

GTT2.5 Protocol

For all variants of the GTT29A, GTT35A, GTT38A, GTT43A, GTT50A, and GTT70A

Developer Manual

Revision 1.0

Firmware Revision: 2.10 or Higher

Designer Version: 1.10 or Higher

Revision History

Revision	Date	Description	Author
1.0	25 May 2018	Initial Release	Divino

Contents

1 GTT2.5 Protocol Introduction 4

1.1 Legacy Protocol Comparison 4

1.2 GTT2.5 Protocol Elements 5

2 GTT2.5 Protocol 5

2.1 Objects 5

Object Types 5

Create an Object 5

2.2 Properties 6

Property Types 6

Set Property Value 6

Get Property Value 6

Return Messages 7

Status Codes 7

2.3 Events 7

Event Types 7

Event Messages 8

2.4 Methods 8

Execute a Method 8

3 Resources 8

3.1 GTT Designer Report 9

3.2 GTT2.5 Code Libraries 9

Designer .c/.h Project Files 10

GTT Client Library 10

Interface Libraries 10

4 Appendix 12

4.1 Data Types 12

5 Contact 12

1 GTT2.5 Protocol Introduction

The GTT Generation 2.5 Command Protocol represents an evolution to Object-Orientated software design. GTT2.5 commands allow users to modify a wide array of object properties on the fly by storing values in RAM, allowing users to read and modify object properties, such as values, strings, and colour settings dynamically. Compared to the legacy GTT2.0 command set, this protocol advance provides developers greater flexibility and control in HMI design.

1.1 Legacy Protocol Comparison

The GTT Generation 2.5 Command Protocol contains advanced object-oriented commands, many of which are based on the legacy GTT Generation 2.0 Command Protocol, but it does not entirely replace the GTT2.0 command set. All GTT2.0 commands can be used in combination with GTT2.5 commands.

In addition, when upgrading to newer firmware containing the Generation 2.5 Protocol, all Generation 2.0 commands will still retain their functionality. GTT Designer projects developed using GTT2.0 commands will still remain compatible and GTT Generation 2.5 commands will be available on GTT's running firmware 1.10 and up.

Table 1 illustrates the implementation differences between GTT2.5 and GTT2.0 Command Protocol for a dynamic label update example.

Table 1: GTT Protocol Comparison

Function	GTT Legacy Protocol	GTT2.5 Protocol
Create a Label	Create a Label • Label ID • Properties	Create an Object • Label Object Type • Label Object ID
Update Label Text	Update a Label • Label ID • Value	Update a Property • Label Object ID • Text Property Type • Value
Update Label Colour	Create a New Label • Label ID • Properties Update New Label • Label ID • Value	Update a Property • Label Object ID • Colour Property Type • Value

1.2 GTT2.5 Protocol Elements

The Object Orientated software design of the GTT Generation 2.5 Command Protocol consists of a hierarchy of elements, as shown in Figure 1. Elements are typically created within the GTT Designer software environment. Within the Designer environment, when Objects are dragged onto Screens, their Properties are displayed in a panel at the right of the screen. When a Designer project is generated and deployed Screens, Objects, and Properties are managed according to the hierarchy below.

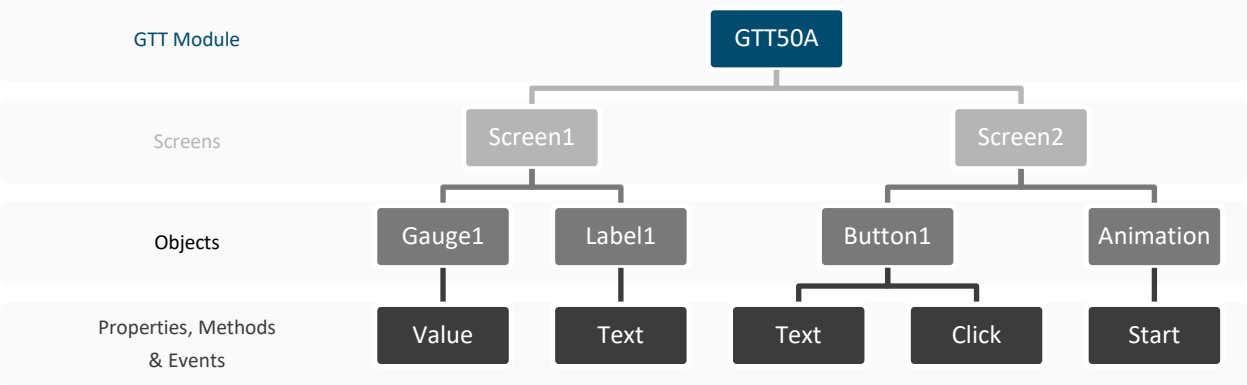


Figure 1: GTT2.5 Protocol Element Hierarchy

2 GTT2.5 Protocol

2.1 Objects

Object Types

In the GTT Generation 2.5 Command Protocol, each element displayed on the screen is represented as an Object. Each Object created belongs to a specific type, and is assigned a unique Identifier value. A list of common objects is shown in Table 2.

Table 2: Common Object Types

Value	Description
3	GTT25Gauge
9	Label
10	Slider
21	Button

Create an Object

Table 3: Object Creation Command

GTT2.5 Command Prefix (U16)	**254**	The GTT2.5 command prefix
	250	
Create Command (U16)	**1**	Command to Create an Object
	0	
Object Type (U16)	**0**	Type of Object (Gauge)
	3	
Object ID (U16)	**0**	Identifier for Object (1)
	1	

The bytes that form the method to create an object are shown in Table 3. An object must be created before any of its relevant properties can be set. Once an object is created, each relevant property will be given a default value. Each of these properties can then be set or retrieved by the user using the appropriate Set or Get method.

2.2 Properties

Property Types

In the GTT Generation 2.5 Command Protocol, every Object can have a number of different Properties associated with it. Each Object has its own set of unique Properties. A list of common properties is shown in the tables below.

Table 4: Common Gauge Property Types

Bytes		Description
3	0	MinValue
3	1	MaxValue
3	2	Value
3	3	NeedleColorR
3	4	NeedleColorG
3	5	NeedleColorB
3	10	LabelR
3	11	LabelG
3	12	LabelB

Table 5: Common Label Property Types

Bytes		Description
9	0	BackgroundR
9	1	BackgroundG
9	2	BackgroundB
9	3	ForegroundR
9	4	ForegroundG
9	5	ForegroundB
9	6	Text

Table 6: Common Slider Property Types

Bytes		Description
10	6	Minimum
10	7	Maximum
10	8	Value
10	11	LabelR
10	12	LabelG
10	13	LabelB

Table 7: Common Button Property Types

Bytes		Description
15	3	Text
15	5	ForegroundR
15	6	ForegroundG
15	7	ForegroundB
15	8	State

Set Property Value

Table 8: Set Property (U8) Command

GTT2.5 Command Prefix (U16)	**254**	The GTT2.5 command prefix
	250	
Set U8 Property Command (U16)	**1**	Command to Set a U8 Property Value
	4	
Object ID (U16)	**0**	Identifier for Object (1)
	1	
Property Type (U16)	**3**	Type of Property (Gauge Needle Red)
	3	
Value (U8)	**255**	Value for Property (255)

The bytes that form the method to set a byte length property value of an object are shown in Table 8. Certain properties, such as text based properties, will require different Value data types. The Project Report generated by the GTT Designer will contain more detail regarding which data types are required to set specific properties, as well as which properties are available for each specific object type on screen.

Get Property Value

The bytes that form the method to get the value of a byte length property of an object are shown in Table 9. All GTT 2.5 Property values can be read using their respective Get property commands, including text strings, button states, and bar graph and gauge values. The Project Report generated by the GTT Designer will contain more detail regarding which properties are available for each specific object type on screen.

Table 9: Get Property (U8) Command

GTT2.5 Command Prefix (U16)	**254**	The GTT2.5 command prefix
	250	
Get U8 Property Command (U16)	**1**	Command to Get a U8 Property Value
	5	
Object ID (U16)	**0**	Identifier for Object (1)
	1	
Property Type (U16)	**3**	Type of Property (Gauge Needle Red)
	3	

Return Messages

Table 10: Set Property (U8) Command Return Message

GTT2.5 Return Prefix (U16)	**252** **250**	The GTT2.5 command prefix
Return Message Length (U16)	**0** **4**	Length of the Return Message to follow
Get U8 Property Command (U16)	**1** **5**	Command to Get a U8 Property Value
Status Code (U8)	**254**	Status of Command (Success)
Value (U8)	**255**	Value for Property (255)

The bytes that form the return message from the execution of a command to get a byte length property value of an object are shown in Table 10. The returned length of data will vary depending on the type property information being returned. For example, Text properties may return a string of characters upon reading. The Project Report generated by the GTT Designer will contain more detail regarding the data types and return length.

When the GTT is asked to provide notifications to the host, by using the Designer to set the Default Channel for example, it is important that Return messages are read. If return messages are not read by the host, they may fill the GTT's return buffer causing the GTT to restart. Return messages from the GTT can be turned off by setting the Communication Channel to None. This will prevent command and event return messages from being generated by the GTT.

Status Codes

In the GTT Generation 2.5 Command Protocol, every command results in the generation of a status return. A list of common status codes is shown in the tables below.

Table 11: Common File Statuses

Value	Description
0	FileNotFound
1	InvalidBitmapFileFormat
2	Invalid9SliceMetrics
7	InvalidAnimationFileFormat
10	DisplayisOUTofRAM
11	InvalidRegionFileFormat
14	InvalidFileFormat

Table 12: Common Object Statuses

Value	Description
3	Invalid9SliceIndex
4	InvalidBitmapIndex
5	InvalidBargraphIndex
6	InvalidAnimationIndex
8	InvalidFontIndex
15	InvalidTraceIndex
16	InvalidTouchRegion
17	InvalidLabelIndex

Table 13: Common Module Statuses

Value	Description
9	InvalidCommandParameters
12	InvalidTouchCalibration
13	SuccessfulTouchCalibration
254	Success
255	UnknownException

2.3 Events

Event Types

Many events are represented by a combination of bytes. Events such as button clicks, successful property changes, and property feedback are reported back to the host, and are differentiated by specific event byte combinations. A list of common event types is shown in Table 14.

Table 14: Common Event Types

Bytes		Description
1	1	GTT25BaseObject_OnPropertyChange
2	0	GTT25VisualObject_OnKey
21	1	Button_Click

Event Messages

Table 15: Event Message from Button Click

GTT2.5 Button Event Prefix (U16)	**254** **235**	The GTT2.5 Button Click Event prefix
Return Message Length (U16)	**0** **5**	Length of the Return Message to follow
Event Type (U16)	**21** **0**	Type of Event (Button Click)
Object ID (U16)	**0** **27**	Identifier for Object (27)
Value (U8)	**1**	Value for Property (Down)

The bytes that form the return message from the click event of a button object are shown in Table 15. All event feedback will follow a similar message format, but may differ based on the type of data that is being returned. Once an event occurs, the GTT will immediately generate an Event Message and return it to the host. The host will be able to determine what happened based on the Event Type, Object ID, and Value, and take the appropriate action.

2.4 Methods

Execute a Method

Table 16: Start Animation Command

GTT2.5 Command Prefix (U16)	**254** **250**	The GTT2.5 command prefix
Start Animation Command (U16)	**32** **1**	Command to Start an Animation
Object ID (U16)	**0** **1**	Identifier for Object (1)

The bytes that form the method to start an animation object are shown in Table 16. In addition to basic Get and Set methods for properties, such as text or Object colours, some Objects also have unique methods, such as the Start method for the Animation object.

3 Resources

With the release of the GTT Generation 2.5 Command Protocol, a number of resources have been made available to developers within the GTT Designer software environment that will dramatically increase creation efficiency of HMI designs.

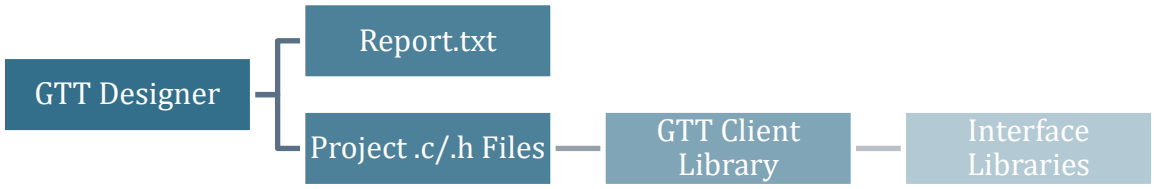


Figure 2: GTT2.5 Protocol Resources

In addition to an upgraded Report file generated by the Designer, developers will now have access to project specific .c/.h code files built on top of a provided GTT Client Library. Finally, developers can build Interface Libraries unique to their specific hardware to support any HMI host controller.

3.1 GTT Designer Report

The GTT Designer Report.txt file lists all Objects and methods for getting and setting relevant Properties. This file is found in the Output directory of a Project after Generation has been completed.

The report can be directly referenced during development, as it highlights which object properties can be changed. In addition, the report provides information on which data types to send, and which data types to expect to receive when modifying object properties.

Table 17: Example GTT Designer Report File

GTT Designer Report File	Element
Project Report:GTT2S Objects	Project
Screen (Screen1)	Screen
Dynamic Label (GTT2.5) (Dynamic_Label_1)	Object
ObjectID: 2 TextValue: Label Label_Foreground_RGB: 238, 240, 242 Label_Background_RGB: 93, 93, 93	Property
Set Dynamic_Label_1__Text: 254 250 1 10 0 2 9 6 eTextEncoding String Byte Length Value String - Parameter Name (Data Type): Description CommandPrefix (FixedDec): 254 250 1 10 ObjectID (U16): 0 2 PropertyID (U16): 9 6 Value_StringType (U8): eTextEncoding - eTextEncoding: Unicode = 0, ASCII = 1, UTF-8 = 2 Value_ByteLength (U16): String Byte Length Value (U8[]): Value String	Method
Return for Set Dynamic_Label_1__Text: 252 250 Return Byte Length 1 10 eStatusCode - Parameter Name (Data Type): Description ReturnPrefix (U16): 252 250 ReturnLength (U16): Return Byte Length Gtt2SCommandID (U16): 1 10 Gtt2SCommandStatus (U8): eStatusCode	Return Message

3.2 GTT2.5 Code Libraries

The GTT Generation 2.5 Command Protocol allows developers to easily interface their GTT Project to a Host Device using a suite of generated code libraries. The code libraries consist of .c/.h files unique to your GTT Project, worker functions to packetize data, and interface libraries specific to your Host Device.



Figure 3: GTT Client Write Process

All of the Object, Property, Event, and Method elements of the GTT2.5 Protocol can be found in the .c/.h files that the Designer generates for your specific project. These elements are then passed as packets either from the GTT to your host device through the protocol worker or from your host device to the GTT through the parser, as shown in Figure 3 and Figure 4.



Figure 4: GTT Client Read Process

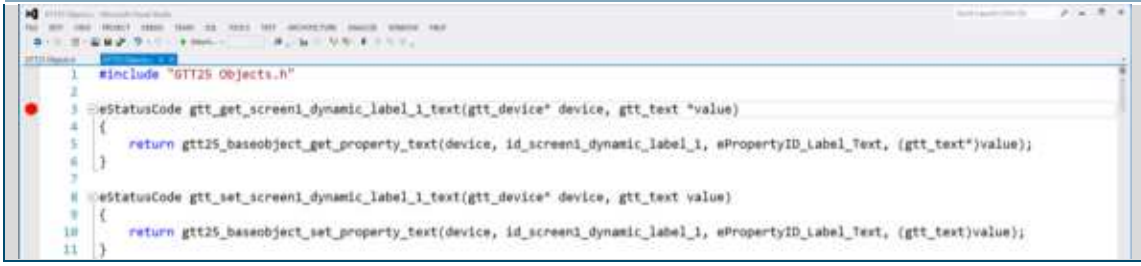
Designer .c/.h Project Files

The GTT Designer .c and .h Project files, found in the Output directory of a Project after Generation has been completed, provide a list of all Object Identifiers and simple functions to get and set relevant Properties of those Objects.

Table 18: Example GTT Designer .h File

GTT Designer Report File	Element
	Project Screen Object Method/ Return

Table 19: Example GTT Designer .c File

GTT Designer Report File	Element
	Property/ Method

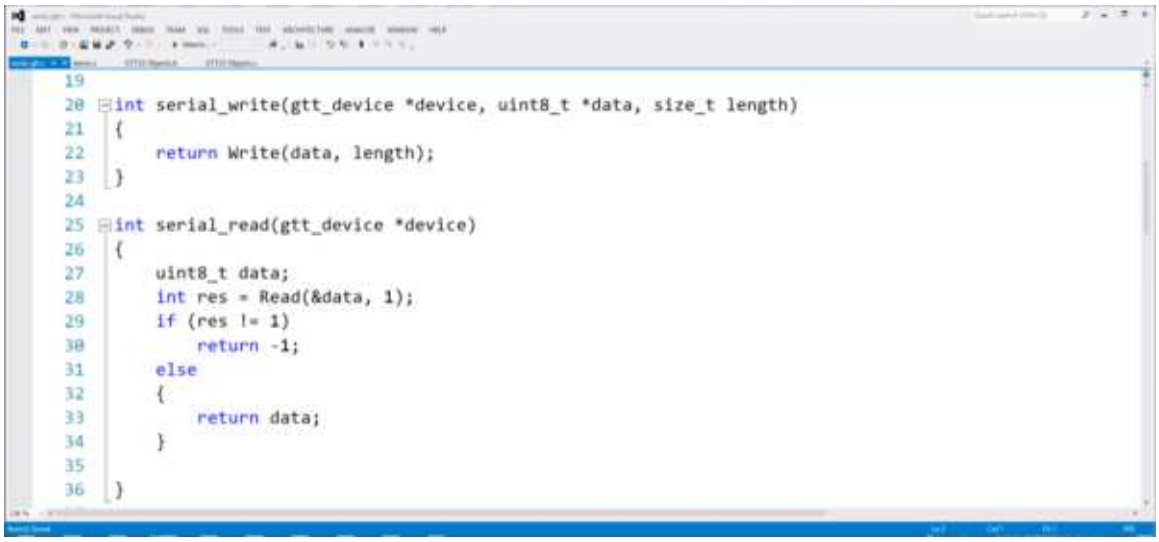
GTT Client Library

The .c and .h Project files created by the GTT Designer will require some supporting code in order to run correctly. This supporting code is contained within the GTT Client Library which can be found in the GTT's Firmware download folder. The Client Library must be included in order to run the .c and .h files generated by the GTT Designer.

Interface Libraries

While the GTT Project .c/.h files are automatically generated and the GTT Client libraries are available for download, when developing with a new Host Device it may be necessary to create an interface library. This interface should include at a minimum one function to read a byte from the GTT module into your device and one function to write a byte from your device to the GTT.

In order for the library to operate properly, generic platform dependent read and write functions must be included in the interface library. The write function must be able to transfer "length" bytes of "data" to the GTT. The write function must also return the number of bytes written to the GTT. The read function must be able to read a single byte, and return a -1 if no data is available. If the read function is not configured properly, the library's read loop may continuously run when reading information from the GTT.



```
int serial_write(gtt_device *device, uint8_t *data, size_t length)
{
    return Write(data, length);
}

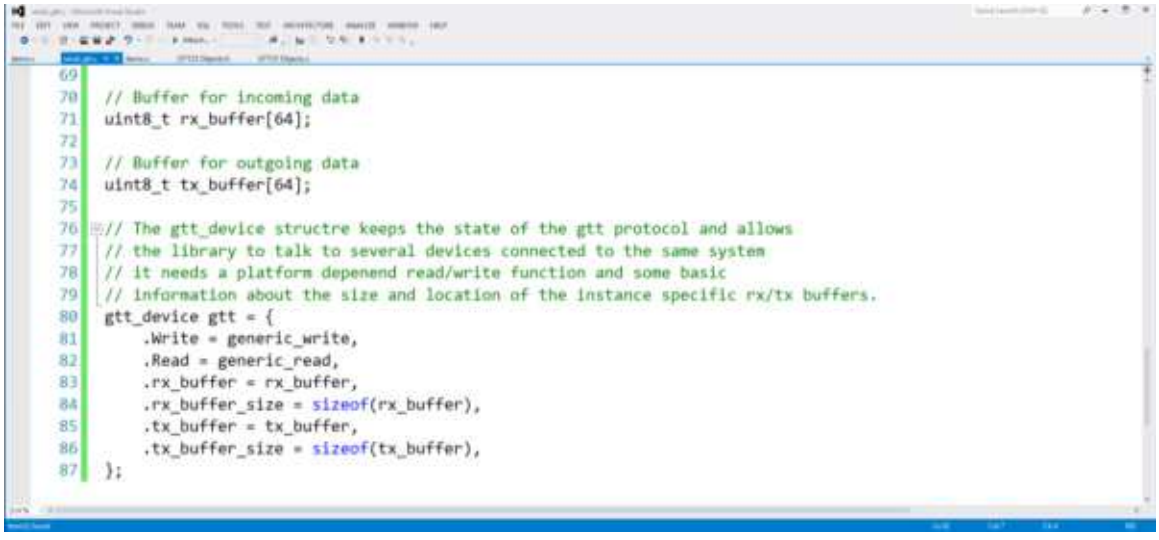
int serial_read(gtt_device *device)
{
    uint8_t data;
    int res = Read(&data, 1);
    if (res != 1)
        return -1;
    else
    {
        return data;
    }

}
```

Figure 5: Generic read and write functions written in C

An output and an input buffer also need to be defined in the interface library. Both the output and input buffer should allocate enough space to handle the largest amount of bytes you plan on sending and receiving at any given point.

Once the communication buffers and read and write buffers are set up the GTT device can be configured. The GTT Device structure will require pointers to the write and read functions, as well as the input and output buffers.



```
// Buffer for incoming data
uint8_t rx_buffer[64];

// Buffer for outgoing data
uint8_t tx_buffer[64];

// The gtt_device structre keeps the state of the gtt protocol and allows
// the library to talk to several devices connected to the same system
// it needs a platform depenend read/write function and some basic
// information about the size and location of the instance specific rx/tx buffers.
gtt_device gtt = {
    .Write = generic_write,
    .Read = generic_read,
    .rx_buffer = rx_buffer,
    .rx_buffer_size = sizeof(rx_buffer),
    .tx_buffer = tx_buffer,
    .tx_buffer_size = sizeof(tx_buffer),
};
```

Figure 6: Configuring a GTT Device

4 Appendix

4.1 Data Types

The following table outlines native data types in common programming languages that can be used to represent the data types used in this manual.

Table 20: Data Types with Representations

	ANSI C/C++	C#	Visual Basic
U8	unsigned char	byte	Byte
U16	unsigned short	ushort	UShort
S16	short	short	Short
U32	unsigned int	uint	UInteger
S32	int	int	Integer
Text	N/A	N/A	N/A

Table 21: Data Type Descriptions

Byte	Unsigned 8 bit data type that can represent a value from 0 to 255.
Short*	Unsigned 16 bit data type can represent values from 0 to 65,535.
Signed Short*	Signed 16 bit data type that can represent values from -32,768 to 32,767.
Integer *	Unsigned 32 bit data type that can represent values from 0 to 4,294,967,295.
Signed Integer*	Signed 32 bit data type that can represent values of -2,147,483,648 to 2,147,483.
Text	String data type that can be Unicode, ASCII, or UTF8. The format of the Text type is Encoding (U8) Length (U16) Data[]. Where Encoding is 0 (Unicode), 1 (ASCII), or 2 (UTF8).

***Note:** Transmission of multiple byte values are transferred in big endian (MSB first) order.

5 Contact

Sales

Phone: 403.229.2737
Email: sales@matrixorbital.ca

Support

Phone: 403.229.2737
Email: support@matrixorbital.ca

Design

Phone: 403.229.2737
Email: design@matrixorbital.ca

Online

Purchase: www.matrixorbital.com
Support: www.matrixorbital.ca