rect.Height() / 3);

diode = { rect.left, rect.top + dim, rect.right, rect.top + dim + (rect.Height() - dim \* 2) };

}

break;

case ChipOrientation::RIGHT:

case ChipOrientation::LEFT:

{

INT dim = (INT)std::floor(rect.Width() / 3);

diode = { rect.left + dim, rect.top, rect.left + dim + (rect.Width() - dim \* 2), rect.bottom };

}

break;

default:

throw std::runtime\_error("Invalid orientation.");

break;

}

dc.FillRect(first, &g\_brush);

dc.FillRect(second, &g\_brush);

CBrush d\_brush;

d\_brush.CreateSolidBrush(LED\_COLOR(isON()));

dc.SelectObject(&d\_brush);

dc.FillRect(diode, &d\_brush);

dc.SelectObject(old\_brush);

init = TRUE;

}

IMPLEMENT\_DYNAMIC(LTLed, LTBase)

BEGIN\_MESSAGE\_MAP(LTLed, LTBase)

ON\_WM\_PAINT()

END\_MESSAGE\_MAP()

void LTLed::OnPaint()

{

CPaintDC dc(this);

OnDraw(dc);

}

Probably the hardest part is the drawing part as WIN32 programming is “**goodly divine**”. But with a little practice you’ll learn it fast. Please note that I’m NOT writing anything about it as there are plenty of tutorial and examples online.

# Bundle

In order to work every module must be placed inside of what’s called a “Bundle”. A Bundle is a folder in which a file named “Info.list” is placed. This file lists the Bundle properties that MUST be defined. Here is the properties:

* ID= this defines the ID of your Bundle. NOTE that 2 bundles can’t have the same ID, if that happens an error will be raised.
* Author= this defines the Author of the Bundle.
* Version= this defines the Version of the Bundle.
* Date= this defines the Date of last update of the Bundle.
* Description= this sets the description path file that will be loaded when the user right-click your module. NOTE that this MUST be an HTML file.
* Schematic= this sets the schematic path file that will be loaded when the user right-click on your module. NOTE that this MUST a BMP file.
* Lib= this is MANDATORY to fill, and is the library path file, the .dll, that will be loaded by the Bundle.
* Models= this is the model file that will be loaded when the user right-click your module. It can be EMPTY. The files must be in OBJ format. You can specify multiple Models= (see LandTiger module).
* LoadAfterID= this is MANDATORY to fill and is explained in the next chapter.

# Bundle LoadAfterID

Like the “Model=” property, also this one can appear multiple times. LoadAfterID specifies the order in which the modules will be loaded. The value of this property is one of the IDs of other modules that will be loaded before your module (you can get other module IDs by sneaking at their Info.list files).

Every module which adds new component on the board MUST at least have “**LoadAfterID=LandTigerBase**” in its “Info.list” file to work properly.

Please note that every module that link to other modules (lib file) need to be loaded after them.

For example if you are developing the USB module, then you need to add few jumpers and a led (LD15) this results in your “Info.list” file to have “LoadAfterID” properties:

* LoadAfterID=LandTigerBase
* LoadAfterID=Jumpers
* LoadAfterID=Leds

This means that your module will be loaded after “LandTigerBase”, “Jumpers” and “Leds” module have been loaded.

# Re-using Components

As stated before, components are written to have re-usability in mind. To “share” your component and make it reusable you need to add the “LTEXPORT” macro in your component class’s definition.

class LTEXPORT LTLed : public LTBase

This will create a “.lib” file in your project directory, so that other modules can link to it.

Currently components have been developed in such way for you to add already available components with ease:

* “LTLed”, for the leds
* “LTJumper”, for the jumpers
* “LTButton”, probably not needed

For example to use the Leds component, you have to modify your Visual Studio project settings like follows:

* “*C/C++ General Additional inclusion directories*” write “<you\_Keil\_installation\_path\ARM\BIN\DOCS\LEDS\EXPORT\ >” (by default is should be “C:\Keil\_v5\ARM\BIN\DOCS\ LEDS\EXPORT\”).
* “*Linker Input* ” add *“<you\_Keil\_installation\_path\ARM\BIN\DOCS\LEDS\EXPORT\Leds.lib>*” (by default is should be “*C:\Keil\_v5\ARM\BIN\DOCS\ LEDS\EXPORT\Leds.lib*”).

Also don’t forget to add LoadAfterID=Leds in your “Info.list” Bundle’s file.