This document describes the interface and usage of the FT8xx emulator library.
Table of Contents

1 Introduction ............................................................................. 2
  1.1 Overview .................................................................................. 2
  1.2 Scope ...................................................................................... 2
  1.3 Requirement ............................................................................ 2
  1.4 Limitations .............................................................................. 2

2 FT8xx Emulator Library Introduction ................................. 3
  2.1 FT8xx Emulator Library Interface ...................................... 3
    2.1.1 FT800EMU::SPII2C.begin() ........................................ 4
    2.1.2 FT800EMU::SPII2C.end() ........................................... 4
    2.1.3 FT800EMU::SPII2C.csLow () ...................................... 4
    2.1.4 FT800EMU::SPII2C.csHigh() ...................................... 4
    2.1.5 FT800EMU::SPII2C.transfer() .................................. 5
    2.1.6 FT800EMU::Emulator.run () ...................................... 5

3 Using the FT8xx Emulator library ..................................... 10
  3.1 Start the FT800 emulator .................................................... 10
  3.2 Working with the SPI/I\(^2\)C interface ............................... 10
  3.3 Sample Application adaptation ......................................... 11
  3.4 Build and Run ................................................................. 13

4 Contact Information ........................................................... 14

Appendix A – References ......................................................... 15
  Document References ........................................................... 15
  Acronyms and Abbreviations ................................................. 15

Appendix B – List of Tables & Figures ................................. 16
  List of Tables ........................................................................ 16
  List of Figures ........................................................................ 16

Appendix C – Revision History ............................................. 17
1 Introduction
The FT8xx Emulator is behavior modeling software targeting to run on a PC. It is designed as a high level (behavior level) emulator other than a low level (clock accurate) emulator. It enables the user to evaluate FT8xx features on a PC without hardware.
This document describes the interface of the FT8xx Emulator library and shows one example of how to integrate it into user’s project.
The emulator library is included in the install package of the EVE Screen Editor, which can be found in the section of the following page:

1.1 Overview
The FT8xx emulator has the same SPI interface configuration and memory map as the FT8xx silicon. As such, the user’s application does not need to write a new interface layer for the emulator version.
The FT8xx emulator has been designed for maximum similarity to the real device although there are a few limitations which are mentioned here and in section 1.4.
For touch functionality, the FT8xx emulator requires the mouse of the PC to simulate single touch input. For visual effects, the FT8xx emulator employs the OS specific graphics driver to display the output on the PC monitor.
The emulator supports the full set of display list commands and most of the coprocessor commands.

1.2 Scope
This document covers the FT8xx emulator library interface and introduces its use by an example application. The emulator is intended to be used in conjunction with the FT8xx programming guide and application examples and as such this guide does not include detailed information on the FT8xx e.g. registers, memory map, commands, etc.

1.3 Requirement
Currently, the FT8xx emulator library is built by Microsoft Visual Studio C++ Express version MSVC 2012. Therefore, this version of the MSVC IDE or newer is recommended to be used for compiling the FT8xx application code which will be run on the emulator to ensure compatibility. In addition, the runtime environment “ft8xxemu.dll” and “SDL2.dll” are required to be on Windows to run the emulator project successfully.

1.4 Limitations
The FT8xx emulator does NOT support the following functionality:

1. Power management (Host commands)
2. Screenshot (coprocessor command "cmd_snapshot" has no effect)
3. Coprocessor engine reset
4. Interrupt
5. Registers that reflect hardware properties, e.g., the pressure value of touch and ADC related touch registers
6. Multi-touch operation
2 FT8xx Emulator Library Introduction

2.1 FT8xx Emulator Library Interface

The interface of FT8xx Emulator library is written in C++ and resides in the “FT800EMU::” name space only. Within the “FT800EMU::” name space, there are two modules “SPII2C” and “Emulator” exposing interface. Here is the structure:

<table>
<thead>
<tr>
<th>Name Space</th>
<th>Module Name</th>
<th>API Name</th>
<th>Parameter</th>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT800EMU::</td>
<td>SPII2C</td>
<td>begin</td>
<td>None</td>
<td>None</td>
<td>Initialize the SPI I²C module</td>
</tr>
<tr>
<td>FT800EMU::</td>
<td>SPII2C</td>
<td>end</td>
<td>None</td>
<td>None</td>
<td>De-Initialize the SPI / I²C Module</td>
</tr>
<tr>
<td>FT800EMU::</td>
<td>SPII2C</td>
<td>csLow</td>
<td>None</td>
<td>None</td>
<td>Set the Chip Select pin low to start one SPI transfer</td>
</tr>
<tr>
<td>FT800EMU::</td>
<td>SPII2C</td>
<td>csHigh</td>
<td>None</td>
<td>None</td>
<td>Set the Chip Select pin low to stop one SPI transfer</td>
</tr>
<tr>
<td>FT800EMU::</td>
<td>SPII2C</td>
<td>transfer</td>
<td>One byte</td>
<td>One byte</td>
<td>Make one byte SPI or I²C transaction on SPI or I²C bus</td>
</tr>
<tr>
<td>FT800EMU::</td>
<td>Emulator</td>
<td>run</td>
<td>See 2.1.6</td>
<td>None</td>
<td>Start the FT800 emulator. Never return unless the emulator exits.</td>
</tr>
</tbody>
</table>

Table 1 – FT8xx Emulator library interface structure
2.1.1 FT800EMU::SPII2C.begin()

- **Prototype**
  void begin();

- **Description**
  Initialize the SPI/I\(^2\)C module of the Emulator

- **Return value**
  None

- **Parameter**
  None

2.1.2 FT800EMU::SPII2C.end()

- **Prototype**
  void end();

- **Description**
  De-Initialize the SPI/I\(^2\)C module of the Emulator

- **Return value**
  None

- **Parameter**
  None

2.1.3 FT800EMU::SPII2C.csLow()

- **Prototype**
  void csLow();

- **Description**
  Call this API to start one SPI/I\(^2\)C transfer. It is equivalent to pulling down chip select pin on the SPI/I\(^2\)C bus in the FT8xx hardware. For I\(^2\)C bus, this function is equivalent to start a message with a START.

- **Return value**
  None

- **Parameter**
  None

2.1.4 FT800EMU::SPII2C.csHigh()

- **Prototype**
  void csHigh();

- **Description**
Call this API to end one SPI/I²C transfer. For SPI bus, it is equivalent to pulling chip select pin high on the SPI/ I²C bus in FT8xx hardware. For I²C bus, this function is equivalent to end a message with a STOP.

- **Return value**
  None

- **Parameter**
  None

### 2.1.5 FT800EMU::SPII2C.transfer()

- **Prototype**
  ```
  uint8_t transfer(uint8_t data);
  ```

- **Description**
  Calling this API is to transfer one byte from/to emulator.
  The data to be sent is specified as a parameter, while the data to be received is given as a return value.

- **Return value**
  One byte of data received from the FT8xx emulator if it is read transfer.

- **Parameter**
  One byte of data sent to FT8xx emulator. In case of an SPI read transfer, this byte can be anything.

### 2.1.6 FT800EMU::Emulator.run ()

- **Prototype**
  ```
  void run(const EmulatorParameters &params);
  ```

- **Description**
  Calling this function will start the emulator immediately and the application control is transferred to the emulator. The emulator’s behavior is configured through the parameters which were passed in. The application’s code will be called through two callback functions in the parameter structure. This API shall never return, unless emulator is killed or the application process exists.

- **Return value**
  None

- **Parameters**
  Please check the following code for details about parameter definition.
1) Definition of parameter structure

typedef struct {
   // Microcontroller function called before loop.
   void(*Setup)();
   // Microcontroller continuous loop.
   void(*Loop)();
   // See EmulatorFlags.
   int Flags;
   // Emulator mode
   FT8XXEMU_EmulatorMode Mode;

   // Called after keyboard update.
   // Supplied function can use Keyboard.isKeyDown(FT8XXEMU_KEY_F3)
   // or FT8XXEMU_isKeyDown(FT8XXEMU_KEY_F3) functions.
   void(*Keyboard)();
   // The default mouse pressure, default 0 (maximum).
   // See REG_TOUCH_RZTRESH, etc.
   uint32_t MousePressure;
   // External frequency. See CLK, etc.
   uint32_t ExternalFrequency;

   // Reduce graphics processor threads by specified number, default 0
   // Necessary when doing very heavy work on the MCU or Coprocessor
   uint32_t ReduceGraphicsThreads;

   // Sleep function for MCU thread usage throttle. Defaults to generic system
   sleep
   void(*MCUSleep)(int ms);

   // Replaces the default built-in ROM with a custom ROM from a file.
   // NOTE: String is copied and may be deallocated after call to run(...)
   char *RomFilePath;

   // Replaces the default built-in OTP with a custom OTP from a file.
   // NOTE: String is copied and may be deallocated after call to run(...)
   char *OtpFilePath;

   // Replaces the built-in coprocessor ROM.
   // NOTE: String is copied and may be deallocated after call to run(...)
   char *CoprocessorRomFilePath;

   // Graphics driverless mode
   // Setting this callback means no window will be created, and all
   // rendered graphics will be automatically sent to this function.
   // For enabling touch functionality, the functions
   // Memory.setTouchScreenXY and Memory.resetTouchScreenXY must be
   // called manually from the host application.
   // Built-in keyboard functionality is not supported and must be
   // implemented manually when using this mode.
   // The output parameter is false (0) when the display is turned off.
   // The contents of the buffer pointer are undefined after this
   // function returns.
   // Return false (0) when the application must exit, otherwise return true (1).
   int(*Graphics)(int output, const argb8888 *buffer, uint32_t hsize, uint32_t vsize, FT8XXEMU_FrameFlags flags);

   // Interrupt handler
   // void (*Interrupt)();
// Exception callback
void (*Exception)(const char *message);

// Safe exit
void (*Close)();

} FT8XXEMU_EmulatorParameters;

Figure 1 – Definition of structure “EmulatorParameters”

2) Flags to configure the emulator
The enumerate code sample shown below defines the emulator feature to be run with. To enable specific features, you can "OR" these enumerate and assign the result values to "Flags" field in the parameter structure "EmulatorParameters" above.

typedef enum {
    // enables the keyboard to be used as input (default: on)
    FT8XXEMU_EmulatorEnableKeyboard = 0x01,
    // enables audio (default: on)
    FT8XXEMU_EmulatorEnableAudio = 0x02,
    // enables coprocessor (default: on)
    FT8XXEMU_EmulatorEnableCoprocessor = 0x04,
    // enables mouse as touch (default: on)
    FT8XXEMU_EmulatorEnableMouse = 0x08,
    // enable debug shortkeys (default: on)
    FT8XXEMU_EmulatorEnableDebugShortkeys = 0x10,
    // enable graphics processor multithreading (default: on)
    FT8XXEMU_EmulatorEnableGraphicsMultithread = 0x20,
    // enable dynamic graphics quality degrading by interlacing and dropping frames
    (default: on)
    FT8XXEMU_EmulatorEnableDynamicDegrade = 0x40,
    // enable emulating REG_PWM_DUTY by fading the rendered display to black
    (default: off)
    FT8XXEMU_EmulatorEnableRegPwmDutyEmulation = 0x100,
    // enable usage of touch transformation matrix (default: on)
    FT8XXEMU_EmulatorEnableTouchTransformation = 0x200,
} FT8XXEMU_EmulatorFlags;

Figure 2 – Flags field definition
3) **Typical setting**

For optimal performance, the settings below are recommended.

The callback functions "setup()" and "loop()" **shall be defined** by the user project and they will be called by the emulator. Function "setup()" is assumed to run once by the emulator for initialization purposes. Function "loop()" will be called periodically by the emulator. These two functions ensure the user project is in the context of the emulator. The failure of assigning "setup()" and "loop()" to the emulator will result in no input to the emulator.

Usually, the function “setup()” and “loop()” in users’ project defines the main logic and the display list will be sent to emulator through SPI/I²C interface.

```c
#include "FT_Platform.h"

#ifdef MSVC_FT800EMU
#include <FT_Emulator.h>

FT8XXEMU_EmulatorMode Ft_GpuEmu_Mode();

extern "C" void setup();
extern "C" void loop();
```
```c
ft_int32_t main(ft_int32_t argc, ft_char8_t *argv[])
{
    FT8XXEMU_EmulatorParameters params;

    FT8XXEMU_defaults(FT8XXEMU_VERSION_API, &params, Ft_GpuEmu_Mode());
    params.Flags &= (~FT8XXEMU_EmulatorEnableDynamicDegrade & ~FT8XXEMU_EmulatorEnableRegPwmDutyEmulation);
    params.Setup = setup;
    params.Loop = loop;
    // params.Graphics = graphics;
    FT8XXEMU_run(FT8XXEMU_VERSION_API, &params);
    return 0;
}
```

**Figure 3 – Start FT8xx emulator**
3 Using the FT8xx Emulator library

This chapter will provide an example on how to use the FT8xx emulator in the FT8xx sample application. Users are encouraged to familiarize themselves with the FT8xx sample application (see section Sample App (+EVE2) on the following page) before starting this chapter.


The FT8xx emulator interface is defined in the following files:
“ft800emu_inttypes.h”: the definition for integer type for different platforms.
“ft800emu_spi_i2c.h”: the SPI/ I²C interface declaration
“ft800emu_emulator.h”: the interface to start the emulator

3.1 Start the FT800 emulator

To make use of the FT800 emulator, the user’s project is required to call the API "FT800EMU::Emulator.run" with the specific parameter. The emulator library will be started properly and ready to be accessed through SPI/ I²C interface.

Please see Figure 3 – Start FT8xx emulator.

3.2 Working with the SPI/I²C interface.

The SPI/I²C interface is the control interface of FT800. FT800 emulator library provides the APIs to simulate the same interface. Since FT800 sample application is built in C language, instead of C++, one simple C API wrapper is introduced as below to ease the calling convention.

```c
#ifdef __cplusplus
extern "C"
{
#endif
#ifdef MSVC_FT800EMU
#define BUFFER_OPTIMIZATION
#endif

void Ft_GpuEmu_SPII2C_begin();
void Ft_GpuEmu_SPII2C_csLow();
void Ft_GpuEmu_SPII2C_csHigh();
void Ft_GpuEmu_SPII2C_end();

void Ft_GpuEmu_SPII2C_StartRead(uint32_t addr);
uint8_t Ft_GpuEmu_SPII2C_transfer(uint8_t data);
void Ft_GpuEmu_SPII2C_StartWrite(uint32_t addr);

#ifdef __cplusplus
}
#endif
```

Figure 4 – The C interface API
The implementation is as below:

```c
void Ft_GpuEmu_SPIII2C_begin()
{
    FT800EMU::SPII2C.begin();
}

void Ft_GpuEmu_SPIII2C_csLow()
{
    FT800EMU::SPII2C.csLow();
}

void Ft_GpuEmu_SPIII2C_csHigh()
{
    FT800EMU::SPII2C.csHigh();
}

void Ft_GpuEmu_SPIII2C_end()
{
    FT800EMU::SPII2C.end();
}

uint8_t Ft_GpuEmu_SPIII2C_transfer(uint8_t data)
{
    return FT800EMU::SPII2C.transfer(data);
}

void Ft_GpuEmu_SPIII2C_StartRead(uint32_t addr)
{
    Ft_GpuEmu_SPIII2C_csLow();
    Ft_GpuEmu_SPIII2C_transfer(((addr >> 16) & 0xFF) | 0x80);
    Ft_GpuEmu_SPIII2C_transfer((addr >> 8) & 0xFF);
    Ft_GpuEmu_SPIII2C_transfer(addr & 0xFF);
    Ft_GpuEmu_SPIII2C_transfer(0); //Dummy Read Byte
}

void Ft_GpuEmu_SPIII2C_StartWrite(uint32_t addr)
{
    Ft_GpuEmu_SPIII2C_csLow();
    Ft_GpuEmu_SPIII2C_transfer(((addr >> 16) & 0xFF) | 0x00);
    Ft_GpuEmu_SPIII2C_transfer((addr >> 8) & 0xFF);
    Ft_GpuEmu_SPIII2C_transfer(addr & 0xFF);
}
```

**Figure 5 – The implementation of C API**

### 3.3 Sample Application adaptation

The FT800 sample application employs a Hardware Abstraction Layer (HAL) to make the application logic independent from the hardware platform. It is defined in “FT_Gpu_Hal.c” and users are assumed to be familiar with it before moving ahead.

To adapt the sample application on the FT800 emulator, the minimum changes are required: just implement the APIs defined in “FT_Gpu_Hal.c” as below:
/* API to initialize the SPI interface */
ft_bool_t  Ft_Gpu_Hal_Init(Ft_Gpu_HalInit_t *halinit)
{
    return TRUE;
}
ft_bool_t  Ft_Gpu_Hal_Open(Ft_Gpu_Hal_Context_t *host)
{
    Ft_GpuEmu_SPII2C_begin();
    host->host->cmd_fifo_wp = host->dti_buff_wp = 0;
    host->status = FT_GPU_HAL_OPENED;
    return TRUE;
}
ft_bool_t  Ft_Gpu_Hal_Close(Ft_Gpu_Hal_Context_t *host)
{
    host->status = FT_GPU_HAL_CLOSED;
    Ft_GpuEmu_SPII2C_end();
}
ft_void_t  Ft_Gpu_Hal_DeInit()
{
/*The APIs for reading/writing transfer continuously only with small buffer system*/
ft_void_t Ft_Gpu_Hal_StartTransfer(Ft_Gpu_Hal_Context_t *host,FT_GPU_TRANSFERDIR_T rw,ft_uint32_t addr)
{
    if (FT_GPU_READ == rw){
        Ft_GpuEmu_SPII2C_StartRead(addr);
        host->status = FT_GPU_HAL_READING;
    }else{
        Ft_GpuEmu_SPII2C_StartWrite(addr);
        host->status = FT_GPU_HAL_WRITING;
    }
}
ft_uint8_t  Ft_Gpu_Hal_Transfer8(Ft_Gpu_Hal_Context_t *host,ft_uint8_t_t value)
{
    return Ft_GpuEmu_SPII2C_transfer(value);
}
ft_void_t  Ft_Gpu_Hal_EndTransfer(Ft_Gpu_Hal_Context_t *host)
{
    Ft_GpuEmu_SPII2C_csHigh();
    host->status = FT_GPU_HAL_OPENED;
    Ft_Gpu_Hal_EndTransfer();
}
ft_uint8_t_t  Ft_Gpu_Hal_WrMem(Ft_Gpu_Hal_Context_t *host,ft_uint32_t addr, const ft_uint8_t *buffer, ft_uint32_t_t length)
{
    ft_uint32_t_t SizeTransfered = 0;
    Ft_Gpu_Hal_StartTransfer(host,FT_GPU_WRITE,addr);
    while (length--){
        Ft_Gpu_Hal_Transfer8(host,buffer);
        buffer++;
    }
    Ft_Gpu_Hal_EndTransfer();
}
ft_void_t  Ft_Gpu_Hal_RdMem(Ft_Gpu_Hal_Context_t *host,ft_uint32_t addr, ft_uint8_t_t *buffer, ft_uint32_t_t length)
{
    ft_uint32_t_t SizeTransfered = 0;
    Ft_Gpu_Hal_StartTransfer(host,FT_GPU_READ,addr);
    while (length--){
        *buffer = Ft_Gpu_Hal_Transfer8(host,0);
        buffer++;
    }
    Ft_Gpu_Hal_EndTransfer();
}

Figure 6 – Hardware Abstraction Layer implementation in emulator API
3.4 Build and Run

After porting the application to the FT800 emulator according to the instructions above, in order to build the final executable, user project is required to specify the path and name of the FT800 emulator library.

For release build, please specify the FT800 emulator library named “FT800Emu.lib”.

For debug build, please specify the FT800 emulator library named “FT800Emud.lib”.

Please note that Microsoft Visual Studio 2012 Express version is a must to link with the emulator library and build your application.

The picture below shows a screenshot of when the FT800 logo application is running on top of FT800 emulator.

![Figure 7 – Logo application running on top of FT800 emulator](image-url)
4 Contact Information

Head Quarters – Singapore  
Bridgetek Pte Ltd  
178 Paya Lebar Road, #07-03  
Singapore 409030  
Tel: +65 6547 4827  
Fax: +65 6841 6071  
E-mail (Sales)  
sales.apac@brtchip.com  
E-mail (Support)  
support.apac@brtchip.com

Branch Office – Taipei, Taiwan  
Bridgetek Pte Ltd, Taiwan Branch  
2 Floor, No. 516, Sec. 1, Nei Hu Road, Nei Hu District  
Taipei 114  
Taiwan, R.O.C.  
Tel: +886 (2) 8797 5691  
Fax: +886 (2) 8751 9737  
E-mail (Sales)  
sales.apac@brtchip.com  
E-mail (Support)  
support.apac@brtchip.com

Branch Office - Glasgow, United Kingdom  
Bridgetek Pte. Ltd.  
Unit 1, 2 Seaward Place, Centurion Business Park  
Glasgow G41 1HH  
United Kingdom  
Tel: +44 (0) 141 429 2777  
Fax: +44 (0) 141 429 2758  
E-mail (Sales)  
sales.emea@brtchip.com  
E-mail (Support)  
support.emea@brtchip.com

Branch Office – Vietnam  
Bridgetek VietNam Company Limited  
Lutaco Tower Building, 5th Floor, 173A Nguyen Van  
Troi,  
Ward 11, Phu Nhuan District,  
Ho Chi Minh City, Vietnam  
Tel: 08 38453222  
Fax : 08 38455222  
E-mail (Sales)  
sales.apac@brtchip.com  
E-mail (Support)  
support.apac@brtchip.com

Web Site  
http://brtchip.com/

Distributor and Sales Representatives  
Please visit the Sales Network page of the Bridgetek Web site for the contact details of our distributor(s) and sales representative(s) in your country.

System and equipment manufacturers and designers are responsible to ensure that their systems, and any Future Technology Devices International Ltd (FTDI) devices incorporated in their systems, meet all applicable safety, regulatory and system-level performance requirements. All application-related information in this document (including application descriptions, suggested FTDI devices and other materials) is provided for reference only. While FTDI has taken care to assure it is accurate, this information is subject to customer confirmation, and FTDI disclaims all liability for system designs and for any applications assistance provided by FTDI. Use of FTDI devices in life support and/or safety applications is entirely at the user’s risk, and the user agrees to defend, indemnify and hold harmless FTDI from any and all damages, claims, suits or expense resulting from such use. This document is subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Neither the whole nor any part of the information contained in, or the product described in this document, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. Future Technology Devices International Ltd, Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH, United Kingdom. Scotland Registered Company Number: SC136640
Appendix A – References

Document References

- FT800 data sheet
- FT800 programmer guide
- AN_240_FT800_From_the_Ground_Up
- FT800 Sample Application

Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Terms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>I^2C</td>
<td>Inter-Integrated Circuit</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>USB-IF</td>
<td>USB Implementers Forum</td>
</tr>
<tr>
<td>Windows</td>
<td>Microsoft Windows Desktop operating system</td>
</tr>
</tbody>
</table>
Appendix B – List of Tables & Figures

List of Tables
Table 1 – FT8xx Emulator library interface structure .................................................. 3

List of Figures
Figure 1 – Definition of structure “EmulatorParameters” .............................................. 7
Figure 2 – Flags field definition .................................................................................... 7
Figure 3 – Start FT8xx emulator ................................................................................... 9
Figure 4 – The C interface API ..................................................................................... 10
Figure 5 – The implementation of C API ...................................................................... 11
Figure 6 – Hardware Abstraction Layer implementation in emulator API .................... 12
Figure 7 – Logo application running on top of FT800 emulator .................................... 13
Appendix C – Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Changes</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Initial Release</td>
<td>2014-07-21</td>
</tr>
<tr>
<td>1.1</td>
<td>Dual branding to reflect the migration of the product to the Bridgetek name – logo changed, copyright changed, contact information changed</td>
<td>2016-09-19</td>
</tr>
</tbody>
</table>