QLARITY-BASED TERMINAL PROGRAMMER'S REFERENCE MANUAL REVISION 2.4

QSI CORPORATION
2212 South West Temple #50
Salt Lake City, Utah 84115-2648
USA

Phone 801-466-8770
Fax 801-466-8792
Email info@qsicorp.com
Web www.qsicorp.com

Manual 0060-01
6345E1 - Printed in USA

FOREWORD

The Qlarity-based terminals' software is divided into four major components: User Application (created in Qlarity Foundry™ or a compatible text editor using the Qlarity™ programming language), Application Binary (the user application compiled as binary so it can be downloaded to the terminal), System Software and Operating System (written in C using µC/OS) and the Hardware Abstraction (for the Windows® and the terminal's hardware drivers).

Chapter 1 **Qlarity-based Terminal Software Fundamentals.** This chapter provides basic information on the Qlarity-based terminal software, including the Qlarity programming language and the message handling system.

Chapter 2 **Qlarity Language Syntax.** This chapter provides detailed information on the syntax for writing in Qlarity.

Chapter 3 **Messages and Message Handler Prototypes.** This chapter provides descriptions of all system messages. Included is the format (number and types of parameters and return type) that must be used to declare a handler for each message.

Chapter 4 **Qlarity API Function Reference.** This chapter describes the available API (Application Programming Interface) functions, their parameters and return values, and the operations they perform.

Appendix A **Built-in Constants and Defined Types.** This appendix contains tables showing all of the Qlarity built-in constants and defined types.

Appendix B **Exception List.** This appendix lists the names and associated descriptions of all possible exceptions.

Appendix C **Qlarity Command Line Compiler.** This appendix provides information on the Qlarity command line compiler, which is used to compile the BASIC file if you created your user application in a text editor.

Appendix D **Qlarity API Functions Quick Reference List.** This appendix provides an alphabetical listing of all Qlarity API functions for quick reference.

MANUAL CONVENTIONS

The following conventions are used to identify selections in this manual.

Syntax, commands, and examples are shown in the Courier typeface: `rem <comment>`

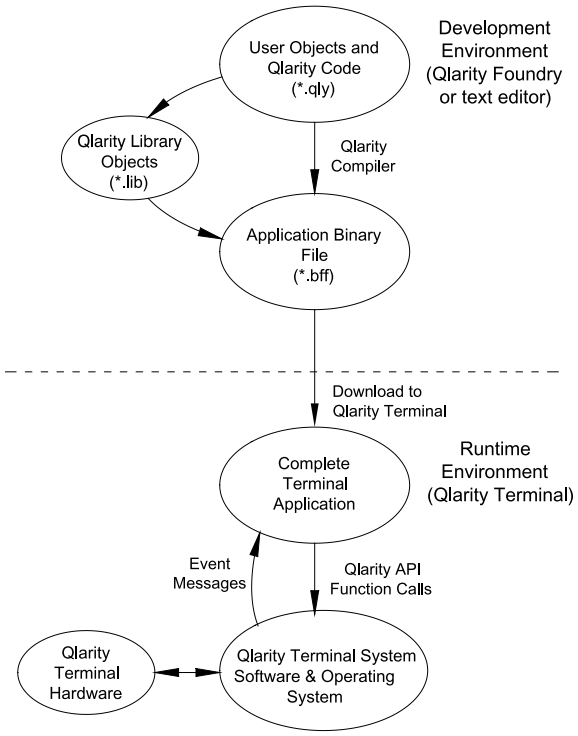
CONTENTS

CHAPTER 1

QLARITY-BASED TERMINAL SOFTWARE FUNDAMENTALS

1.1 Qlarity-based Terminal Software

Qlarity-based terminal software is divided into four major components, as shown below.



1.1.1 System Software (Firmware)

The system software (firmware) is the terminal resident software that controls the Qlarity-based terminal. System software consists of the following:

- Operating system
- Qlarity™ execution engine
- Message handling system
- Qlarity API functions (operating system calls)

The following third-party components are used in the system software:

- The operating system is based on µC/OS, the Real Time Kernel, by Jean Labrosse.
- The FreeType library is used to render TrueType fonts.
- The TCP/IP stack is based on code in the XINU operating system, written by Douglas Comer and others.
- The ZLIB library is used for compression / decompression.

1.1.2 User Application

The user application code consists of:

- Object templates
- Global code and data
- Object instances

You can create a user application using Qlarity Foundry™, which runs in Windows®, or by writing a text description of the application in the Qlarity language (using a text editor). You then compile the design file, which converts it to an application binary file that can be downloaded to the terminal and stored in flash memory.

1.2 Qlarity Programming Language

The Qlarity language provides an object-based, event driven framework for the user application. The user application is a collection of objects and data that determine how a user will interact with the terminal at runtime. You design the application by selecting or creating “object templates” for the objects you want to use. Each object template has properties and methods that define the function of the object. You create “object instances” from the object templates by setting up the properties for each object in the user application. Code written in Qlarity defines the objects, their properties, and their functions.

Objects are the basic units of a Qlarity application. Each object template is an abstract representation of a user interface element such as text, a picture, a line, a window, a button, and so on. The abstract representation allows you to control each element by setting familiar properties of the element without worrying about the low level details of the actual manipulation.

Objects are characterized by their properties (data) and methods (code). The properties and methods define what an object is and how it behaves.

1.2.1 Object Templates

An object template is defined by the properties and methods that it contains, as well as, those of the template it extends (if any). All object instances created from the same object template have the same number and type of properties (although the values stored in these properties may be different). Similarly, all object instances from the same template have the same list of methods (although each object may override the default functionality of a method with user-defined code).

Each object template can define default values for its properties and default code for its methods. When an object is instantiated (i.e., an object instance is created from the object template), any properties that are not explicitly given values retain their default values. Any method that is not explicitly overridden with user-defined code retains its default functionality.

1.2.1.1 Defining a New Object Template

To define a new object template, you give the object a name, declare what type of object it will be (non-drawable, area, or container), and then add the properties and methods to the object definition.

Variables that are created (using the dim statement) inside the object definition are object properties. These variables may be given a default initial value (using the init statement) that will be assigned to the variable if no initial value is given the object instance.

Functions that are created inside the object definition are object methods. These methods may consist of ordinary functions, validation functions, or event handlers. Methods need not contain any actual code; however, a method must be declared in the object definition before it can be overridden in instances of that object template.

The actual syntax for defining a new object template is covered in the chapter on "Advanced Design" in the *Qlarity Foundry User's Manual*.

1.2.1.2 Creating Instances of an Object

To create instances of an object, you give the instance a name and declare the name of the template from which the instance will be created.

The properties of the object instance are those defined in the template. New properties may not be defined in the instance, but initial values for the instance may be assigned inside the instance declaration (using the init statement).

The methods of the instance are also those defined in the template. You may override these methods by declaring a function inside the instance with the same name, parameters, and return value as the method defined in the template. This function replaces the template function for that instance only. The code for the override function may call the template method if desired.

1.2.2 Object Types

There are three basic types of object templates: non-drawable objects, area objects, and container objects. Each of these object types has certain built-in functionality that helps the system software handle the object efficiently.

1.2.2.1 Non-Drawable Objects

The non-drawable object type includes objects that do not directly interact with the terminal display. Examples include a keypad object (not a touch key object) and a serial communications object. Non-drawable objects have the following built-in data that is maintained by the system software and accessible through the Qlarity API (Application Programming Interface):

NAME (character string) – the name of the object

ENABLED (boolean) – indicates whether the object is eligible to process messages.

PARENT (reference to container) – the parent container for the object

Note that these attributes are NOT actual properties of the object. However, when using Qlarity to create an object, property variables that represent these attributes are often

added to the object. Regardless of whether the object itself maintains properties analogous to these attributes, the attributes must be initialized and maintained by the object through Qlarity API function calls. The enabled status of an object can be modified through the Enable() API function, and the parent can be assigned with the Attach() API function. All Qlarity API functions are described in Chapter 4, "Qlarity API Function Reference".

1.2.2.2 Area Objects

The area object type includes objects that directly interact with the terminal display by drawing something on the display and/or processing area-based messages. All area objects are rectangular. Examples include a text object, a bitmap object, and a touch key object. Area objects have the following built-in data that is maintained by the system software and accessible through the Qlarity API:

NAME (character string) – the name of the object

ENABLED (boolean) – indicates whether the object is eligible to process messages

PARENT (reference to container) – the parent container for the object

XPOSITION (integer) – the horizontal displacement (in pixels) from the origin of the object's parent container. Positive values indicate displacement to the right.

YPOSITION (integer) – the vertical displacement (in pixels) from the origin of the object's parent container. Positive values indicate downward displacement.

XSIZE (integer) – the width of the object in pixels

YSIZE (integer) – the height of the object in pixels

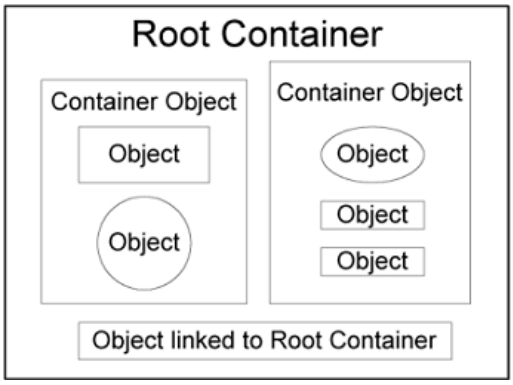
As with non-drawable objects, these attributes are not strictly properties of an object, but the object often defines analogous properties. In any event, every object of this type must initialize and maintain these attributes through Qlarity API function calls. The position and size of the object are established and can be modified through the Relocate() and Resize() API functions. The enabled status of an object can be modified through the Enable() API function, and the parent can be assigned with the Attach() API function.

The separation of these system software attributes and the object properties provides you with the freedom to choose properties that best describe an object. There is no need to make the "built-in properties" or system software attributes visible outside the object. For example, a circle object might best be described with a center point and a radius. When these properties are set, code inside the object calculates the position and size from the property values and calls the Qlarity API functions that set the attributes.

1.2.2.3 Container Objects

The container object type is very similar to the area object type, except that container objects may "contain" other objects. Container objects can be used to create a hierarchical organization of objects in an application; thus they are central to the Qlarity messaging system.

Each user application starts with a "root container." Though invisible, it is the container in which you place all other objects. The following illustration shows a root container with several objects in it.



In this illustration, two container objects plus another object are linked to the root container. In addition, each container object has objects linked to it, which are referred to as its "children." The container object is the "parent." If an object is not attached to a container object, it is linked to the root container by default.

All container objects are rectangular. Examples might include a form object or a window object, which could serve as generic containers to organize objects into areas on the terminal display.

Container type objects have the following built-in data that is maintained by the system software and may be set through the Qlarity API:

NAME (character string) – the name of the object

ENABLED (boolean) – indicates whether the object is eligible to process messages

PARENT (reference to a container) – the parent container for the object

CHILDREN (ordered list of object references) – the list of objects attached to the container (maintained in Z-order, see section 1.4)

XPOSITION (integer) – the horizontal displacement (in pixels) from the origin of the object's parent container. Positive values indicate displacement to the right.

YPOSITION (integer) – the vertical displacement (in pixels) from the origin of the object's parent container. Positive values indicate downward displacement.

XSIZE (integer) – the width of the object in pixels.

YSIZE (integer) – the height of the object in pixels.

ORIGINX (integer) – the x coordinate of the top left pixel in the container, relative to the parent container.

ORIGINY (integer) – the y coordinate of the top left pixel in the container, relative to the parent container.

These attributes are identical to the attributes of an area type object except for the list of children. The position and size of the container are accessible through the Relocate() and Resize() API functions. The enabled status of an object is accessible through the Enable() API function, and the parent and children attributes can be manipulated with the Attach() API function and the Z-order change functions.

1.3 Event Processing

Qlarity is event-driven, which means that instead of one long, linear program being run that has constant control of the terminal, relatively short and independent sections of code are run in response to specific events. These code sections are called event handlers. Typical events include: a timer tick, a keypad press or release, a touch screen press or release, or a serial character receive.

In Qlarity, event handlers can be defined as object methods or global functions that are registered to handle a given event. Events are sent through the message handling system (see section 1.5, "Message Handling System").

1.4 Z-Order

Z-order describes "front to back" ordering (which is the same as the object hierarchy ordering). On a two-dimensional display, where X denotes left to right spacing and Y denotes top to bottom spacing, the Z axis extends out normal to the plane of the display. Thus, Z-ordering is the implied third dimension on the display, allowing objects to be "in front of" or "behind" other objects.

1.5 Message Handling System

When an event occurs, the system software (firmware) and/or hardware drivers generate a message indicating that the event has occurred. When a message is generated, it is passed through the message handling system located in the system software, which determines who gets the message and in what order the message is processed.

An object must be enabled to process most messages. This also causes area and container objects with non-zero area to be drawn on the screen. Disabled containers do not process most messages and therefore cannot pass those messages to their children (attached objects). (Broadcast, tool, and direct user messages are processed by enabled or disabled objects.)

Each message is processed to completion before the next message is examined and distributed. Processing of the current message is never preempted by another message, except for certain user messages. See Chapter 3, "Messages and Message Handler Prototypes" for more details.

There are seven different types of events, classified by how each type is handled by the messaging system, as follows:

- Broadcast messages
- Area messages
- Draw messages
- Registered messages
- User messages
- Direct messages
- Tool messages (generated only in Qlarity Foundry)

1.5.1 Broadcast Messages

Broadcast messages are sent to all objects (regardless of enabled or disabled status) in Z-order. A message is first passed to the root container, which passes it to each of its children beginning with the front-most object (highest Z-order). When the message is passed to a container object, the container object first handles the message then passes it to each of its children beginning with the front-most object. In this manner, the message filters down through the object hierarchy until all objects have received the message.

Refer to section 3.1 for a list of valid broadcast messages.

1.5.2 Area Messages

Area messages are associated with a given area, or point, (X,Y coordinate) on the terminal display. The message is first passed to the root container, which checks for any enabled children that contain or overlap the area of the message. This checking is done in Z-order, starting with the front-most object. If a child is eligible to receive the message, the message is passed to the child. When the message is passed to a container object, it first handles the message then does a similar check for enabled children which contain or overlap the message area. In this manner, the message filters down through the object hierarchy until all eligible objects have received it.

Refer to section 3.2 for a list of valid area messages.

1.5.3 Draw Messages

A draw message is a type of area message that is processed in reverse Z-order, or back to front (see section 1.4, "Z-Order").

Refer to section 3.3 for more information on draw messages.

1.5.4 Registered Messages

Registered messages are associated with system events, such as a serial character receive or timer tick. Event handlers that process these messages are registered with the system software using registration functions in the Qlarity API. Because these events are usually handled by a small number of functions, registration avoids the need to filter the message through the entire object hierarchy.

If the handler in the registration list is an object method, the object owning the method is checked by the system software to ensure that it and every parent container (back to the root container) is enabled. If this condition is satisfied, the event handler is called. Because the root container cannot be disabled, global function event handlers are called without this check.

If more than one object has registered for a given message, the objects receive the message in Z-order (as with area messages) with the root container (global handlers) receiving highest priority.

Refer to section 3.4 for a list of valid registered messages.

1.5.5 User Messages

Although hardware events are generated by the system software, it is often desirable for the application to have the ability to send messages to indicate software events. Qlarity provides a mechanism to define user messages and a flexible set of Qlarity API functions to send these messages. User messages may be broadcast messages (object may be enabled or disabled), send messages (behave like area messages), or direct (object may be enabled or disabled).

Refer to section 3.5 for details on defining and sending user messages and defining user message handlers.

1.5.6 Direct Messages

Direct messages are sent by the system to a specific object. These messages are enqueued in the message queue as are most other messages, but they are only passed to a single object when they are handled.

Refer to section 3.6 for more information on direct messages.

1.5.7 Tool Messages

Because objects may be completely defined by the user, Qlarity Foundry has no knowledge of the appearance or behavior of a given object. Therefore, the object itself must define its behavior for Qlarity Foundry development activities such as drag-and-drop, resize by dragging resize grips on a selected object, and so on. Tool messages are generated by the Qlarity Foundry software to tell an object that these activities are taking place. These messages are sent directly to the target object (i.e., they do not propagate through the

object hierarchy). The object should have message handlers for these events to properly function in Qlarity Foundry.

Refer to section 3.7 for a list of valid tool messages.

1.5.8 Handling Events

A message is handled by defining a global function or a function within an object that handles that event. Because information about the event is often passed as a parameter to the handler function, the handlers for a given message must follow a specific format (number and type of parameters and type of return value).

Often, when a message is handled by an object or function, it is desirable that the message be terminated (i.e., not passed on to other objects). This is indicated by the boolean return value of the message handler. If the handler returns true, the message is terminated. If the handler returns false, then the message is allowed to propagate to the next eligible object in the hierarchy. Some messages (the draw message in particular) cannot be terminated in this manner.

If a global message handler is defined, it is considered to be a method of the root container. It therefore will be called before the message is passed to any other object.

CHAPTER 2

QLARITY LANGUAGE SYNTAX

The syntax of the Qlarity language is based on the BASIC programming language, with extensions to handle objects, allow structured programming, and facilitate the development of user interface applications.

When referring to language syntax, the following notations are used:

Notation	Description
`font`	Language examples are shown in `monospaced font`.
`[ ]`	Any part of a statement in brackets ([ ]) is optional.
< >	A word that appears in angle brackets (< >) must be replaced by an appropriate name, keyword, or value.

2.1 Qlarity Statements

A statement is defined as a single line of code. In order for the compiler to distinguish between statements, each statement must be separated by a "newline" character. If a statement must be split into multiple lines, type `->` at the end of the line to tell the compiler to look for the rest of the statement on the next line.

2.2 White Space

While spaces and tabs are ignored by the compiler, judicious use of white space is encouraged to make programs more readable.

2.3 Comments

Syntax:

```
rem <comment>
'<comment>        (where ' is an apostrophe)
```

Example:

```
rem this is a comment
'this is another comment
```

Description:

Comments help to explain and increase the readability of the code. Comments can be placed on the same line as a statement or they can be placed on their own line. Everything between the "rem" keyword or apostrophe and the end of the line is considered part of the comment and ignored.

NOTE:
The apostrophe character is also used to denote character literals. For example: 'X '. For this reason, you cannot specify an apostrophe as the second character of a comment.

Example:

```
'I'll fix this tomorrow
```

Would be an error. This could be alternately written:

```
REM   I'll fix this tomorrow
```

or

`' I'll fix this tomorrow` (space after the first apostrophe)

There is another class of comments that may be used to define documentation meta data for an object, property or method. These documentation meta data comments, referred to as AutoDoc, are used to document Qlarity software elements and are described in the *Qlarity Foundry User's Manual*. Regardless of which type of comment you use, it is ignored by the compiler.

2.4 Naming of Identifiers

Reserved keywords may not be used as object, variable, or constant names in Qlarity. Identifier names must begin with any letter A through Z or the underscore character (_). After the initial character, any alphanumeric character is legal, and names may optionally end with a #, $, or % character. Identifier names and keywords in Qlarity are not case sensitive.

2.5 Built-In Data Types

Data types are used to tell the compiler how a specific variable or property will be used. A variable must be given a type when it is declared. The following built-in types are supported:

Data Type	Description	Range
integer	A 32-bit signed whole number	-2,147,483,648 to 2,147,483,647
float	A 32-bit floating point number	-3.402823e+38 to 3.402823e+38, smallest positive float is 1.175494351e-38
boolean	True/false	TRUE, FALSE
byte	8-bit unsigned value	0 to 255
unibyte	16-bit unsigned value	0 to 65535
string	Array of bytes	
unistring	Array of unibytes	
color	Reference to a color	COL_0 to COL_255
enumerations	User defined types (integer values)	Items have integer values
bdffont	A raster font resource	
ttfont	A TrueType font resource	
bitmap	A bitmap image resource	
objref	Reference to an object	
anytype	Any type is allowed (only allowed in API functions)	
Aggregate%	Array of Bytes	
Colormap%	8-bit unsigned value	0 to 255
Stylemap%	32-bit signed whole number	-2,147,483,648 to 2,147,483,647

2.6 User-Defined Data Types

In addition to the built-in data types, there are four kinds of user-defined types. Objects are the major forms of user-defined types and are described elsewhere in this manual. Constants, enumerations, and starttypes are other user-defined types.

NOTE:
User-defined data types must be declared in the global space; they cannot be declared inside of object templates or instances.

2.6.1 Constants

Syntax:

```
constant <name> := <value> [ as <data_type> ]
```

or

```
const <name> := <value> [ as <data_type> ]
```

Example:

```
const myConstant := 100
```

Description:

With the "constant" keyword, a value of any type can be assigned to a name for use throughout the application. That is, the value and the name are synonymous to the compiler. If desired, the constant can be given a data type. Because it is a constant, its value cannot be changed at runtime.

2.6.2 Enumerations

Syntax:

```
enumerate <enumeration_name> as <name_1> ->
[,<name_n>]
```

or

```
enumerate <enumeration_name> as <name_1> ->
:= <value_1> [,<name_2> := <value_2>]
```

NOTE:
The keyword "enum" can be substituted for "enumerate."

Example:

```
enumerate mysize as small := 1, medium ->
:= 2, large := 3
enum stooge as larry,curly,moe,shemp
dim favorite as stooge
```

Description:

An enumeration allows you to define a data type that has a restricted set of values. Any variable declared with an enumeration type can only hold the values defined in the enumeration. The default value of an enumeration is the first value that it defines. Numeric integer values can also be given to each enumeration item. Comparisons such as "greater than" or "less than" can be used between variables of the enumeration type.

In the example above, the variable "favorite" can only be assigned the values "larry," "curly," "moe," and "shemp."

2.6.3 Start Type

Syntax:

```
starttype <name> as <data_type>
   [<name1> := <value1>]
   [<name2> := <value2>]
   [etc.]
endtype
```

Example:

```
starttype control_char as byte
   STX := 0x02
   ETX := 0x03
   ACK := 0x06
   NAK := 0x15
endtype
```

Description:

"Starttype" is similar to an enumeration. The difference is that the items in a "starttype" must be initialized to a value and can be used in any mathematical operations available to the base type. "Starttypes" can only be defined for booleans, unibytes, bytes, integers, and floats. There is no limit to the number of values that can be placed in a "starttype." The first value listed is the default.

2.7 Variables

2.7.1 Declaration

Syntax:

```
dimension <name> as <data_type>
```

or

```
dim <name> as <data_type>
```

Example:

```
dim myVar as integer
dimension myVar as integer
```

Description:

Variables that are used must be declared somewhere in the application. A variable must be assigned a type when it is declared.

The "dim" keyword is used to declare a variable, as shown above. When a variable is declared, it is automatically initialized to a default value. The "init" keyword is used to initialize a variable to a specific value (see section 2.7.2).

Variables declared outside of object definitions and functions are considered global. These variables are accessible anywhere in the application.

Variables declared inside an object definition (but outside the object methods) are considered to be properties of the object. Each instance of an object maintains its own object properties. Inside the object definition or instance, a variable is referenced with its name only. Outside the instance or object definition, the property of a given object instance can be referenced using the instance name and the . (dot) operator (as described in section 2.10.4).

Variables declared inside a function (either a global function or object methods) are local to that function and are not accessible outside the function.

The declaration of each variable need not be stated before it is initialized or used on a global level or in an object definition. This is not true in a function; function variable declarations must appear before any other code in the function.

It is possible to declare multiple variables of the same type on one line, as follows:

Syntax:

```
dim <name>, <name>, <name>, ...  ... as ->
<data_type>
```

Example:

```
dim var1, var2, var3, var4 as string
```

2.7.2 Variable Initialization

Syntax:

```
init <name> := <value>
```

Examples:

```
dim myVar as integer
init myVar := 10

dim myString as string
init myString := "Hello World.\r\n"
```

Description:

The "init" keyword is used to initialize a variable to a specified value at compilation. Variables can be initialized with literal values, other constants, enumerations, or names of object instances (for objref variables). The value given must be of the same type as the variable, and it cannot be an expression containing another variable. Runtime changes to variable values must take place within a function.

2.7.3 Private and Protected Variables

Syntax:

```
private dim <name> as <data_type>
protected dim <name> as <data_type>
```

Example:

```
private dim hiddenvar as integer
```

Description:

Object property variables can be declared as private or protected. A variable that is declared as private can be used throughout the object template but cannot be referenced outside the template (including in any object instance).

A protected variable may be accessed in the object template and in the declaration of an object instance of that template (i.e., in variable initializations and in override functions within the instance). Protected variables may also be accessed by templates that extend (inherit from) the template in which they are defined. No other access is allowed.

2.8 Object References

Object references are variables of type "objref." These variables can refer to an instance of an object. There are two types of object references: typed and untyped.

When writing Qlarity code, the name of an object may be used wherever a reference to the object is desired.

2.8.1 Untyped Object References

Syntax:

```
dim <name> as objref
```

Example:

(assume that the object "mycircle" exists)

```
func untypedobjref( )
   dim anyobject as objref
   init anyobject := mycircle
   [...]
endfunc
```

Description:

Variables of this type cannot be used to access object properties or methods, but they can be used anywhere else that an object name or object reference is required. For example, an objref can be passed to the Attach() API function to attach an object to a parent container. Because they are untyped, these variables can refer to an object of any type.

It is possible to convert an untyped object reference to a typed object reference by converting the untyped object reference to a string then converting the string to a typed object reference.

```
func ConvertToButton(untypedObj as objref)
   dim buttonRef as objref ButtonV2
   dim buttonName as string
   buttonName = str(untypedObj)
   check error
      val(buttonRef, buttonName)
   on error
      _print("Illegal conversion")
      rethrow()
   enderr
   return buttonRef
   [...]
endfunc
```

2.8.2 Typed Object References

Syntax:

```
dim <name> as objref <template_name>
init <name> := <obj_instance>
```

Example:

```
func typedobjref( )
   dim bmpobjref as objref bitmapobj
   init bmpobjref := mybitmap

   bmpobjref.enabled := true
   bmpobjref.foo(10,20)
   [...]
endfunc
```

Description:

Variables of this type can only refer to instances of the named template. For example, if there is an object template "gauge" and the variable "mygauge" is an objref of "gauge," then "mygauge" can only refer to gauge objects and to no other types. In addition to the capabilities of an untyped objref, typed objrefs can be used to access properties and methods of the specified object template.

Typed objrefs are empty by default. If you use an objref that is empty, a runtime error will occur. Use the keyword "empty" to indicate an empty objref.

In the example above, the function has a typed objref of type "bitmapobj" that is used to access the enabled property of the bitmap object instance. The "bitmapobj" method "foo" is also called.

2.8.3 Special Object References

The object reference "me" always refers to the object in which it is used. "Me" is considered a typed object reference. "Default," when used in the context of an untyped object reference, refers to the root container. When "default" is used in the context of a typed object reference, it is the same as "empty." The object reference "empty" indicates that no object is referenced by this value. This is similar to a "null" or a "nil" value in other programming languages.

2.9 Arrays

Syntax:

```
dim <array_name> [ <array_size> ] as ->
<data_type>
init <array_name> := [ <val_0>, ... , ->
<val_n-1> ]
```

Description:

Array declaration is similar to variable declaration. The array size is an integer value. Arrays can be declared of any type except string and unistring. A byte array (or string) may be assigned a string of alphanumeric characters within double quotes. An array that is not initialized is automatically initialized to the default value of the data type.

An array can be initialized with fewer values than the size of the array. In which case, the remainder of the array is initialized to the default value.

The array size is optional. If it is omitted, the array will be sized to the number of elements in the initialization. If the initialization is omitted, the array will remain unsized until it is assigned.

Array elements can be accessed using integer indices in the array as follows:

```
<array_name> [ <index> ]
```

Arrays in Qlarity are zero-based; that is, the index of the first element of an array is 0.

Example:

```
dim myArray[11] as byte
init myArray := "Hello World"
dim anotherArray[4] as integer
init anotherArray := [1, 2, 3, 4]
```

An array can be assigned to another array, as shown in the following example. The array receiving the assignment is automatically resized to the size of the assigned array.

Example:

```
func test( )
   dim a[2], b[2] as integer.
   init a := "hi"
   b = a
' a[ ] is being assigned to b[ ]
endfunc
```

Unlike some languages, arrays that are passed as parameters to a function are not passed by reference unless the array parameter to the function is a reference parameter. Therefore, passing large arrays to functions can be computationally expensive.

2.10 Operators

2.10.1 Arithmetic Operators

Arithmetic operators can be used with any float, integer, byte or unibyte value.

Operator	Description
+	Addition operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
mod	Modulus operator (cannot be used with floats)

2.10.2 Boolean Operators

Operator	Description
not or !	NOT operator[1]
and or &	AND operator[1]
or or |	OR operator[1]
xor or ^	XOR operator[1]
<=	Less than or equal to[2]
>=	Greater than or equal to[2]
<	Less than[2]
>	Greater than[2]
==	Equals operator[3]
<>	Not equal to[3]

1. Can be used with variables, literals, or constants of the following types: boolean, byte, unibyte.

2. Can be used with variables, literals, or constants of the following types: float, integer, byte, unibyte, and arrays of those types. A boolean expression is generated.

3. Can be used with variables, literals, constants, or arrays of any type. A boolean expression is generated.

2.10.3 Assignment Operators

Operator	Description
:=	Strict assignment operator. This must be used anytime a variable or constant is initialized. **No validation is performed on the assignment**. In most cases, this operator should not be used to assign properties of other objects because no validation will be performed.
=	Validation assignment operator. This operator is the same as := unless a validation function with the same name as the variable was written. If this is the case, then a validation is performed. Validation functions are discussed later. Generally, this operator should be used except when initializing a variable or assigning a value inside a validation function.

See section 2.12.3 for an explanation of validation methods.

2.10.4 Dereference Operator

Operator	Description
.	This operator allows access to object properties and methods from outside the object. The dot is placed between the object name or reference and the property or method that will be accessed.

2.10.5 Miscellaneous Operators

Line Continuation

Operator	Description
->	This construct is used to continue a statement on the next line if it is too long. This can increase the readability of your code.

Strings (enclosed in double or single quotes; e.g., "Hello world")

Operator	Description
\n	Newline character
\r	Carriage return

Operator	Description
\xDD	Represents a hexadecimal value where each "D" is 0 through 9 or A through F. You must use two hexadecimal digits; for example, \x00 *not* \x0, \x0A *not* \xA, and so on.
\\	Backslash character
\"	Quotation marks
\'	Apostrophe

2.11 Casting

It is possible to convert a value from one numeric data type to another. While Qlarity will automatically convert data types when necessary, it is occasionally useful to manually cast a value to a new data type. You can cast values to the following built-in data types:

integer
float
byte
unibyte

Additionally, you can cast the keyword "default" to type "objref."

To cast a value, type the data type to which you want to cast, and then enter the value to cast in parentheses. You can use the casting operators as part of any legal expression.

Example:

```
'Calculate the sum of two simulated dice
die1 = integer(6*GetRandomNum()) + 1
die2 = integer(6*GetRandomNum()) + 1
diceTotal = die1 + die2
```

Casting is commonly used to allow integer literals to be used with bitwise operators such as the AND operator:

```
'Calculate the red, green, and blue ->
components for a given color.
'The resultant values range from 0 to 7
rValue = (myColor and byte(0xE0)) / 32
gValue = (myColor and byte(0x1C)) / 4
bValue = myColor and byte(0x03) * 2
```

Casting a value may cause a loss of precision if the target data type is incapable of expressing the original value. For example, the expression "integer(6.7)" produces the value 6 (since integers cannot express fractional values), and the expression "byte(257)" produces the value 1 (since bytes can only express values from 0 to 255).

You may cast the keyword "default" to any of the allowed casting data types. The default value for integer, float, byte, and unibyte is zero (0). You may also cast "default" to an "objref." The value obtained from this cast is the root or global container. This is useful for certain comparisons; for example:

```
if parent == objref(default) then
   transmit (com1, str(me) + ->
      "is attached to ROOT.\n", false)
endif
```

2.12 Functions

Syntax:

```
func <name> ([<var> as <data_type>, ...]) ->
   [returns <data_type>]
   [handles <msg_name>]
   [Variable declarations and ->
      initializations]
   [statements...]
endfunc
```

Example:

```
func circ_area(x as float) returns float
   dim pi as integer
   init pi := 3.1415927
   return x*x*pi
endfunc
```

Description:

A function can contain any number of input parameters (separated by commas) and statements. If the function handles any event messages, the "handles" statement is used. The "handles" statement must be the first statement in the function. The "dim" and "init" statements must be the next statements in a function. The actual order of "dim" and "init" statements is unimportant.

When a message is received by an object, the function that handles the message is called. The "handles" statement specifies the message type that must be sent for the function to be called. There can be more than one "handles" message in a function as long as they appear before any other declaration or statement. See Chapter 3, "Messages and Message Handler Prototypes" for a complete listing of messages.

Variables declared inside the function are local to that function and cannot be accessed from outside the function.

The "returns" clause specifies the type of value that is returned from the function. If a returns type is specified, then the function is required to return a value of that type. The value is passed back from the function using the "return" statement, which also terminates the function. Multiple "return" statements can be included in a function in separate execution paths. The first "return" statement executed will end the function. Return statements with no value may be used to terminate a function with no return value. Functions with no return value have an implied return at the "endfunc" statement.

The actual code for the function is entered between the local variable declarations and initialization (if any) and the "endfunc" statement.

Example:

```
func getChar(data[] as byte) returns boolean
   handles msg_comm_receive
   dim newData[5] as byte
   init newData := [10,20,30,40,50]
   dim countdown, countup as integer
   init countup := 0
   if len(data) == 5
      for countdown = 4 to 0 step -1
         newData[countup] =
             data[countdown]
         countup = countup +1
      next
   endif
      return true
endfunc
```

2.12.1 Calling a Function

Use the following syntax to call a function:

```
<funcname>(<parameter>, ...)
```

Use the following syntax to call an object method:

```
<objname>.<funcname>(<parameter>, ...)
```

The parameters to call a function can be literal values, variables of the correct data type, or expressions that produce the correct data type.

2.12.2 Private, Protected, and Fixed Functions

Syntax:

```
private func <name> (<parameters>)
   [...]
endfunc

protected func <name> (<parameters>)
   [...]
endfunc

fixed func <name> (<parameters>)
   [...]
endfunc

protected fixed func <name> (<parameters>)
   [...]
endfunc
```

Description:

A function can be declared private or protected. A function that is declared private can be used throughout the definition of an object but cannot be overridden and cannot be called from outside the object definition.

A protected function may be accessed in the object template and in the declaration of an object instance of that template (i.e., in override functions within the instance). Protected functions may also be accessed by templates that extend (inherit from) the template in which they are defined. No other access is allowed.

A function can also be declared fixed or protected fixed. A fixed function cannot be overridden. A protected fixed function cannot be overwritten either in an instance of the template or in another template that extends the template which defines the fixed function. Additionally, the protected fixed function, like a protected function, may only be used my instances of the template and by other templates which extend the template.

2.12.3 Validation Methods

A very powerful and useful feature of Qlarity is the validation method. These special functions are tied to variables (usually object properties). When the variable is assigned a value using the validation assignment operator (=), the validation method is implicitly called and passed the new value as a parameter. This allows the object property or global variable to perform validation on the new value (e.g., is the value within allowable bounds?) and to perform other

actions that update the state of the object or variable in the application.

Qlarity objects derive much of their power from this feature. For example, a simple area object might have properties defining the location of the object on the display. The validation methods for the x- and y-position properties might check to make sure that the value is within the boundaries of the screen and call Qlarity API functions to redraw the object at the new location. All of this occurs from simple assignment of a new value to the object property.

A validation function must have the same name as the variable being validated. Also, there can be only one parameter to the function, and it must be of the same data type as the associated variable. Otherwise, these functions have the same syntax as normal functions.

IMPORTANT:
Do not use the validation assignment operator on the validated variable within the validation method! This will recursively call the validation function until the system software gives an exception.

Example:

```
dim x as integer
dim flag as integer
func x (newval as integer)
   if newval > 20 & newval < 100 then
      x := newval
   else
      flag := -1
   endif
endfunc
```

In this example, if the new value being assigned is not between 20 and 100, then flag is given -1 and x retains its old value. If no new value is assigned to the variable during its validation function, then it will retain its old value.

2.12.4 Array Validation Functions

Syntax:

```
func <array_name> (<newarray>[]as ->
<data_type)
   [statements...]
endfunc
```

These functions are called when the entire array is assigned a new value with the validation assignment operator.

Example:

```
dim array1[10], array2[20] as integer
array1 = array2
```

If a validation function for "array1" exists, it will be called and passed to "array2" as a parameter.

Example:

```
dim myarray[10] as integer

func myarray(newarray[] as integer)
   if len(newarray) > 10 then
      myarray := newarray
   endif
endfunc
```

2.12.5 Array Element Validation Functions

Syntax:

```
func <array_name>[] (<new_value> as ->
<data_type>, <index> as integer)
   [statements...]
endfunc
```

Description:

These functions are called whenever an element of an array is assigned a new value with the validation assignment operator.

The "<index>" parameter contains the index of the array that is being assigned a new value.

Example:

```
dim myarray[10] as integer

func myarray[](newval as integer, index ->
as integer)
   if index == 0 then
      if newval > 10 then
         myarray[index] := newval
      endif
   endif
endfunc
```

2.12.6 Reference Parameters

Syntax:

```
func <func_name> (<varname> as reference ->
to <datatype>)
   [...]
endfunc
```

Description:

Qlarity normally uses a pass by value convention, which means that functions receive a copy of variables passed as parameters. They cannot modify the value of the original variable. Reference parameters make it possible to modify the values of the variables that are passed into a function as parameters. These are especially useful when a function needs to return more than one value.

Example:

```
func myFunc(newVar as reference to integer)
   newVar=5
endfunc
```

When "myFunc" is executed, a reference to the variable is passed into the function. The value of "newVar" reflects the change after the function returns.

2.13 Conditionals (if Statement)

Syntax:

```
if <boolean_expr> then
   [...]
[elseif <boolean_expr> then]
   [...]
[else]
   [...]
endif
```

The "then" is required. The "elseif" and "else" statements are optional. There is no limit to the number of "elseif" clauses that can be added to the "if" statement

Example:

```
dim x as integer
if x < 10 then
   x = x +1
elseif x == 10 then
   x=x+2
else
   x=x-2
endif
```

2.14 Looping and Leaping

2.14.1 For/Next loops

Syntax:

```
for <name> = <start_expression> to ->
<stop_expression> [step <expression>]
   [...]
next
```

Description:

The "for" loop executes starting with "<name> = <start_expression>" and increments "<name>" by the amount specified in "step" until "<name>" is equal to "<stop_expression>." "Step" may be positive or negative. If no "step" is specified, "<name>" is incremented by 1. When "<name> = <stop_expression>," the loop executes for the last time.

If "<name>" has a validation function, it is called each time through the loop. When the loop terminates, "<name>" will have been incremented or decremented beyond "<stop_expression>." (In the example below, the final value for "countdown" is -1.) Also, "<stop_expression>" is evaluated each time through the loop. If "<stop_expression>" is complex or involves a function call, and the value of "<stop_expression>" does not change, it is recommended that you store the value of "<stop_expression>" in a variable and use that for "<stop_expression>."

Example:

```
dim Data[5] as byte
init Data := [10, 20, 30, 40, 50]
dim stepamount as integer
init stepamount := -1
dim countdown, countup as integer
init countup := 0

for countdown = 4 to 0 step stepamount

   newData[countup] = data[countdown]
   countup = countup +1
next
```

2.14.2 While Loops

Syntax:

```
while <boolean_expression> do
   [...]
loop
```

Description:

The statements inside the "while" loop are executed until the boolean expression becomes false. If the boolean is false before the "while" loop starts, then the loop is not executed.

2.14.3 Do/While Loops

Syntax:

```
do
   [...]
loop while <boolean_expression>
```

Description:

The "do/while" loop is similar to the "while" loop except that it will always execute at least once. The boolean expression is evaluated each time execution reaches the end of the loop. When the expression becomes false, the loop is finished.

2.14.4 Goto/Label

Syntax:

```
goto <label_name>

label <label_name>
```

Description:

The "goto" command is used to stop executing the code at the "goto" command and begin where the "<label_name>" is specified.

The "label" command is used to mark a spot to be jumped to with the "goto" command.

These two statements must appear within the same function. Jumping between functions is not allowed. Also, you cannot jump into, out of, or between different "check error" blocks. It is generally good coding practice to avoid "goto" whenever possible.

2.15 Exception Handling

Syntax:

```
check error
   [...]
on error
   [...]
enderr
```

Description:

The purpose of the "check error" block is to check for an error and then resolve the error. The code between "check error" and "on error" is executed normally. If an exception is thrown by the system or by using the Throw() API function, then execution immediately jumps to the first statement in the "on error" block with no option to return to the previous execution point.

Within the "on error" block, the application can retrieve the last exception using the GetException() API function. Code in the "on error" block is intended to handle exceptions in a manner defined by the application. If the "on error" block doesn't resolve the exception, it can call Rethrow() to allow a higher level "check error" to handle it.

If no "check error" block appears in the function, the error "bubbles up" to the function that called it and eventually out to the system where MSG_ERROR functions can handle it.

For more detailed information on exception handling, refer to the section onf "Exception Handling in the *Qlarity Foundry User's Manual*.

2.16 Defining Objects

Syntax:

```
define <object_type> type <name> [extends ->
   <Name>]
   [...]
enddef
```

Description:

An object type must be defined in order for an instance of that object to be declared. Object properties are defined using "dim" statements. Default values are assigned using "init" statements. Methods are defined by putting function definitions inside the definition of the object. Any instances of an object type will have these properties and functions.

"<object_type>" may be any of the three object type keywords: "object," "area object," or "container." "<name>" may be any legal and unique identifier and becomes the name of the defined object type.

"Extends" indicates that this template will extend the template <name>, where <name> is the name of the template this one will extend. By extending a template, the new template will inherit all methods and properties of the base template.

Extending templates via inheritance can be a powerful tool to allow you to reuse existing code and create modified objects quickly. When extending templates there are several items to remember.

- The extending template receives all of the base template's variables and functions. Care should be exercised not to declare any variables or functions with the same name as those in the base template unless the extending template is overriding those elements
- The extending template may not access the base template's private variables and functions
- The extending template may override any variable declaration, variable initialization or function definition.
- When overriding a variable or function, the extending template may reduce the access to the element. For instance, a template may override a protected variable declaration in a base template as private. Conversely, it is not allowed to increase access to an element in the extending template. For example, you could not override a private function in a base template as protected in the extending template.
- An extending template may add validation functions for variables in a base template. The compiler will issue a warning for this unless you also override the variable in the extending template.
- An extending template may add variables that would be validated by functions in the base template. As this is usually undesired behavior, the compiler will issue a warning in this case. To avoid the warning either rename the variable or override the validation function in the extending template.

Chapter 10 in the *Qlarity Foundry User's Manual* gives examples of defining an object type and creating instances of the object.

2.17 Declaring Object Instances

Syntax:

```
declare <name> as <defined_type>
   [...]
enddec
```

Description:

Once an object type is defined, instances of that object may be created by declaring them.

"<name>" may be any legal unique identifier; it is used to refer to the instance of the object. "<defined_type>" is the name of the object type, and it specifies the template to be used when creating the object instance.

All functions and properties defined for an object type will be included in any instance of that type.

Variables may not be created using a "dim" statement inside an object instance, but they may be given initial values for that instance using "init" statements.

Functions declared in an instance must also have been declared in the object template; they will override the functions in the template. The function name, events handled, parameters, name, type, and return type must match the declaration of the function in the template.

Chapter 10 in the *Qlarity Foundry User's Manual* gives examples of defining an object type and creating instances of the object.

2.18 Level

Syntax:

```
Level <integer>
```

Description:

The "Level" command specifies the Z-order of an instance at application startup. Instances with a higher level appear higher (towards the front) in the Z-order. If two instances have the same level, then their Z-order relative to each other is undefined. This directive must immediately follow a "declare" statement. Qlarity Foundry normally sets this directive based on the object's location in the Object Tree.

This command is useful when writing Qlarity applications in a text editor. Qlarity Foundry typically ignores any Level commands typed in the Code View window.

2.19 Including Files and Resources

Syntax:

```
include "<file path>"
```

Description:

This includes the specified file to be compiled with your project. The "include" command is also used to make bitmaps, ttfonts, and bdffonts available as resources in the application.

Syntax:

```
include bitmap "<bitmap_file_path>" as ->
<name>
```

```
include ttfont "<ttfont_file_path>" as ->
<name>
include bdffont "<bdffont_file_path>" as ->
<name>
```

Description:

The "<name>" field is any name to be used to refer to the included bitmap, ttfont, or bdffont. If the keyword default is used as the name of the resource, then that bitmap, ttfont, or bdffont becomes the default resource of that type. Resource variables that are not explicitly initialized will receive the default resource.

Since the backslash (\) character is a special character used in strings, it is necessary to use either a single forward slash (/) or a double backslash (\\) to separate directories in file paths.

2.20 Libraries

Libraries are similar to other files that can be included except they may contain more than just source code. A library may contain any of the following:

- Qlarity source code
- Native source code
- Bitmaps
- BDF fonts
- TrueType fonts
- Modules

To include a library:

Syntax:

```
Include library "<file_name>"
```

The library "natives.lib" is automatically included by the compiler. Do not manually include this file in your project. See Chapter 10 of the *Qlarity Foundry User's Manual* for more information on libraries.

Including a library simply informs the compiler of the existence of the library. To use the code or other entries in a library, you must reference them explicitly, as shown below.

Syntax:

```
Library <library name><entry type><entry ->
name>
```

Description:

"<library name>" is the name of the library in which the entry resides. This is generally different from the library file name. "<entry type>" is the type of entry in the library: source, bitmap, bdffont, ttfont, native, or module. "<entry name>" is the name of the library entry.

By referencing the library entries explicitly, you can use only the entries you want. Qlarity Foundry may bring in certain library entries automatically. To determine the name of a library and the names and types of entries in it, refer to the documentation that came with the library, or view the library in Qlarity Foundry.

2.21 Precompile Directives

2.21.1 #if/#else/#endif

Syntax:

```
#if <option_name>
   [statements...]
#else
   [statements...]
#endif
```

Description:

"#if" statements are used by the compiler to determine if an option has been declared. If so, the code between the "#if" and the "#else" or "#endif" is compiled into the project. Otherwise, the enclosed code is not compiled. Code in the "#else" section is compiled only if the option was not declared.

#ifnot/#else/#endif is also a legal construct, where the code between the #if and the #else or #endif is compiled if the option is not declared. Otherwise, the enclosed code is compiled. Code in the #else section is compiled only if the option was previously declared.

2.21.2 #option

Syntax:

```
#option <option_name>
```

Description:

This function is used to define a keyword that will be checked by the "#if" statement. **Unlike Qlarity variables, a "#option" declaration for a given keyword must be used before it can be examined by the "#if" statement.**

A common predefined option is "_TOOL," which is defined if Qlarity Foundry is currently running the application. Code that should only run in Qlarity Foundry (and be excluded from the application that runs in the terminal) should be enclosed in an "#if _TOOL/ #endif" block.

2.21.3 #Toolimage

Syntax:

```
#Toolimage <string of hexadecimal digits>
```

Description:

This directive defines the icon used by a template and instances of the template in Qlarity Foundry. Qlarity Foundry automatically generates and updates the "#Toolimage" line for each template in a workspace. The line with the "#Toolimage" directive must follow a template definition statement, and it must appear before any code in the template. Manually editing a "#Toolimage" line is not recommended. This directive is ignored when a workspace is compiled with the command line compiler.

2.21.4 #Hidden

Syntax:

```
#Hidden <dimension statement>
```

Description:

This directive informs the Qlarity Foundry compiler that the variables declared in "<dimension statement>" should not be displayed in the Object Properties list. This only applies to global variables and object properties; it has no effect on local variables in a function. This directive is ignored when a workspace is compiled with the command line compiler.

2.21.5 #Setfile

Syntax:

```
#Setfile <line number> <filename>
```

Description:

This directive tells the compiler to change file names and line numbers for error reporting. The line number is immediately set to "<line number>." Qlarity Foundry uses this directive internally and ignores any such directives typed in by the user. Altering these directives is not recommended.

2.21.6 #Visible

Syntax:

```
#Visible <true or false>
```

Description:

This directive informs Qlarity Foundry about the visible state of an object (i.e., whether an object should be displayed or hidden). Normally, Qlarity Foundry inserts or alters this directive whenever you click on the "eye" icon in the Object Tree. This directive is ignored when a workspace is compiled with the command line compiler.

2.21.7 #Lock

Syntax:

```
#Lock <true, false, or me>
```

Description:

This directive informs Qlarity Foundry about the lock state of an object (i.e., whether an object or its children can be altered in Qlarity Foundry). Normally, Qlarity Foundry inserts or alters this directive whenever you click on the "lock" icon in the Object Tree. This directive is ignored when a workspace is compiled with the command line compiler.

2.21.8 #STPBuilderApp

Syntax:

```
#STPBuilderApp
```

Description:

This directive appears on the first line of a workspace that has been saved by Qlarity Foundry. It is ignored when a workspace is compiled with the command line compiler.

2.21.9 #endfile

Syntax:

```
#endfile
```

Description:

This directive instructs the compiler to immediately cease processing the current file. Any code after the "#endfile" directive is ignored.

CHAPTER 3

MESSAGES AND MESSAGE HANDLER PROTOTYPES

In Qlarity, functions that handle messages must fit a specified format, or number and type of parameters, and the type of return value. The message handler formats may vary depending on whether the message is being handled by a global function, a function inside a container type object, or a function inside an area or non-drawable object.

Message handler functions that return a boolean value can typically terminate the message (stop it from moving forward through the message handling system) by returning a value of "true." Returning a value of "false" allows the message to continue on through the message handling system.

Descriptions of all system messages are provided in this chapter. Included is the format (number and types of parameters and return type) used to declare a handler for each message.

3.1 Broadcast Messages

Broadcast messages are sent to all objects (regardless of their enabled or disabled status) according to the Z-order. A message is first passed to the root container (where it is possibly handled by global functions), which then passes the message to each of its children, beginning with the front-most object (highest Z-order). When the message is passed to a container object, the container object first handles the message then passes it to each of its children, beginning with the front-most object. In this manner, the message filters down through the object hierarchy until all objects have received the message.

Broadcast message handlers do not have a return value and, therefore, cannot terminate the message.

Message:

MSG_INIT

Description:

This message is sent to all objects on system startup. It is the first message generated in the system. It allows each object to synchronize its system software attributes (enabled, parent, position, size, etc.) with its properties and to perform other startup functions. Do not rely on the order of reception for the init message (don't assume others are already inited).

Handler Format:

```
func <name>( )
   handles MSG_INIT
   [...]
endfunc
```

MSG_INIT handlers take no parameters and do not have a return value.

3.2 Area Messages

Area messages are associated with a given area, or point, (X,Y coordinate) on the terminal display. A message is first passed to the root container (where it is possibly handled by global functions), which then checks for any enabled children that contain or overlap the area of the message. This checking is done in Z-order, starting with the front-most object. If a child is eligible to receive the message, the message is passed to the child. When the message is passed to a container object (including the root container), it first handles the message then does a similar check for enabled children that contain or overlap the message area. After all of a container's children have received the message, the handler in the container (if any) is called a second time. In this manner, the message filters down through and back up the object hierarchy until all eligible objects have received it.

Global area message handlers are considered to be methods of the root container. They are therefore eligible to receive a message twice (once before any of the root's children and once after all of root's children have received the message). Only area and container objects can receive area messages. Non-drawable objects are not eligible to handle them.

The parameters of an area message handler indicate the location of the event on the screen. Area message handlers have a boolean return value used to terminate the message. Returning a value of "true" terminates the message, while returning a value of "false" allows the message to continue through the message handling system.

The format for the handler is different for container and area objects. Containers have an additional boolean parameter. The value of this parameter is "false" when the container handler is called before the message passes to the container's children. When the handler is called after the container's children have received the message, the value of this parameter is "true." A container, such as a form object, that doesn't handle an area message directly (e.g., a screen press) generally returns "false" the first time the handler is called. This allows the container's children to handle the message. However, "true" is returned on the second call to prevent objects behind the container (and therefore hidden) from receiving the message.

Message:

MSG_SCREEN_PRESS

Description:

This message indicates that a touch screen press event has occurred.

Handler Format:

(if handled by a container object/global)

```
func <name>(<x> as integer, <y> as ->
integer, pass as boolean) returns boolean
   handles MSG_SCREEN_PRESS
   [...]
endfunc
```

(if handled by an area object)

```
func <name> (<x> as integer, <y> as ->
integer, returns boolean
   handles MSG_SCREEN_PRESS
   [...]
endfunc
```

Parameters:

<x>
The x-location of press event (with respect to the parent's origin)

<y>
The y-location of press event (with respect to the parent's origin)

<pass>
False if the container's children have not yet received the message, and true if the container's children have already received the message.

Message:

MSG_SCREEN_RELEASE

Description:

This message indicates that a touch screen release event has occurred.

Handler Format:

(if handled by a container object/global)

```
func <name> (<x> as integer, <y> as ->
integer, pass as boolean) returns boolean
   handles MSG_SCREEN_RELEASE
   [...]
endfunc
```

(if handled by an area object)

```
func <name> (<x> as integer, <y> as ->
integer returns boolean
   handles MSG_SCREEN_RELEASE
   [...]
endfunc
```

Parameters:

<x>
The x-location of press event (with respect to the parent's origin)

<y>
The y-location of the press event (with respect to the parent's origin)

<pass>
False if the container's children have not yet received the message, and true if the container's children have already received the message.

Message:

MSG_SCREEN_MOVE

Description:

This message indicates that the user has moved his or her finger across the touch screen without lifting it. Objects whose area contains the starting or ending (or both) coordinates receive this message.

Handler Format:

(if handled by a container object)

```
func <name> (<x_to> as integer, <y_to> as ->
integer, <x_from> as integer, <y_from> as ->
integer, <pass> as boolean) returns boolean
   handles MSG_SCREEN_MOVE
   [...]
endfunc
```

(if handled by an area object)

```
func <name> (<xto> as integer, <yto> as ->
integer, <xfrom> as integer, <yfrom> as ->
integer) returns boolean
   handles MSG_SCREEN_MOVE
   [...]
endfunc
```

Parameters:

<x_to>
The final x-location after the move (with respect to the parent's origin).

<y_to>
The final y-location after the move (with respect to the parent's origin).

<x_from>
The initial x-location before the move (with respect to the parent's origin).

<y_from>
The initial y-location before the move (with respect to the parent's origin).

<pass>
False if the container's children have not yet received the message, and true if the container's children have already received the message.

3.3 Draw Messages

A draw message cannot be terminated, and it is passed to containers (including the root container) twice, as described in section 3.2, "Area Messages." This allows the container to draw borders around itself or perform other actions that should take place after the container's children have finished handling the message.

Unlike area messages, the actual invalid region is not passed to the handler as a set of parameters. Rather, the handler should just repaint the object entirely, and the system software clips the repainted region to the invalid region.

Non-drawable objects are not eligible to receive this message.

Draw messages are handled by the system as a low priority. Draw messages are placed in a separate message queue that is serviced when the regular message queue is empty. Also, the system software may combine multiple invalid regions into a single draw message to increase drawing efficiency.

Message:

MSG_DRAW

Description:

This message indicates that the receiving object needs to paint itself on the screen.

Handler Format:

(if handled by a container object)

```
func <name> (<pass> as boolean)
   handles MSG_DRAW
   [...]
endfunc
```

(if handled by an area object)

```
func <name>()
   handles MSG_DRAW
   [...]
endfunc
```

Parameters:

(if handled by a container object)

<pass>
False if the container's children have not yet received the message, and true if the container's children have already received the message. A container should draw its background when "pass = false" and its borders when "pass = true."

(if handled by an area object)

No parameters.

3.4 Registered Messages

Registered messages are associated with system events such as a serial character receive, network packet receive, or

timer tick. Event handlers that process these messages are registered with the system software using registration functions in the Qlarity API. Because these events are usually handled by a small number of functions, registration avoids the need to filter the message through the entire object hierarchy. If the handler in the registration list is an object method, the object owning the method is checked by the system software to ensure that it and every parent container (back to the root container) are enabled. If this condition is satisfied, the event handler is called. Because the root container cannot be disabled, global function event handlers are called without this check. If more than one object has registered for a given message, the objects receive the message in Z-order (as with area messages) with the root container (global handlers) receiving highest priority.

Objects with handlers for registered messages may be registered to receive these messages by calling the RegisterMsgHandler() API function. This is typically done in the MSG_INIT handler for the object, but it can be done at any time. Each different type of message must be registered individually for a given object.

Key handlers are registered using the RegisterKey() API function. Objects may be registered for a specific key or for all keys (using the parameter KEY_ALL). The actual keycode is passed to the handler as a parameter, so a handler registered for KEY_ALL can determine what key was pressed or released. A list of constants defining all keys for the keyboard or keypad is included in Appendix A, "Built-in Constants and Defined Types".

Registered objects may also be unregistered by calling the UnregisterMsgHandler() API function, after which the object will no longer receive the message. To unregister a key handler, register the object with the parameter KEY_NONE.

The key messages MSG_KEY_DOWN, MSG_KEY_PRESS, and MSG_KEY_RELEASE are similar to area messages in that container objects receive the message twice: once before it is passed to the container's children, and once after all of its children have received it. A boolean parameter is used to indicate which pass is taking place when the handler is called. The parameter value is false if the children have not yet received the message, and true if the children have already received it.

Also similar to area message handlers, registered message handlers return a boolean value that is used to terminate the message. Returning a value of "true" from the handler terminates the message, while returning a value of "false" allows the message to continue on through the system.

NOTE:
The messages MSG_COMM_RECEIVE, MSG_COMM_RECEIVE_URGENT and MSG_COMM_RECEIVE_MULTICAST are all registered for at the same time. In other words, if an object registers for one of these messages, then it registers for all of them.

Message:

MSG_COMM_RECEIVE

Description:

This message indicates that one or more characters have been received from a communications interface (COM port or Ethernet port). You can register for multiple communications resources. As a result, you will receive this message each time data is received from any one of the resources for which you are registered. Use the GetComMessageSource API to determine the communications resource that resulted in the generation of this message.

Handler Format:

```
func <name> (<data>[] as byte) returns ->
boolean
   handles MSG_COMM_RECEIVE
   [...]
endfunc
```

Parameters:

<data>
A byte array containing the data that was received from the communications interface.

Message:

MSG_COMM_RECEIVE_URGENT

Description:

This message indicates that one or more characters have been received via the Transmission Control Protocol (TCP) urgent or out-of-band data channel. This message is only generated by a TCP communications resource. You can register for multiple communications resources. As a result, you will receive this message each time data is received from any one of the resources for which you are registered. Use the GetComMessageSource API to determine the communications resource that resulted in the generation of this message.

Handler Format:

```
func <name> (<data>[] as byte) returns ->
boolean
   handles MSG_COMM_RECEIVE_URGENT
   [...]
endfunc
```

Parameters:

<data>
A byte array containing the data that was received via a TCP urgent data channel.

Message:

MSG_COMM_RECEIVE_MULTICAST

Description:

This message indicates that a packet has been received from a multicast communications interface. You can register for multiple communications resources. As a result, you will receive this message each time data is received from any one of the resources for which you are registered. Use the GetComMessageSource API to determine the communications resource that resulted in the generation of this message.

Handler Format:

```
func <name> (<data>[] as byte, <lport> as ->
unibyte, <fport> as unibyte, <ip>[] as ->
byte) returns boolean
    handles MSG_COMM_RECEIVE_MULTICAST
   [...]
endfunc
```

Parameters:

<data>
A byte array containing the information that was received from the communications interface.

<lport>
The local port number on the unit that the received data was destined for.

<fport>
The foreign port number on the host that was the origin of the data.

<ip>
The IP address of the host that was the origin of the data.

Message:

MSG_COMM_TRANSMIT

Description:

This message indicates that one or more characters are to be transmitted by the communications interface. (EIA-232, -422, -485, Ethernet, etc.) This message is generated by calling the Send() API function, and it is useful for implementing a protocolizing function (adding a header, tail, checksum, etc.) for transmitted data in a single location. Handlers for this message should call the Transmit() API function to actually send the data.

Handler Format:

```
func <name> (<data>[] as byte) returns ->
boolean
   handles MSG_COMM_TRANSMIT
   [...]
endfunc
```

Parameters:

<data>
A byte array containing the data that is to be transmitted by the communications interface.

Message:

MSG_TIMETICK

Description:

This message indicates that a system timer has expired. When registering for this message, an object specifies the interval between messages in 20 millisecond increments. The minimum interval is 40 milliseconds. A MSG_TIMETICK message is sent to the registered object at each time interval.

Handler Format:

```
func <name> ( )
   handles MSG_TIMETICK
   [...]
endfunc
```

Handlers for MSG_TIMETICK have no parameters and no return value.

Message:

MSG_KEY_DOWN

Description:

This message indicates that a key has been pressed on the keyboard or keypad. The keyboard and keypad are distinguished by unique keycodes. If the key repeat feature is available on the keyboard or keypad and the key is held down, then the MSG_KEY_DOWN will only occur on the initial press (not as the key is repeated). Use this message to determine a key's state (up or down); use MSG_KEY_PRESS to determine the character typed (e.g., for a text editor). To be eligible to process this message, an object must register for it using the RegisterKey() API function (see section 4.2.3).

Handler Format:

(if handled by a container object/global)

```
func <name> (<keycode> as unibyte, <pass> ->
as boolean)
   handles MSG_KEY_DOWN
   [...]
endfunc
```

(if handled by an area/non-drawable object)

```
func <name> (<keycode> as unibyte)
   handles MSG_KEY_DOWN
   [...]
endfunc
```

Parameters:

<keycode>
A unibyte containing the keycode for the key that was pressed. Most keycodes are defined as constants, which are listed in Appendix A.

<pass>
False if the container's children have not yet received the message, and true if the container's children have already received the message.

Message:

MSG_KEY_PRESS

Description:

This message indicates that a key has been pressed on the keyboard or keypad. The keyboard and keypad are distinguished by unique keycodes. If the key repeat feature is available on the keyboard or keypad and the key is held down, then the MSG_KEY_PRESS will occur for the initial key (immediately after the MSG_KEY_DOWN message is sent) and for each key repeat event. To be eligible to process this message, an object must register for it using the RegisterKey() API function (see section 4.2.3).

Handler Format:

(if handled by a container object/global)

```
func <name> (<keycode> as unibyte, <pass> ->
as boolean)
   handles MSG_KEY_PRESS
   [...]
endfunc
```

(if handled by an area/non-drawable object)

```
func <name> (<keycode> as unibyte)
   handles MSG_KEY_PRESS
   [...]
endfunc
```

Parameters:

<keycode>
A unibyte containing the keycode for the key that was pressed. Most keycodes are defined as constants, which are listed in Appendix A.

<pass>
False if the container's children have not yet received the message, and true if the container's children have already received the message.

Message:

MSG_KEY_RELEASE

Description:

This message indicates that a key has been released on the keyboard or keypad. The keyboard and keypad are distinguished by unique keycodes. This message is unaffected by the key repeat feature. To be eligible to process this message, an object must register for it using the RegisterKey() API function (see section 4.2.3).

Handler Format:

(if handled by a container object/global)

```
func <name> (<keycode> as unibyte, <pass> ->
as boolean)
   handles MSG_KEY_RELEASE
   [...]
endfunc
```

(if handled by an area/non-drawable object)

```
func <name> (<keycode> as unibyte)
```

```
    handles MSG_KEY_RELEASE
    [...]
endfunc
```

Parameters:

<keycode>
A unibyte containing the keycode for the key that was pressed. Most keycodes are defined as constants, which are listed in Appendix A.

<pass>
False if the container's children have not yet received the message, and true if the container's children have already received the message.

3.5 User Messages

Although hardware events are generated by the system software, it is often desirable for the application to have the ability to send messages to indicate software events. Qlarity allows you to define user messages and a flexible set of Qlarity API functions to send these messages.

3.5.1 Defining User Messages

Syntax:

```
constant message <msg_name>
```

Description:

This statement declares a user message named "<msg_name>." The name can be any legal identifier; however, it is advisable to use a common convention for naming user messages, such as UMSG_XXXX.

Like other user-defined types, this declaration must appear in the global code area. Messages cannot be declared inside object definitions or instances.

3.5.2 Sending User Messages

User messages are sent using the UserBroadcastMsg(), UserSendMsg(), and UserDirectMsg() API functions. These functions are described in detail in section 4.12, "User Message Functions" in this manual.

Messages sent with UserBroadcastMsg() behave as broadcast messages. They go to all objects (enabled or disabled) in Z-order and cannot be terminated.

Messages sent with UserSendMsg() behave as area messages. They are passed to eligible (enabled) objects in Z-order and may be terminated with a handler return value of "true." Unlike area messages, these user messages are not sent to container objects twice. Container handlers are called before the message is passed to the container's children.

A message sent with UserDirectMsg() goes to the object regardless of the object's enabled status.

UserBroadcastMsg(), UserSendMsg(), and UserDirectMsg() have a boolean parameter that determines whether the current message processing is suspended while the user message is handled by the messaging system. Passing a value of "true" will suspend the current processing until the user message has been processed. Processing of the suspended message then resumes at the statement following the API call to send the user message. Passing a value of "false" in this parameter causes the message to be enqueued as any other message. Any exceptions occurring during immediate handling filter up to where the message was initiated.

All user messages carry a single integer parameter that can be used for any desired purpose. The integer is passed as a parameter to the particular API function and is received as a parameter in the handler function.

3.5.3 Handlers for User Messages

Description:

This message indicates that a user message named "<msg_name>" has been sent.

Handler Format:

```
func <name> (<intdata> as integer) ->
returns boolean
    handles msg_name
    [...]
endfunc
```

Parameters:

<intdata>
An integer that can be used for any desired purpose.

The meaning of the return value depends on how the message was sent, as follows:

Broadcast
The return value is ignored.

User
A return value of true = kill message; false = continue.

Direct
If immediate processing was specified, the return value is the return value of the UserDirectMsg() API function. If not immediate, the return value is ignored.

3.6 Direct Messages

Direct messages are sent by the system to a specific object. They are similar to a direct user message except that they are sent by the system rather than the user application. The messages are enqueued in the message queue as are most other messages, but they are only passed to a single object when they are handled. The object must have a handler for the message.

Message:

MSG_COMM_ACCEPT

Description:

This message is generated as the result of a successful call to the NetOpen() API function. When the network channel is opened, the associated communications resource is passed to the object that called NetOpen() via this message. The communications resource may then be used to transmit data using the Send(), Transmit(), and (for TCP channels) TransmitUrgent() API functions. The resource may also be used to register for the MSG_COMM_RECEIVE message.

Handler Format:

```
func <name> (<socket> as comm)
   handles MSG_COMM_ACCEPT
   [...]
endfunc
```

Parameters:

<socket>
A value of type "comm" that is associated with the opened network communications channel. (See section 4.1, "Communications Interface".)

Message:

MSG_COMM_ERROR

Description:

This message is generated when errors are detected with a network channel. An error code and an error message are passed to the object that opened the channel. In general, the network channel should be closed with a call to NetClose() in the MSG_COMM_ERROR handler. If necessary, the comm resource can be obtained with a GetComMessageSource() API call.

Handler Format:

```
func (<errcode> as integer, <errmsg>[] as ->
byte)
   handles MSG_COMM_ERROR
   [...]
endfunc
```

Parameters:

<errcode>
An integer indicating what type of error occurred.

<errmsg>
A byte array containing an error message string.

Message:

MSG_ZENABLED

Description:

This message is a direct message sent by the system software to notify an object whether or not it is enabled back to root (this does not include the object's enabled status). Any changes in the object's enabled path to root generate this message.

Handler Format:

```
func <name> (<status> as boolean)
   handles MSG_ZENABLED
   [...]
endfunc
```

Parameters:

<status>
Indicates whether the path to root is enabled (true = enabled to root). This does not take into account whether the object receiving the message is enabled.

Message:

MSG_SOUND_DONE

Description:

This message is issued by the system software when a sound event (note or sound) is generated by the PlayNoteNotify or PlaySoundNotify API functions has been completed. The object that receives this message is determined by the <obj> parameter passed to the PlayNoteNotify and PlaySoundNotify API functions. The <parm> parameter

allows the application to determine which sound has completed.

Handler Format:

```
func <name> (<remaining> as integer, ->
<parm> as integer)
   handles MSG_SOUND_DONE
   [...]
endfunc
```

Parameters:

<remaining>
Number of sound events (notes and sounds) that remain to be played by the system.

<parm>
The identifier that was passed to the PlayNoteNotify or PlaySoundNotify API functions as the <parm> parameter.

3.7 Tool Messages

Because objects may be completely defined by the user, Qlarity Foundry has no knowledge about the appearance or behavior of a given object. Therefore, the object itself must define its behavior for Qlarity Foundry development activities such as drag-and-drop, resize by dragging resize grips on a selected object, and so on. Tool messages are generated by the Qlarity Foundry system software to tell an object that these activities are taking place. The object must have message handlers for these events to properly function in Qlarity Foundry.

Tool message handlers must be enclosed in "#if _TOOL/ #endif" compiler directives. This only includes the handlers used during development with Qlarity Foundry.

Since the tool message handlers usually change the properties of an object, the system software must be informed that these properties have changed so that it can update its object data structures. This is the purpose of the Tool_Persist() API function. Tool_Persist() is a special API function (available only in Qlarity Foundry) that takes a property name as its only parameter. If Tool_Persist() is not called when a property is changed by a message handler, the change is not properly recorded in the application. Therefore, Tool_Persist() must be called once for each changed object property in a tool message handler.

If you use Qlarity Foundry to create the object, you have the option to create an object that already handles these messages in a manner appropriate for most types of objects. Refer to Chapter 10, "Advanced Design," in the *Qlarity Foundry User's Manual* for sample implementations of the functions. If you elect to have Qlarity Foundry generate an object "ready to operate in Qlarity Foundry," handlers appropriate for most objects are included.

Message:

MSG_TOOL_ATTACH

Description:

This message is sent by Qlarity Foundry when an object needs to be attached to a parent container, usually due to mouse activity in the object hierarchy window. The handler should call the Attach() API function using "<newparent>" as the new parent.

Handler Format:

```
func <name> (<newparent> as objref)
   handles MSG_TOOL_ATTACH
   Attach (me,<newparent>)
   [...]
endfunc
```

Parameters:

<newparent>
A reference to the container to which the object should attach itself.

Do not assign the "parent" property to "newparent" (i.e., do not do "parent := newparent"). This will be done in the MSG_TOOL_ATTACHED handler.

This handler code will suffice for almost any conceivable object, because Qlarity Foundry first calls "attach," but the object doesn't have to attach.

Message:

MSG_TOOL_ATTACHED

Description:

When API functions that change Z-order [such as Attach()] are called, the requested change is scheduled. Because Z-order is critical to the proper functioning of the message handling system, the actual changes are postponed until the current message has been processed to completion. Therefore, it is not possible to know if an Attach() succeeded until the handler that called Attach() has finished executing. If the attach fails, properties that indicate an object's parent should not be changed. If the attach succeeds, then the property needs to be updated.

The MSG_TOOL_ATTACHED message is sent when a successful attachment has occurred. The handler is responsible for updating any property that is affected by a change in parent.

Handler Format:

```
func <name> (<newparent> as objref)
   handles MSG_TOOL_ATTACHED
   parent := newparent
   tool_persist(parent)
   [...]
endfunc
```

Parameters:

<newparent>
A reference to the container to which the object should attach itself.

This handler code works for any object with an objref property named "parent" that holds a reference to the object's parent. Note that the strict assignment operator is used (to avoid calling a validation function that will likely call the Attach() API) and that the Tool_Persist() API function is called to inform the system software that the value of the parent property has changed.

Message:

MSG_TOOL_MOVE

Description:

This message indicates that the object has been moved in Qlarity Foundry (probably due to a mouse drag). The handler should update any properties relating to object position, call API functions to update the system software position attributes for the object, and call Tool_Persist() for any properties that have changed.

Handler Format:

```
func <name> (<dx> as integer, <dy> as ->
integer)
   handles MSG_TOOL_MOVE
   xpos = xpos + dx
   ypos = ypos + dy
   tool_persist(xpos)
   tool_persist(ypos)
   [...]
endfunc
```

Parameters:

<dx>
The change in the object x-position caused by the move.

<dy>
The change in the object y-position caused by the move.

This code assumes that the object has properties "xpos" and "ypos," which hold the current position of the object. Normally, the validation functions for these properties will handle updating the system software attributes, so "xpos" and "ypos" should be assigned using the validation assignment operator. The Tool_Persist() function is called once for each property that was changed.

Message:

MSG_TOOL_GETHANDLES

Description:

This message indicates that the object has been selected in Qlarity Foundry, causing sizing handles (resize grips) to appear on the outline of the object. The object designer has significant freedom to specify the number and location of the resize grips and the cursor that appears as the mouse cursor is positioned over the grip. The handler is passed references to three arrays that must be filled with the x-position, y-position, and desired cursor, respectively, for each grip. The handler should size the arrays to the number of grips that will be displayed on the object.

Handler Format:

```
func <name> (<xCoords>[ ] as reference to ->
integer, <yCoords>[ ] as reference to ->
integer, <cursors>[ ] as reference to ->
GuiCursors, <closed> as reference to ->
boolean)
   handles MSG_TOOL_GETHANDLES
   [...]
endfunc
```

Parameters:

<xcoords>[]
An array that defines the x-position of each grip.

<ycoords>[]
An array that defines the y-position of each grip.

<cursor>[]
An array that defines the cursor type for each grip.

<closed>
A boolean value that determines whether a selection outline segment should be drawn between the last grip and the first grip (outline segments are always drawn between all other grips). A value of "true" causes the outline segment to be

drawn. This also indicates whether clicking inside the object will select it.

GuiCursors is a standard defined type with the following possible values:

GuiCursors	Description
CSR_UPDOWN	Cursor has arrows pointing up and down. The resize grip can only be dragged vertically.
CSR_LEFTRIGHT	Cursor has arrows pointing left and right. The resize grip can only be dragged horizontally.
CSR_UPLEFT	Cursor has arrows pointing diagonally up and left and down and right. The resize grip can be dragged in any direction.
CSR_UPRIGHT	Cursor has arrows pointing diagonally up and right and down and left. The resize grip can be dragged in any direction.
CSR_DOWNLEFT	Cursor has arrows pointing diagonally down and left and up and right. The resize grip can be dragged in any direction.
CSR_DOWNRIGHT	Cursor has arrows pointing diagonally down and right and up and left. The resize grip can be dragged in any direction.
CSR_ALL	Cursor has four arrows pointing in all directions. The resize grip can be dragged in any direction.
CSR_BLOCK	Cursor appears as a block. The resize grip may not be dragged.
CSR_NONE	Cursor does not indicate the presence of a resize grip, and the resize grip is not drawn. The location cannot be dragged.
CSR_SELECT	Cursor has a different color resize grip for selection. Resize grip cannot be dragged. (Usually combined with CSR_NOLINETO.)
CSR_NOLINETO	Cannot draw a line between.
CSR_OBJFIXED	Use on first resize grip to indicate that the object cannot be moved by the mouse (in a fixed location).
CSR_DELETE	Cursor has a different color resize grip for selection. Resize grip cannot be dragged.
CSR_PLUS	Cursor has a different color resize grip for selection. Resize grip cannot be dragged.

CSR_NOLINETO and CSR_OBJFIXED are flags that should be combined with another flag using the "or" operator (e.g., `cursor[2] = CSR_ALL or CSR_NOLINETO`)

The typical handler for this function has three local arrays (one integer array for x, one integer array for y, and one array of GuiCursors for cursors) with the dimension size indicating the number of resize grips. The array elements should be assigned to the desired values. The arrays are then assigned to xCoords, yCoords, and cursors, respectively. Since the parameters are references to arrays, the values assigned in the handler are passed back to the system, which uses the data to create the resize grips for the object.

NOTE:
The initial size for the parameters is 0 (zero) elements.

Message:

MSG_TOOL_MOVEHANDLE

Description:

This message indicates that one of the object's resize grips has been dragged in Qlarity Foundry. The typical response to this activity is to resize the object, although certain handles may be associated with other object properties. The handler should set object properties relating to size (or whatever property is associated with the dragged grip), call API functions to alter attributes in the system software (usually done in property validation functions), and call Tool_Persist() for any property that has been modified.

Handler Format:

```
func <name> (<handle> as reference to ->
integer, <dx> as integer, <dy> as integer)
   handles MSG_TOOL_MOVEHANDLE
   [...]
endfunc
```

Parameters:

<handle>
An index of the handle defined in the arrays in the MSG_TOOL_GETHANDLES handler. If the handle type is CSR_SELECT, "<dx>" and "<dy>" are zero. If the handle number changes due to dragging, change the value of the handle to the appropriate number.

<dx>
The change in the resize grip's x-position caused by the drag.

<dy>
The change in the resize grip's y-position caused by the drag.

Message:

MSG_TOOL_DRAGCREATE

Description:

This message indicates that an object is being created by a click-and-drag operation in the Qlarity Foundry workspace. The format for the handler differs depending on the type of object being created. Non-drawable objects receive information about the parents to which they should attach. Area and container objects receive information about their position, size, and parents. The handler should set the object's initial properties, call API functions to set the object attributes in the system software (usually done in property validation functions), and finally call Tool_Persist() for any property that has been modified.

Handler Format:

(if handled by an area or container object)

```
func <name> (<parent> as objref, <x1> as ->
integer, <y1> as integer, <x2> as integer, ->
<y2> as integer)
   handles MSG_TOOL_DRAGCREATE
   [...]
endfunc
```

(if handled by a non-drawable object)

```
func <name> (<parent> as objref)
   handles MSG_TOOL_DRAGCREATE
   [...]
endfunc
```

Parameters:

<parent>
A reference to the parent to which the new object should attach itself.

<x1>
The x-position of the location where the drag was initiated.

<y1>
The y-position of the location where the drag was initiated.

<x2>
The x-position of the location where the drag was completed

<y2>
The y-position of the location where the drag was completed.

The coordinates (x1,y1) and (x2,y2) are not normalized, which means that (x2,y2) might be above and to the left of (x1,y1). The handler should normalize these coordinates, if necessary, before calculating the initial position and size of the object.

A typical handler for a rectangular object will normalize (x1,y1) and (x2,y2) if necessary, set the values for local properties that indicate position, size, and parent, call Tool_Persist() for all modified properties, and then call the object's MSG_INIT handler to handle the object initialization.

For more information on these messages and their handlers, refer to Chapter 10 in the *Qlarity Foundry User's Manual*.

Message:

MSG_TOOL_DELETEOBJ

Description:

This message is sent to an object in the GUI development environment just before the object is deleted. This allows the object to notify other objects that it is being removed from the workspace so other objects can respond appropriately.

Handler Format:

```
func <name> ()
   handles MSG_TOOL_DELETEOBJ
   [...]
endfunc
```

3.8 Special Messages

This section describes messages that are unique or that do not fit into another message category.

Message:

MSG_ERROR

Description:

This message is issued by the system software if an exception remains unhandled after all possible enclosing "check error/on error" blocks have been checked. It indicates to the handler that at least one unhandled exception resides in the system exception stack. The MSG_ERROR message can only be handled by a global message handler. Since the handler takes no parameters, the exception information must be retrieved by calling the GetException() API function.

Refer to the *Qlarity Foundry User's Manual* for more information on the exception handling system.

Handler Format:

```
func <name> ()
   handles MSG_ERROR
   [...]
endfunc
```

Parameters:

MSG_ERROR handlers take no parameters and do not have a return value.

Message:

MSG_DRAW_DONE

Description:

This message is sent by the system after a draw message has completed. The parameters indicate the bounding rectangle which was redrawn. This message, like MSG_ERROR, is only sent to a global message handler.

Handler format:

```
func DrawDone(<left> as integer, <top> ->
as integer, <width> as integer, <height> ->
as integer)
   handles MSG_DRAW_DONE
   return
   [...]
endfunc
```

Parameters:

<left>
The x-location of the upper left corner pixel of the invalid region.

<top>
y-location of the upper left corner pixel of the invalid region.

<width>
Width of the invalid region in pixels.

<height>
Height of the invalid region in pixels.

CHAPTER 4

QLARITY API FUNCTION REFERENCE

The Qlarity API (Application Programming Interface) is a library of functions that allow Qlarity applications to interact with the Qlarity-based hardware and perform common tasks that would be tedious or difficult to program in Qlarity. This chapter describes the available API functions, their parameters and return values, and the operations they perform.

4.1 Communications Interface

The communications API functions use an enumerated type called “comm” to specify the communications resource or interface. Variables of type “comm” may be assigned the values “COM1” (for the primary serial interface), “COM2” (for the secondary serial interface) or a value may be obtained from a call to NetOpen(), which assigns access to the Ethernet interface.

4.1.1 Send

Syntax:

```
send(resource as comm, data[] as anytype)
```

Parameters:

{resource}
The communications resource to transmit the data.

{data}
A byte array that contains the data for transmission.

Description:

This function generates a MSG_COMM_TRANSMIT message. The data in “{data}” is passed to any handlers for the message. This function is useful when implementing a common transmission interface (the MSG_COMM_TRANSMIT handler) to perform protocolization, gather statistics, and so on.

4.1.2 Transmit

Syntax:

```
transmit(resource as comm, data[] as ->
reference? to anytype, block as boolean)
```

Parameters:

{resource}
The communications resource to transmit the data.

{data}
An array of any type that contains the data for transmission.

{block}
A boolean flag to select buffered or unbuffered transmission.

Description:

The transmit function sends the data in “{data}” to the communications interface specified by “{resource}.” If the “{block}” parameter is true, then the unit waits until the transmission has finished before returning from the API function. If “{block}” is false, then the system software buffers the data and returns immediately. Transmission takes place in the background. Using buffered output is generally a more efficient use of system resources and is recommended for most transmissions.

4.1.3 SetBreak

Syntax:

```
setbreak(resource as comm, state as boolean)
```

Parameters:

{resource}
The communications resource to change the state of.

{state}
A boolean flag used to indicate whether a break state should begin. A value of “true” indicates to place the resource in break, where as “false” indicates that the break condition should be terminated.

Description:

This function allows the user to set the break state of a communications port.

4.1.4 GetComMessageSource

Syntax:

```
getcommessagesource()returns comm
```

Description:

This function returns the "comm" identifier for the current communications message.

4.1.5 NetOpen

Syntax:

```
netopen(obj as objref, prot as ->
netprotocol, localport as unibyte, ->
foreignport as unibyte, ipaddr[] as ->
reference? to byte)
```

Parameters:

{obj}
A reference to the object that will receive the MSG_COMM_ACCEPT message (see below).

{prot}
The protocol to be used for the connection (see below).

{localport}
The local port number to be used for TCP or UDP transmissions.

{foreignport}
The port number on the remote machine to be used for TCP or UDP transmissions.

{ipaddr}
A 4-byte array containing the IP address of the remote machine.

Description:

The NetOpen() function is used to establish an Ethernet communications channel. NetProtocol is an enumerated type with the following legal values:

NET_UDP
Use the User Datagram Protocol (UDP/IP)

NET_TCP
Use the Transmission Control Protocol (TCP/IP)

NetOpen() issues the request to open a communications channel. When the channel is established, the communications resource (used to receive and transmit data) is passed by the system to the object referenced by "{obj}" by the system in a MSG_COMM_ACCEPT message. This communications resource is passed to the Send() and Transmit() API functions to indicate which network channel should be used for transmission. It is also used to register for MSG_COMM_ RECEIVE messages generated by the network channel.

If "{foreignport}" is set to 0, the Qlarity-based terminal acts as a server listening for TCP connection requests or UDP datagrams on the port specified by "{localport}." Connections or datagrams with any foreign port number are accepted on this local port. When a new connection or datagram (from a unique foreign port) is accepted, a network channel is allocated and passed to the application in a MSG_COMM_ ACCEPT message. If no TCP connection is established or UDP datagram received, the application continues to listen on the specified local port until the terminal is reset.

NOTE:
This is a deprecated method for starting network servers. Use the NetServerOpen() API function instead.

Up to 64 channels (32 TCP channels) may be open at any given time. Channels may be released by calling NetClose().

Multicast Information Only:

If you are opening a multicast channel (any class D IP address), the following information applies

Calling NetOpen and specifying any class D IP address for the "{ipaddr}" parameter will open a multicast channel.

You must specify NET_UDP as the protocol. TCP/IP does not support multicast communication.

The "{foreignport}" parameter is only used to specify which port outgoing transmissions will be directed to. Incoming transmission's foreign port are not compared against this parameter. In other word, it doesn't matter what local port a device uses when transmitting to the multicast group.

The "{localport}" parameter is normally used to filter incoming data. Generally, you will only receive data packets that are directed at the multicast group specified by "{ipaddr}" and directed to the port specified by "{localport}". If you specify zero (0) for the local port, then

ALL transmissions directed to the multicast group are received. You cannot transmit on a comm channel that was opened with a local port of zero.

Incoming data on a multicast channel will be received via MSG_COMM_RECEIVE_MULTICAST messages. You should call RegisterMsgHandler as normal in a MSG_COMM_ACCEPT message to register for the MSG_COMM_RECEIVE_MULTICAST message.

4.1.6 NetServerOpen

Syntax:

```
netserveropen(obj as objref, prot as ->
netprotocol, localport as unibyte) ->
returns servercomm
```

Parameters:

{obj}
A reference to the object that will receive MSG_COMM_ACCEPT messages for this server (see below).

{prot}
The protocol to be used for the connection (see below).

{localport}
The local port number where the server will listen for datagrams (UDP) or connection requests (TCP).

Description:

The NetServerOpen() function is used to start an Ethernet communications server listening on the port specified by localport. NetProtocol is an enumerated type with the following legal values:

NET_UDP
Use the User Datagram Protocol (UDP/IP)

NET_TCP
Use the Transmission Control Protocol (TCP/IP)

NetServerOpen() starts the server and returns a server communications resource of type "servercomm." Connections or datagrams with any foreign port number are accepted on the local server port. When a new connection or datagram (from a unique foreign port) is accepted, a network communications resource is allocated and passed to the object referenced by "{obj}" in a MSG_COMM_ ACCEPT message. This communications resource is used with the Send() and Transmit() API functions to indicate which network channel should be used for transmission. It is also used to register for MSG_COMM_RECEIVE messages generated by the network channel.

Connections or datagrams with any foreign port number are accepted on this local port. When a new connection or datagram (from a unique foreign port) is accepted, a network channel is allocated and passed to the application in a MSG_COMM_ ACCEPT message. The application continues to listen on the specified local port until the terminal is reset.

Up to 64 channels (32 TCP channels) may be open at any given time. Channels may be released by calling NetClose(). The server will continue to listen on the specified local port until it is closed by calling the NetServerClose() API function.

4.1.7 NetClose

Syntax:

```
netclose(channel as comm)
```

Parameters:

{channel}
The communications channel to be closed.

Description:

This function closes the channel associated with the specified comm resource, which must have been obtained from a previous call to NetOpen().

4.1.8 NetServerClose

Syntax:

```
netserverclose(channel as servercomm)
```

Parameters:

{channel}
The communications channel to be closed.

Description:

This function closes the channel associated with the specified servercomm resource, which must have been obtained from a previous call to NetServerOpen().

4.1.9 ChangePort

Syntax:

```
changeport(channel as comm, {newport} as ->
unibyte)
```

Parameters:

{channel}
The communications resource to modify.

{newport}
The new local port number for the communications resource.

Description:

This function modifies a UDP communications resource that was previously obtained from a call to NetOpen(). The local port number for the communications resource is changed to "{newport}." ChangePort() can only be called for UDP resources; other resource types are not affected by this function.

4.1.10 TransmitUrgent

Syntax:

```
transmiturgent(channel as comm, data[] as ->
reference? to anytype)
```

Parameters:

{channel}
The TCP communications resource to transmit the data.

{data}
An array of any type that contains the data for urgent transmission via TCP.

Description:

This function sends the data in "{data}" via TCP urgent data mode to the TCP communications interface specified by "{channel}." Transmission takes place in the background.

4.1.11 GetNetChannelInfo

Syntax:

```
getnetchannelinfo(channel as comm, prot ->
as reference to netprotocol, lport as ->
reference to unibyte, fport as reference ->
to unibyte, ipaddr[] as reference to byte)
```

Parameters:

{channel}
The communications resource whose information will be returned.

{prot}
A reference to type "NetProtocol" that will receive "{channel}'s" protocol.

{lport}
A unibyte variable that will receive "{channel}'s" local port number.

{fport}
A unibyte variable that will receive "{channel}'s" foreign port number.

{ipaddr}
A byte array that will receive "{channel}'s" foreign IP address.

Description:

This function retrieves information about the "{channel}" communications resource. The protocol, local port number, foreign port number, and foreign IP address are stored in the reference parameters.

4.1.12 SetSerialRecvSize

Syntax:

```
setserialrecvsize(res as comm, newsize as ->
integer)
```

Parameters:

{res}
A serial communications resource (COM1 or COM2).

{newsize}
The desired size for the receive buffer.

Description:

This function sets the size of the receive buffer for serial communications on the specified interface. The receive buffer is the maximum number of characters that can be received with a single MSG_COMM_RECEIVE message. A MSG_COMM_RECEIVE message is generated whenever (1) the receive buffer is full, or (2) the receive buffer is not empty and no characters have been received for the timeout period (see SetSerialTimeout()).

4.1.13 SetSerialTimeout

Syntax:

```
setserialtimeout(res as comm, newtimeout ->
as integer)
```

Parameters:

{res}
A serial communications resource (COM1 or COM2).

{newtimeout}
The desired number of 20 ms intervals that should elapse before a timeout occurs (1 = 20ms).

Description:

This function sets the timeout period for serial communications on the specified interface. The timeout period is the maximum period of time between received characters before a MSG_COMM_RECEIVE message is generated. A MSG_COMM_RECEIVE message is generated whenever (1) the receive buffer is full (see SetSerialRecvSize()), or (2) the receive buffer is not empty and no characters have been received for the timeout period. Setting the timeout period to 0 disables timeout, meaning that MSG_COMM_RECEIVE messages are only generated when the receive buffer is full.

4.1.14 SetCTS

Syntax:

```
setcts(resource as comm, outValue as ->
boolean)
```

Parameters:

{resource}
The communications resource in which to change the CTS line.

{outValue}
The value to be placed on the CTS line.

Description:

This function is used to set the CTS line on a serial port of the Qlarity-based terminal if the line is available for use. (RTS/CTS flow control is only supported for the EIA-232 interface.)

4.1.15 ReadRTS

Syntax:

```
readrts(resource as comm) returns boolean
```

Parameters:

{resource}
The communications resource from which to read.

Description:

This function is used to read the RTS line on a serial port of the Qlarity-based terminal if the line is available for use. (RTS/CTS flow control is only supported for the EIA-232 interface.)

4.1.16 SetDSR

Syntax:

```
setdsr(resource as comm, outValue as ->
boolean)
```

Parameters:

{resource}
The communications resource to change the DSR line of.

{outValue}
The value to be placed on the DSR line.

Description:

This function is used to set the DSR line on a serial port of the Qlarity-based terminal, if that line is available for use.

NOTE:
The DSR line is not available on the Qlarity-based terminal primary or auxiliary serial ports. It may be available on expansion serial ports.

4.1.17 ReadDTR

Syntax:

```
readdtr(resource as comm) returns boolean
```

Parameters:

{resource}
The communications resource from which to read.

Description:

This function is used to read the DTR line on a serial port of the Qlarity-based terminal, if that line is available for use.

NOTE:
The DTR line is not available on the Qlarity-based terminal primary or auxiliary serial ports. It may be available on expansion serial ports.

4.1.18 Read DCD

Syntax:

```
readdcd(resource as comm) returns boolean
```

Parameters:

{resource}
The communications resource from which to read.

Description:

This function is used to read the DCD (carrier detect) line on a serial port of the Qlarity-based terminal, if that line is available for use.

4.1.19 NetSendDatagram

Syntax:

```
netsenddatagram(localport as unibyte, ->
foreignport as unibyte, ipaddr[] as ->
reference? to byte, data[] as reference? ->
to byte)
```

Parameters:

{localport}
The local port number to be used for the UDP datagram.

{foreignport}
The port number on the remote machine to be used for the UDP datagram.

{ipaddr}
A 4-byte array containing the IP address of the remote machine.

{data}
An array of bytes that contains the data for transmission in the datagram.

Description:

The NetSendDatagram() function is used to send a single datagram on the network using the User Datagram Protocol (UDP/IP). This function allows a datagram to be transmitted without opening a network channel (i.e. with the NetOpen API function). Both "{foreignport}" and "{localport}" must be set to non-zero values. This function is for transmission only. No response to the datagram will be received unless the appropriate channel has been established.

4.2 Registering for Messages

4.2.1 RegisterMsgHandler

Syntax:

```
registermsghandler(obj as objref, msgnum ->
as message, msgparm as unibyte)
```

Parameters:

{obj}
The object registering for the message.

{msgnum}
The name of the message (e.g., MSG_TIMETICK).

{msgparm}
A parameter whose meaning depends on the message (see section 3.4).

Description:

This function registers an object to receive registered messages. After this call, the object passed in "{obj}" is eligible to receive "{msgnum}" messages if it has an enabled path to root.

4.2.2 UnregisterMsgHandler

Syntax:

```
unregistermsghandler(obj as objref, ->
msgnum as message, msgparm as unibyte)
```

Parameters:

{obj}
The object registering for the message.

{msgnum}
The name of the message (e.g. MSG_-TIMETICK).

{msgparm}
A parameter whose meaning depends on the message (see section 3.4).

Description:

This function unregisters the "{obj}" object so that it will no longer receive "{msgnum}" messages.

4.2.3 RegisterKey

Syntax:

```
registerkey(obj as objref, keycode as ->
unibyte)
```

Parameters:

{obj}
The object registering for the message.

{keycode}
The keycode that causes a message to be generated for the object.

Description:

This function registers an object to receive MSG_KEY_DOWN, MSG_KEY_PRESS, and MSG_KEY _RELEASE messages when the keyboard or keypad key associated with "{keycode}" is activated. A list of valid keycodes is included in Appendix A.

In addition, there are two special keycodes not directly associated with specific keys. The KEY_ANY keycode registers the object to receive key messages for all keys. The KEY_NONE keycode effectively unregisters the object for key messages.

A given object may only be registered for one keycode value. Each call to RegisterKey() preempts any keycode that was previously registered for the "{obj}" object.

4.3 Manipulating Objects

4.3.1 GetObjref

Syntax:

```
getobjref(name}[] as reference? to byte) ->
returns objref
```

Parameters:

{name}
A string (byte array) that contains the name of the desired object.

Description:

This function takes the name of an object and returns an objref to the object.

4.3.2 GetObjProp

Syntax:

```
getobjprop(obj as objref, name[] as ->
reference? to byte) returns string
```

Parameters:

{obj}
A reference to the object with the desired property.

{name}
A string containing the name of the property to retrieve.

Description:

This function returns the value for the desired property converted to a string. The returned value is converted to a string (identical to calling the API function Str() on the property).

4.3.3 SetObjProp

Syntax:

```
setobjprop(obj as objref, name[] as ->
reference? to byte, {value}[] as ->
reference? to byte)
```

Parameters

{obj}
A reference to the object with the desired property.

{name}
A string containing the name of the property to set.

{value}
The new value for the property converted to a string.

Description:

This functions converts the string in "{value}" to the correct data type for the specified property then assigns that value to the property (refer to the "Val" function for conversion rules; see section 4.10.2).

4.3.4 Enable

Syntax:

```
enable(obj as objref, flag as boolean)
```

Parameters:

{obj}
A reference to the object to be enabled/disabled.

{flag}
A boolean flag containing the desired status. A value of "true" enables the object, and a value of "false" disables it.

Description:

This function enables or disables an object according to the "{flag}" parameter. An object must be enabled to receive messages and appear on the display.

4.3.5 GetContainer

Syntax:

```
getcontainer(obj as objref) returns objref
```

Parameters:

{obj}
A reference to a child object.

Description:

This function returns a reference to the parent container of the "{obj}" object.

4.3.6 GetChildren

Syntax:

```
getchildren(contobj as objref) returns ->
objref[]
```

Parameters:

{contobj}
A reference to a container type object.

Description:

This function returns an array containing references to all of the object's direct children (all objects whose parent is "{contobj}"). The children are returned in Z-order.

4.3.7 GetEnableInfo

Syntax:

```
getenableinfo(obj as objref, eval as ->
enable_info) returns boolean
```

Parameters:

{obj}
The object for which the status information is to be returned.

{eval}
The type of information desired.

Description:

This function returns the status of the object "{obj}" based on the information desired (indicated by "{eval}"). Legal values for "{eval}" are:

GET_ENABLED
Request the enabled status of an object.

GET_ZENABLED
Request whether the object's parents are enabled to root (does not include the object itself).

4.3.8 GetPosInfo

Syntax:

```
getposinfo(obj as objref, pval as ->
position_info) returns integer
```

Parameters:

{obj}
The object for which the position information is to be returned.

{pval}
The type of information desired.

Description:

This function returns the requested position information for "{obj}" based on the information desired (indicated by "{pval}"). Legal values for "{pval}" are:

GET_X
The current x-position of the object (relative to its parent).

GET_Y
The current y-position of the object (relative to its parent).

GET_WIDTH
The current width of the object.

GET_HEIGHT
The current height of the object.

GET_ORIGIN_X
The current origin x value (for a container only).

GET_ORIGIN_Y
The current origin y value (for a container only).

GET_XGLOBAL
The current x-position of the object (top, left of display: 0,0).

GET_YGLOBAL
The current y-position of the object (top, left of display: 0,0)

4.3.9 SetOrigin

Syntax:

```
setorigin(cont as objref, originX as ->
integer, originY as integer)
```

Parameters:

{cont}
The container for which to set the origin.

{originX}
The x coordinate of the top left pixel in the container, relative to the parent container.

{originY}
The y coordinate of the top left pixel in the container, relative to the parent container.

Description:

This function sets the origin of the container so that the upper left pixel of the container has the coordinates "{originX},{originY}" (with a default value of 0,0). The "{originX}" and "{originY}" coordinates are relative to the container itself. This causes all objects drawn on the container to be shifted appropriately. This does not change the manner in which the container's background is drawn.

4.4 Manipulating Z-Order

These functions change the Z-order of objects in the object hierarchy. Because the message handling system is highly dependent on Z-order, all changes to Z-order requested by these functions are postponed until processing of the current message is complete. Z-order changes are then executed in the order in which they were requested.

4.4.1 Attach

Syntax:

```
attach(obj as objref, parent as objref)
```

Parameters:

{obj}
A reference to the object that will be attached.

{parent}
A reference to the new parent object.

Description:

This function attaches an object to a container. The object being attached is placed in front of any other objects already attached to the parent. If the object is already attached to the parent, it will be moved to the top of the Z-order.

4.4.2 SendtoFront

Syntax:

```
sendtofront(obj as objref)
```

Parameters:

{obj}
A reference to the object to be moved in Z-order.

Description:

This function moves the object to the front of the list of children of the object's parent. If the object is the only child of its parent, or is already at the front, this has no effect.

4.4.3 SendtoBack

Syntax:

```
sendtoback(obj as objref)
```

Parameters:

{obj}
A reference to the object to be moved in Z-order.

Description:

This function moves the object to the back of the list of children for the object's parent. If the object is the only child of its parent, or is already at the back, this function has no effect.

4.4.4 Raise

Syntax:

```
raise(obj as objref)
```

Parameters:

{obj}
A reference to the object to be moved in Z-order.

Description:

This function moves the object in front of the sibling object directly in front of it. If the object is the only child of its parent, or is already at the front, this function has no effect.

4.4.5 Lower

Syntax:

```
lower(obj as objref)
```

Parameters:

{obj}
A reference to the object to be moved in Z-order.

Description:

This function moves the object behind the sibling object directly behind it. If the object is the only child of its parent, or is already at the back, this function has no effect.

4.5 Redrawing Portions of the Display

4.5.1 Rerender

Syntax:

```
rerender(obj as objref)
```

Parameters:

{obj}
A reference to the object to be redrawn.

Description:

This function invalidates the display region for the area of the specified object causing a MSG_DRAW message to be generated.

4.5.2 Resize

Syntax:

```
resize(obj as objref, width as integer, ->
height as integer)
```

Parameters:

{obj}
A reference to the object to resize.

{width}
The new width of the object.

{height}
The new height of the object.

Description:

This function resizes an object to the specified height and width by altering the system software object attributes for the specified object. The display regions for both the old and new area of the object are invalidated, and a MSG_DRAW message is generated. If you specify -1 for either parameter, the dimension for that parameter remains unchanged.

4.5.3 Relocate

Syntax:

```
relocate(obj as objref, x as integer, y ->
as integer)
```

Parameters:

{obj}
A reference to the object to be relocated.

{x}
The new x-position (horizontal coordinate) in pixels for the object (relative to the calling object's parent).

{y}
The new y-position (vertical coordinate) in pixels for the object (relative to the calling object's parent).

Description:

This function relocates an object to the specified position by altering the system software object attributes for the specified object. The display regions for both the old and new positions of the object are invalidated, and a MSG_DRAW message is generated.

4.6 Painting to the Display

Functions that draw on the display must be placed inside a MSG_DRAW message handler. Attempting to call these functions outside a MSG_DRAW message handler will cause a runtime exception to occur.

Many of these functions take a parameter of type "color," which is an enumerated type representing the many colors that are available on the color display. See Appendix A for a complete list of available colors. Color values may also be obtained at runtime using the RGB() function (see section 4.6.5).

For grayscale displays, the luminances of the selected colors are automatically converted to grayscale. This allows an application to run on either the color or grayscale display with little modification. (Since the grayscale conversion compresses multiple colors into each grayscale value, the transparency settings may cause some differences in how the application is displayed.)

The default foreground color is RGB_WHITE, and the default background color is RGB_BLACK.

4.6.1 SetTransparent

Syntax:

```
settransparent(newcolor as byte)
```

Parameters:

{newcolor}
A byte specifying the transparent color.

Description:

This function sets the transparent color to "{newcolor}" for the current message handler. This color setting persists until the message handler returns. The transparent color is not drawn on the display, allowing objects behind any transparent regions to remain visible.

4.6.2 UseTransparent

Syntax:

```
usetransparent(flag as boolean)
```

Parameters:

{flag}
A boolean value specifying whether transparency is enabled (true) or disabled (false).

Description:

This function enables or disables transparency for the duration of the current message handler or until it is called again. As transparency is disabled by default, this function must be called at least once in each message handler that uses the transparency feature.

4.6.3 SetFgColor

Syntax:

```
setfgcolor(newcolor as byte)
```

Parameters:

{newcolor}
A byte specifying the foreground color.

Description:

This function sets the foreground color to "{newcolor}" for the current message handler or until this function is called again. The last color setting persists until the message handler returns. The foreground color is used by various drawing API functions.

4.6.4 SetBgColor

Syntax:

```
setbgcolor(newcolor as byte)
```

Parameters:

{newcolor}
A byte specifying the background color.

Description:

This function sets the background color to "{newcolor}" for the current message handler or until this function is called again. The last color setting persists until the message handler returns. The background color is used by various drawing API functions.

4.6.5 RGB

Syntax:

```
rgb(red as byte, green as byte, blue as ->
byte)returns color
```

Parameters:

{red}
An 8-bit integer indicating the amount of red in the desired color. 255 is maximum, 0 is minimum.

{green}
An 8-bit integer indicating the amount of green in the desired color. 255 is maximum, 0 is minimum.

{blue}
An 8-bit integer indicating the amount of blue in the desired color. 255 is maximum, 0 is minimum.

Description:

This function is used to obtain a color value at runtime from user-specified red, green, and blue values. The returned color value may be used anywhere a color value is required [e.g., SetFgColor()].

4.6.6 SetPixel

Syntax:

```
setpixel(x as integer, y as integer)
```

Parameters:

{x}
The x-position of the pixel to draw (relative to the calling object's parent).

{y}
The y-position of the pixel to draw (relative to the calling object's parent).

Description:

This function draws the pixel at the specified location using the foreground color.

4.6.7 DrawLine

Syntax:

```
drawline(x1 as integer, y1 as integer, x2 ->
as integer, y2 as integer)
```

Parameters:

{x1}
The x-position of the first endpoint of the line (relative to the calling object's parent).

{y1}
The y-position of the first endpoint of the line (relative to the calling object's parent).

{x2}
The x-position of the second endpoint of the line (relative to the calling object's parent).

{y2}
The y-position of the second endpoint of the line (relative to the calling object's parent).

Description:

This function draws a line, using the foreground color, between the two points defined by (x1, y1) and (x2, y2).

4.6.8 DrawBitmap

Syntax:

```
drawbitmap(x as integer, y as integer, ->
bmp as bitmap)
```

Parameters:

{x}
The desired x-position of the upper-left corner of the bitmap (relative to the calling object's parent).

{y}
The desired y-position of the upper-left corner of the bitmap (relative to the calling object's parent).

{bmp}
The bitmap resource to draw.

Description:

This function displays a bitmap at the coordinate given. The origin of the x and y values is at the top left corner of the parent container.

4.6.9 DrawBitmapRegion

Syntax:

```
drawbitmapregion(x as integer, y as ->
integer, xoffset as integer, yoffset as ->
integer, width as integer, height as ->
integer, bmp as bitmap)
```

Parameters:

{x}
The desired x-position of the upper-left corner of the bitmap (relative to the calling object's parent).

{y}
The desired y-position of the upper-left corner of the bitmap (relative to the calling object's parent).

{xoffset}
The x-position of the pixel in the bitmap to draw at "{x},{y}."

{yoffset}
The y-position of the pixel in the bitmap to draw at "{x},{y}."

{width}
Number of pixels per row to copy from the bitmap.

{height}
Number of pixels per column to copy from the bitmap.

{bmp}
The bitmap resource to draw.

Description:

This function prints a portion of a bitmap at the coordinate specified by "{x},{y}." The displayed portion of the bitmap is specified by "{xoffset}," "{yoffset}," "{width}," and "{height}." The location of the "{x}" and "{y}" values is relative to the top left corner of the parent container.

4.6.10 GetObjPixmap

Syntax:

```
getobjpixmap(width as reference to ->
integer, height as reference to integer)->
returns color[]
```

Parameters:

{width}
Variable in which the width of object "pixmap" is to be stored.

{height}
Variable in which the height of object "pixmap" is to be stored.

Description:

This function is used to retrieve the pixel map for an object. The pixel map indicates the individual colors drawn in each pixel location for the object (based on the size of the object given to the Resize() function). The returned array is a color array of size "{width} * {height}," indicating the colors drawn at each location in the object up to this point. Thus, the color drawn at pixel coordinate x,y in the object (with 0,0 being the top left pixel of the object) is the color indicated by the (({y} * {width}) + {x}) element in the returned array.

4.6.11 DrawPixmap

Syntax:

```
drawpixmap(x as integer, y as integer, ->
pixmap[] as reference? to byte, mapwidth ->
as integer, mapheight as integer)
```

Parameters:

{x}
The desired x-position of the upper-left corner of the pixel map (relative to the calling object's parent).

{y}
The desired y-position of the upper-left corner of the pixel map (relative to the calling object's parent).

{pixmap}
The pixel map array.

{mapwidth}
The width of the pixel map.

{mapheight}
The height of the pixel map.

Description:

This function draws a pixel map. The "{pixmap}" array must have a size of "{mapwidth} * {mapheight}." The entire pixel map is copied starting with the top left pixel of the pixel map copied to the location specified by "{x},{y}."

4.6.12 DrawPixmapRegion

Syntax:

```
drawpixmapregion(x as integer, y as ->
integer, xoffset as integer, yoffset as ->
integer, width as integer, height as ->
integer, pixmap[] as reference? to byte, ->
mapwidth as integer, mapheight as integer)
```

Parameters:

{x}
The desired x-position of the upper-left corner of the pixel map (relative to the calling object's parent).

{y}
The desired y-position of the upper-left corner of the pixel map (relative to the calling object's parent).

{xoffset}
The x-position of the pixel in the pixel map to draw at "{x},{y}."

{yoffset}
The y-position of the pixel in the pixel map to draw at "{x},{y}."

{width}
Number of pixels per row to copy from the pixel map.

{height}
Number of pixels per column to copy from the pixel map.

{pixmap}
The pixel map array.

{mapwidth}
The width of the pixel map.

{mapheight}
The height of the pixel map.

Description:

This function draws a section of a pixel map. The "{pixmap}" variable must have a size of "{mapwidth}∗{mapheight}." The coordinate "{xoffset},{yoffset}" indicates which pixel in the pixel map should map to the position "{x},{y}." Starting at that point, a rectangular section of the pixel map is copied so that the total number of rows copied is "{height}," and the total number of columns copied is "{width}."

4.6.13 GetBitmapSize

Syntax:

```
getbitmapsize(width as reference to ->
integer, height as reference to integer, ->
bmp as bitmap)
```

Parameters:

{width}
An integer to receive the width of the bitmap resource.

{height}
An integer to receive the height of the bitmap resource.

{bmp}
The bitmap resource.

Description:

This function determines the size of a bitmap resource. "{width}" and "{height}" receive values, in pixels, corresponding to the width and height of the "{bmp}" bitmap resource.

4.6.14 DrawBox

Syntax:

```
drawbox(left as integer, top as integer, ->
right as integer, bottom as integer)
```

Parameters:

{left}
The x-position for the left side of the box (relative to the calling object's parent).

{top}
The y-position for the top of the box (relative to the calling object's parent).

{right}
The x-position for the right side of the box (relative to the calling object's parent).

{bottom}
The y-position for the bottom of the box (relative to the calling object's parent).

Description:

This function draws a filled box at the position specified. The box has a one-pixel border drawn in the foreground color and is filled with the background color.

4.6.15 DrawPolygon

Syntax:

```
drawpolygon(xpoints[] as integer, ->
ypoints[] as integer, flags as poly_flags)
```

Parameters:

{xpoints}
An array of integers containing the x-position for each point of the polygon (relative to the calling object's parent).

{ypoints}
An array of integers containing the y-position for each point of the polygon (relative to the calling object's parent).

{flags}
A value of type "poly_flags" (see below) that determines how the polygon will be drawn.

Description:

This function draws a polygon by connecting line segments between each of the points defined in the "{xpoints}" and

"{ypoints}" arrays. The "{flags}" parameter has type "poly_flags," which is an enumerated type with the following legal values:

POLY_NORMAL
Draw the outline of the polygon in the foreground color. Do not fill the polygon.

POLY_FILL
Draw the outline of the polygon in the foreground color and fill it with the background color.

POLY_NOCONNECT
Do not connect the ends of the polygon (it is a line - the POLY_FILL flag is ignored).

The polygon may have intersecting regions, if desired.

4.6.16 DrawEllipse

Syntax:

```
drawellipse(xoffset as integer, yoffset ->
as integer, a as float, {xfocal} as ->
float, yfocal as float, theta as float, ->
gamma as float, flags as ellipse_flags)
```

Parameters:

{xoffset}
The x-position of the center of the ellipse (relative to the calling object's parent).

{yoffset}
The y-position of the center of the ellipse (relative to the calling object's parent).

{a}
The distance along the primary focal point vector from the center to the edge of the ellipse (relative to the calling object's parent).

{xfocal}
The horizontal distance from the center to the primary focal point of the ellipse (relative to the calling object's parent).

{yfocal}
The vertical distance from the center to the primary focal point of the ellipse (relative to the calling object's parent).

{theta}
The angle (in radians) from the primary focal point vector segment to the starting point of the arc. Positive angles increase counter-clockwise from the primary focal point vector segment.

{gamma}
The angle (in radians) from the primary focal point vector segment to the ending point of the arc. Positive angles increase counter-clockwise from the primary focal point vector segment.

{flags}
A value of type "ellipse_flags" (see below) that determines how the ellipse will be drawn.

Description:

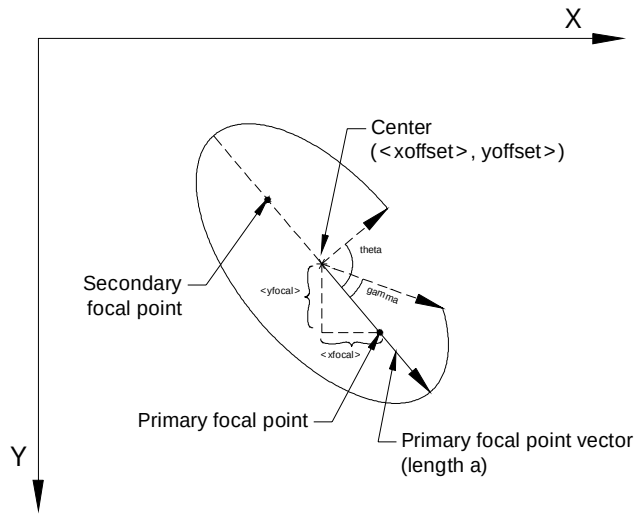
This complex but powerful function draws an open or closed elliptical arc centered at ({xoffset}, {yoffset}) relative to the calling object's parent.

The focal points of the ellipse are at the following points:

({xoffset} + {xfocal} and {yoffset} + {yfocal})
({xoffset} – {xfocal} and {yoffset} – {yfocal})

The mathematical definition of an ellipse is that the sum of the distances from the focal points to any point on the ellipse is a constant. In this case, the constant is defined as "2∗{a}." The elliptical arc is drawn counterclockwise from "{theta}" to "{gamma}." The simplest way to draw a closed ellipse is to draw a circle. Set "{theta}" to "0" and "{gamma}" to "2∗PI" (PI is a constant defined as 3.14159).

A circle is an ellipse with both focal points at the center. To draw a circular arc, set "{a}" to the desired radius, and set "{xfocal}" and "{yfocal}" to "0." To draw the complete circle, also set "{theta}" to "0," and set "{gamma}" to "2∗PI."



The "{flags}" parameter has type "ellipse_flags," which is an enumerated type with the following legal values:

ELLIPSE_NORMAL
Draw the specified arc in the foreground color.

ELLIPSE_FILL
Draw the specified arc in the foreground color, and fill the arc with the background color. If only this flag is specified, the boundaries of the filled region are the arc and a line segment connecting the endpoints of the arc. (Also known as ELLIPSE_CONNECT_ENDS.)

ELLIPSE_CONNECT_CENTER
Draw line segments connecting the endpoints of the arc to the center in the foreground color.

ELLIPSE_CONNECT_ENDS
Draw a line segment connecting the endpoints of the arc in the foreground color.

Multiple flags may be specified by combining them with a boolean AND operation.

4.6.17 GetEllipseSize

Syntax:

```
getellipsesize(maxleft as reference to ->
integer, maxright as reference to integer,
maxup as reference to integer, maxdown as ->
reference to integer, a as float, xfocal ->
as float, yfocal as float, theta as float,->
gamma as float, flags as ellipse_flags)
```

Parameters:

{maxleft}
An integer to receive the maximum distance in pixels from the center to the left edge of the ellipse.

{maxright}
An integer to receive the maximum distance from the center to the right edge of the ellipse.

{maxup}
An integer to receive the maximum distance from the center to the top edge of the ellipse.

{maxdown}
An integer to receive the maximum distance from the center to the bottom edge of the ellipse.

{a}
The distance along the primary focal point vector from the center to the edge of the ellipse.

{xfocal}
The horizontal distance from the center to the primary focal point of the ellipse.

{yfocal}
The vertical distance from the center to the primary focal point of the ellipse.

{theta}
The angle (in radians) from the primary focal point vector segment to the starting point of the arc. Positive angles increase counter-clockwise from the primary focal point vector segment.

{gamma}
The angle (in radians) from the primary focal point vector segment to the ending point of the arc. Positive angles increase counter-clockwise from the primary focal point vector segment.

{flags}
A value of type "ellipse_flags" (see section 4.6.16, "DrawEllipse") that determines how the ellipse is drawn.

Description:

This function determines the size of an elliptical arc relative to its center. The size is determined from the major axis, focal point, starting angle, and ending angle.

4.6.18 GetScreenPixmap

Syntax:

```
getscreenpixmap(x as integer, y as ->
integer, width as integer, height as ->
integer) returns color[]
```

Parameters:

{x}
The x-position of the first screen location to read (relative to root).

{y}
The y-position of the first screen location to read (relative to root).

{width}
The width indicates how many pixels per row should be copied from the screen.

{height}
The height indicates how many pixels per column should be copied from the screen.

Description:

This function is used to retrieve the pixel map for a section of the screen as it is currently drawn. This pixel map indicates the individual colors drawn up to this point in each pixel location of the screen. The returned array will be a color array of size "width * height", indicating the colors drawn so far. Please note that for grayscale units, the color returned may not be the exact color that was drawn to that location (but it will map to the same gray color as the original). The color drawn at pixel coordinate x,y in the object (with 0,0 being the top left pixel of the object) has the color indicated by the "((y * width) + x)"-th element in the returned array. Any requests for an area off of the screen will be transparent when drawn using drawpixmap.

4.6.19 UseDrawCache

Syntax:

```
usedrawcache(cachelevel as anytype)
```

Parameters:

{cachelevel}
Selects the method for draw caching (see below).

Description:

This function sets the method of draw caching which is used by the system software to improve graphics performance. The parameter "{cachelevel}" is of type "drawcache_level" and can take on any one of the following values:

CACHE_OFF
Disables draw caching.

CACHE_ALL
Maintains a cache for all objects in the application (best performance, highest memory requirement).

CACHE_ENABLED
Maintains a cache for all enabled objects (intermediate performance and memory requirement).

CACHE_EFFECTIVE_ENABLED
Only maintains a cache for objects that are both enabled and their ancestors are enabled (tracing back to root).

Thus, the level of draw caching can be tailored to the memory requirements of the application. DEPRECATED USAGE: This function will also accept a boolean value. A value of "true" for "{cachelevel}" selects CACHE_ALL, and a value of "false" selects CACHE_OFF.

4.6.20 IgnoreDrawCache

Syntax:

```
ignoredrawcache(obj as objref, ignore as ->
boolean)
```

Parameters:

{obj}
A reference to the object desiring to change status.

{flag}
A boolean flag containing the desired status. A value of "true" sets the object to ignore the drawcache setting, and a value of "false" sets the object to use the drawcache setting (the default setting).

Description:

This function is used to cause an object to ignore the drawcache settings so it will always get a draw when another object is drawn in its area.

4.6.21 DrawBorder

Syntax:

```
drawborder (x1 as integer, y1 as integer, ->
x2 as integer, y2 as integer, style as ->
integer, drawFlags as unibyte)
```

Parameters:

{x1}
The left edge of the border.

{y1}
The top edge of the border.

{x2}
Either the right edge of the border, or the width of the border rectangle, depending on whether BDR_WIDTHHEIGHT was specified for the drawFlags parameter.

{y2}
Either the bottom edge of the border, or the height of the border rectangle, depending on whether BDR_WIDTHHEIGHT was specified for the drawFlags parameter.

{style}
A special integer value that specifies how the border is to be drawn. Rather than specify your own value for this parameter, you should use either a value created by the PC development tool's border editor and stored in the _stylemap array, or you should use a value returned from the _CreateBorderStyle function.

{drawFlags}
Flags which modify how the border is drawn. See the description below.

Description:

This API allows objects to draw complex borders with ease. To use this function, specify the boundary rectangle for your border as well as a border style. In general, you should use either _CreateBorderStyle() to create a border style, or you should pass in a value that was stored in the global array _styleMap (this array is only available for workspaces created using the PC development tool). This style defines such things as whether the border is flat, or 3d, the border width, the corner radius, etc.

The drawFlags parameter is one or more of the following values combined using the OR operator:

BDR_FILL
Fill the interior of the border rectange with the current background color

BDR_WIDTHHEIGHT
If specified, the x2 and y2 parameters represent the width and height of the border rectangle. Otherwise, x2 and y2 are the right and bottom edges of the border rectangle.

BDR_CLEARCURVES
For rounded borders, this flag removes any drawing that may have occurred in the rectangle specified by x1, y1, x2 and y2, but outside the corner radius (i.e. it erases any drawing in the corners of the border when using a curved border). This is intended for objects drawing their outside border where previous drawing code may have drawn outside the border.

BDR_INVERSE
Draw the border in inverse. This causes raised borders to appear sunken, and sunken borders to appear raised. Flat borders draw using their natural inverse color.

BDR_FGCURVES
This option is occasionally used for borders on the interior of an object. Specify this option with a rounded border to cause the area inside the border rectangle but outside of the curves to be filled in with the foreground color.

4.6.22 GetObjPixmapRegion

Syntax:

```
getobjpixmapregion (x as integer, y as ->
integer, width as integer, height as ->
integer)
```

Parameters:

{x}
The x-position of the first pixel to read (relative to the object).

{y}
The y-position of the first pixel to read (relative to the object).

{width}
The variable passed in as width should contain the desired width of the resultant pixmap.

{height}
The variable passed in as height should contain the desired height of the resultant pixmap.

Description:

This function is used to retrieve a region of the pixel map for an object. This pixel map indicates the individual colors drawn up to this point in each pixel location for the object (based on the size of the object given to the resize function). The returned array will be a color array of size "width * height", indicating the colors drawn so far. Thus, the color drawn at pixel coordinate x,y in the object (with 0,0 being the top left pixel of the region) has the color indicated by the "(y * width) + x)"-th element in the returned array.

4.7 Rendering Text on the Display

Qlarity includes support for both raster (BDF) and True-Type (TTF) font resources. This section describes the API functions to render and paint text using these fonts.

4.7.1 GetBdfTextSize

Syntax:

```
getbdftextsize(width as reference to ->
integer, height as reference to integer, ->
xoffset as reference to integer, yoffset ->
as reference to integer, font as bdffont, ->
{data}[] as reference? to anytype, flags ->
as font_flags)
```

Parameters:

{width}
An integer to receive the width of the text box.

{height}
An integer to receive the height of the text box.

{xoffset}
An integer to receive the horizontal offset from the upper left corner of the rendered region to the origin of the first character.

{yoffset}
An integer to receive the vertical offset from the upper left corner of the rendered region to the origin of the first character.

{font}
The bdffont resource.

{data}
The data to be rendered as text. The type can be a unibyte or byte array (unistring or string).

{flags}
A value of type "font_flags" (see below) that determines how the text will be rendered.

Description:

This function finds the size of a given text string and returns it in the variables passed as parameters. The "{width}" and "{height}" variables receive the width and height in pixels of the rendered region. The "{xoffset}" and "{yoffset}" variables receive the offset from the upper left corner of the region to the origin of the first rendered character.

The parameter "{flags}" is of type "font_flags," which is a type that can have any combination of following legal values (other font_flags settings are ignored):

FONT_NORMAL
Render the text horizontally with the text in the foreground color and remainder of the region in the background color.

FONT_VERTICAL
Render the text vertically with the text in the foreground color and the remainder of the region in the background color. (Most fonts that use Latin characters do not support this option.)

FONT_INVERSE
Render the text horizontally with the text in the background color and the remainder of the region in the foreground color.

FONT_HBASELINE
Instead of calculating the standard width (text fits exactly in this width), calculate the baseline width (multiple lines will always have the same xoffset and thus will always line up properly).

FONT_VBASELINE
Instead of calculating the standard height (text fits exactly in this height), calculate the baseline height (multiple lines will always have the same yoffset and thus will always line up properly).

Multiple flags may be specified by combining them with a boolean AND operation.

Typically, this function is called as a precursor to calling the DrawBDFText() API function. The "{width}," "{height}," "{xoffset}," and "{yoffset}" values obtained from this function call are used as parameters in the call to DrawBdfText().

4.7.2 GetBDFTextFit

Syntax:

```
getbdftextfit(multiflags[] as reference ->
to multiline_flags, xpos[] as reference ->
to integer, ypos[] as reference to ->
integer, widths[] as reference to ->
integer, heights[] as reference to ->
integer, xoffsets[] as reference to ->
integer, yoffsets[] as reference to ->
integer, indices[] as reference to ->
integer, lengths[] as reference to ->
integer, font as bdffont, data[] as ->
reference? to sametype!, wordbrks[] as ->
reference? to sametype!, linebrks[] as ->
reference? to sametype!, flags as ->
font_flags)
```

Parameters:

{multiflags}
An array of type "multiline_flags" returned by the function giving information on how each line was fitted (see below).

{xpos}
An array of integers returned by the function that tells the relative horizontal offset for where each line of text should be drawn.

{ypos}
An array of integers returned by the function that tells the relative vertical offset for where each line of text should be drawn.

{widths}
An integer array used as both an input and an output. For input, the 0-th element is used to indicate the total width (in pixels) of the box that the text needs to fit into (unless FONT_HFIT is specified, in which case it is ignored). On return, the width for each line of text will be in the individual elements of the array, with the 0-th element always containing the final full width of all the text lines (if FONT_HFIT is specified then this is calculated, otherwise it is the same value as was passed in).

{heights}
An integer array used as both an input and an output. For input, the 0-th element is used to indicate the total height (in pixels) of the box that the text needs to fit into (unless FONT_VFIT is specified, in which case it is ignored). On return, the height for each line of text will be in the individual elements of the array, with the 0-th element always containing the final full height of all the text lines (if FONT_VFIT is specified then this is calculated, otherwise it is the same value as was passed in).

{xoffsets}
An integer array used as both an input and an output. For input, if FONT_HLEFT, FONT_HRIGHT, or FONT_HCENTER is not specified (by default, this is the case), the 0-th element is used to determine how far from the left edge the origin of the first text character should be. On return, the elements indicate how far from the left edge each line's origin should be placed.

{yoffsets}
An integer array used as both an input and an output. For input, if FONT_VTOP, FONT_VBOTTOM, or FONT_VCENTER is not specified (by default, this is the case), the 0-th element is used to determine how far from the top edge the origin of the first text character should be. On return, the elements indicate how far from the top edge each line's origin should be placed.

{indices}
An integer array used as both an input and an output. For input, the 0-th element gives the first index into "{data}" indicating where to begin the text fitting. Characters before this index are not fitted. A value of 0 starts at the beginning. On return, the index to the first character for each line of text will be in the individual elements of the array, with the 0-th element containing the index where text fitting began.

{lengths}
An integer array used as both an input and an output. For input, The number of elements of "{data}" to fit, or -1 to indicate all elements from the starting index to the end of "{data}." On return, the number of elements that should be drawn for each line of text is indicated in the individual elements of the array, with the 0-th element containing the total length of data fitted.

{font}
The bdffont resource.

{data}
The data to be rendered as text. The type can be a unibyte or byte array (unistring or string).

{wordbrks}
An array of the same type as "{data}" that determines which characters to treat as word breaks (spaces, tabs, etc.).

{linebrks}
An array of the same type as "{data}" that determines which characters to treat as line breaks (newline, form feed, etc.).

{flags}
A value of type "font_flags" (see below) that determines how the text will be rendered.

Description:

This function tries to find the best text fit for "{data}." A good understanding of the GetBDFTextSize and DrawBDF-Text functions should prelude attempts to use this function.

The parameter "{flags}" is of type "font_flags," which is a type that can have any combination of following legal values (other font_flags settings are ignored):

FONT_NORMAL
Render the text horizontally with the text in the foreground color and remainder of the region in the background color.

FONT_VERTICAL
Render the text vertically with the text in the foreground color and the remainder of the region in the background color. (Most fonts that use Latin characters do not support this option.)

FONT_INVERSE
Render the text horizontally with the text in the background color and the remainder of the region in the foreground color.

FONT_DRAWLINEBREAKS
Consider all line break characters as actually drawn when calculating fit (otherwise line break characters are NOT drawn).

FONT_DRAWWORDBREAKS
Consider all word break characters as actually drawn when calculating fit (otherwise word break characters are NOT drawn).

FONT_NOSOFTBREAKS
Disallow breaking a line anywhere that is not a line break or a word break.

FONT_HBASELINE
Instead of calculating the standard width (text fits exactly in this width), calculate the baseline width (multiple lines will always have the same xoffset and thus will always line up properly).

FONT_HFIT
Don't use the passed in width, but calculate the actual width needed to fit all of the given text.

FONT_HABS
Use the 0-th element of the xoffsets array to determine how far from the left edge the origin of the text should be placed.

NOTE:
Specifying FONT_HLEFT, FONT_HRIGHT, or FONT_HCENTER overrides FONT_HABS.

FONT_HLEFT
Fit the text such that it is all left justified.

FONT_HCENTER
Fit the text such that it is all center justified (horizontally).

FONT_HRIGHT
Fit the text such that it is all right justified.

FONT_VBASELINE
Instead of calculating the standard height (text fits exactly in this height), calculate the baseline height (multiple lines will always have the same yoffset and thus will always line up properly).

FONT_VFIT
Don't use the passed in height, but calculate the actual height needed to fit all of the given text.

FONT_VABS
Use the 0-th element of the yoffsets array to determine how far from the top edge the origin of the text should be placed.

NOTE:
Specifying FONT_VTOP, FONT_VBOTTOM, or FONT_VCENTER overrides FONT_VABS.

FONT_VTOP
Fit the text such that it is all top justified.

FONT_VCENTER
Fit the text such that it is all center justified (vertically).

FONT_VBOTTOM
Fit the text such that it is all bottom justified.

Multiple flags may be specified by combining them with a boolean AND operation.

Typically, this function is called as a precursor to calling the DrawBDFTextFit() API function. All values obtained from this function call are used as parameters in the call to DrawBDFTextFit().

On return, the array "{multiflags}" will contain information about how each line was rendered. Any combination of the following flags can be expected:

MULTILINE_SOFTBREAK
Use a boolean AND with this flag and the returned "{multiflags}" element to determine if the line was terminated by using a soft break.

MULTILINE_WORDBREAK
Use a boolean AND with this flag and the returned "{multiflags}" element to determine if the line was terminated at a word break element.

MULTILINE_LINEBREAK
Use a boolean AND with this flag and the returned "{multiflags}" element to determine if the line was terminated at a line break element.

MULTILINE_PARTIAL_WIDTH
Use a boolean AND with this flag and the returned "{multiflags}" element to determine if the line only fits partially inside the given width.

MULTILINE_NONE_WIDTH
Use a boolean AND with this flag and the returned "{multiflags}" element to determine if the line did not fit at all inside the given width.

MULTILINE_PARTIAL_HEIGHT
Use a boolean AND with this flag and the returned "{multiflags}" element to determine if the line only fits partially inside the given height.

MULTILINE_NONE_HEIGHT
Use a boolean AND with this flag and the returned "{multiflags}" element to determine if the line did not fit at all inside the given height.

4.7.3 GetBdfFontMetrics

Syntax:

```
getbdffontmetrics(maxleft as reference to ->
integer, maxright as reference to ->
integer, maxup as reference to integer, ->
maxdown as reference to integer, ->
xnextline as reference to integer, ->
ynextline as reference to integer, font ->
as bdffont, flags as font_flags)
```

Parameters:

{maxleft}
An integer to receive the maximum distance in pixels from a character origin to the left edge of any character in the font.

{maxright}
An integer to receive the maximum distance from a character origin to the right edge of any character in the font.

{maxup}
An integer to receive the maximum distance from a character origin to the top edge of any character in the font.

{maxdown}
An integer to receive the maximum distance from a character origin to the bottom edge of any character in the font.

{xnextline}
An integer to receive the horizontal offset from the origin of the first character on a line to the origin of the first character on the next line.

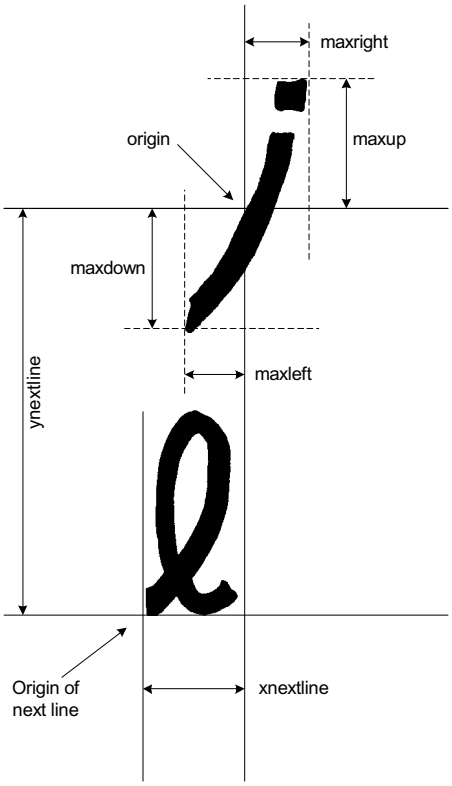
{ynextline}
An integer to receive the vertical offset from the origin of the first character on a line to the origin of the first character on the next line.

{font}
The bdffont resource.

{flags}
A value of type "font_flags" (see section 4.7.1) that determines how the text will be rendered

Description:

This function returns important metrics for the "{font}" resource which can be used to determine values for parameters in calls to DrawBdfText(). The diagram below illustrates the meaning of the various metrics. This function is particularly useful when rendering multiple lines of text.



4.7.4 DrawBdfText

Syntax:

```
drawbdftext(x as integer, y as integer, ->
width as integer, height as integer, ->
xoffset as integer, yoffset as integer, ->
font as bdffont, data[] as reference? to ->
anytype, {flags} as font_flags)
```

Parameters:

{x}
The x-position of the upper left corner of the rendered region (relative to the calling object's parent).

{y}
The y-position of the upper left corner of the rendered region (relative to the calling object's parent).

{width}
The width of the region to use for rendering.

{height}
The height of the region to use for rendering.

{xoffset}
The horizontal offset from the upper left corner of the rendered region to the origin of the first character.

{yoffset}
The vertical offset from the upper left corner of the rendered region to the origin of the first character.

{font}
The bdffont resource.

{data}
The data to be rendered as text. The type can be a unibyte or byte array (unistring or string).

{flags}
A value of type "font_flags" (see section 4.7.1) which determines how the text will be rendered.

Description:

This function renders the text contained in "{data}" and paints the rendered region to the display. The "{width}," "{height}," "{xoffset}," and "{yoffset}" values are typically obtained from a call to GetBdfTextSize() or inferred from metrics obtained from a call to GetBdfFontMetrics(). The rendered text is clipped if the area of the region defined by "{width}" and "{height}" is not sufficient, or if the values of "{xoffset}" and "{yoffset}" place some or all of the text outside of the rendering region.

4.7.5 DrawBDFTextFit

Syntax:

```
drawbdftextfit(multiflags[] as reference? ->
to multiline_flags, xpos[] as reference? ->
to integer, ypos[] as reference? to ->
integer, widths[] as reference? to ->
integer, heights[] as reference? to ->
integer, xoffsets[] as reference? to ->
integer, yoffsets[] as reference? to ->
integer, indices [] as reference? to ->
integer, lengths [] as reference? to ->
integer, font as bdffont, data[] as ->
reference? to anytype, flags as font_flags)
```

Parameters:

{multiflags}
An array of type "multiline_flags" indicating some information on how well the lines fit within the specified region.

{xpos}
An array of integers. The 0-th element indicates the x-position of the upper left corner of the rendered region (relative to the calling object's parent). Subsequent elements identify the relative horizontal offset from the initial x-position where each line of text should be drawn.

{ypos}
An array of integers. The 0-th element indicates the y-position of the upper left corner of the rendered region (relative to the calling object's parent). Subsequent elements identify the relative vertical offset from the initial y-position where each line of text should be drawn.

{widths}
An array of integers indicating the width for each line of text.

{heights}
An array of integers indicating the height for each line of text.

{xoffsets}
An array of integers indicating the horizontal offset from the upper left corner of the rendered region to the origin of the first character for each line.

{yoffsets}
An array of integers indicating the vertical offset from the upper left corner of the rendered region to the origin of the first character for each line.

{indices}
An array of integers the tells the index into "{data}" where each line of text begins.

{lengths}
An array of integers indicating the number of elements to draw in each line of "{data}."

{font}
The bdffont resource.

{data}
The data to be rendered as text. The type can be a unibyte or byte array (unistring or string).

{flags}
A value of type "font_flags" (see below) that determines how the text will be rendered.

Description:

This function renders the text contained in "{data}." The parameters are typically obtained by a call to GetBDFTextFit(). The rendered text is clipped as necessary.

A good understanding of the GetBDFTextSize and DrawBDFText functions should prelude attempts to use this function.

The parameter "{flags}" is of type "font_flags," which is a type that can have any combination of following legal values (other font_flags settings are ignored):

FONT_NORMAL
Render the text horizontally with the text in the foreground color and remainder of the region in the background color.

FONT_VERTICAL
Render the text vertically with the text in the foreground color and the remainder of the region in the background color. (Most fonts that use Latin characters do not support this option.)

FONT_INVERSE
Render the text horizontally with the text in the background color and the remainder of the region in the foreground color.

FONT_DRAW_HPARTIAL
Draw any horizontal lines that are marked as appearing partially within the fitted text region.

FONT_DRAW_VPARTIAL
Draw any vertical lines that are marked as appearing partially within the fitted text region.

Multiple flags may be specified by combining them with a boolean AND operation.

This function is included mostly to increase the efficiency of drawing multiline text. In Qlarity, it would look something like the following:

```
for tmp = 1 to len(lengths) - 1
drawbdftext(x[0] + x[tmp], y[0] + y[tmp], ->
widths[tmp], heights[tmp],xoffsets[tmp], ->

yoffsets[tmp],thefont,mid(text,->
indices[tmp], lengths[tmp]), flags)
next
```

4.7.6 GetTTFTextSize

Syntax:

```
getttftextsize(width as reference to ->
integer, height as reference to integer, ->
xoffset as reference to integer, yoffset ->
as reference to integer, {font} as ttfont,->
facenum as integer, {ptsize} as integer, ->
data[] as reference? to anytype, {flags} ->
as font_flags)
```

Parameters:

{width}
An integer to receive the width of the text box.

{height}
An integer to receive the height of the text box.

{xoffset}
An integer to receive the horizontal offset from the upper left corner of the rendered region to the origin of the first character.

{yoffset}
An integer to receive the vertical offset from the upper left corner of the rendered region to the origin of the first character.

{font}
The ttfont resource.

{facenum}
The number for the desired face in the ttfont resource (typically 0).

{ptsize}
The desired point size of the rendered text.

{data}
The data to be rendered as text (same types as BDF functions allowed).

{flags}
A value of type "font_flags" (see below) that determines how the text will be rendered.

Description:

This function finds the size of a given text string rendered in the given point size and typeface. The calculated values are returned in the variables passed as parameters. The "{width}" and "{height}" variables receive the width and height in pixels of the rendered region, respectively. The "{xoffset}" and "{yoffset}" variables receive the offset from the upper left corner of this region to the origin of the first rendered character. The "{flags}" is of type "font_flags," which is described in section 4.7.1.

Typically, this function is called as a precursor to calling DrawTTFText(). The "{width}," "{height}," "{xoffset}," and "{yoffset}" values obtained from this function call are used as parameters in the call to DrawTTFText().

4.7.7 GetTTFFontMetrics

Syntax:

```
getttfontmetrics(maxleft as reference to ->
integer, maxright as reference to integer,->
maxup as reference to integer, maxdown as ->
reference to integer, xnextline as ->
reference to integer, ynextline as ->
reference to integer, font} as ttfont, ->
facenum as integer, ptsize as integer, ->
flags as font_flags)
```

Parameters:

{maxleft}
An integer to receive the maximum distance in pixels from a character origin to the left edge of any character in the font.

{maxright}
An integer to receive the maximum distance from a character origin to the right edge of any character in the font.

{maxup}
An integer to receive the maximum distance from a character origin to the top edge of any character in the font.

{maxdown}
An integer to receive the maximum distance from a character origin to the bottom edge of any character in the font.

{xnextline}
An integer to receive the horizontal offset from the origin of the first character on a line to the origin of the first character on the next line.

{ynextline}
An integer to receive the vertical offset from the origin of the first character on a line to the origin of the first character on the next line.

{font}
The ttfont resource.

{facenum}
The number for the desired face in the ttfont resource (typically 0).

{ptsize}
The point size.

{flags}
A value of type "font_flags" (see section 4.7.1) that determines how the text is rendered

Description:

This function returns important metrics for the "{font}" resource that can be used to determine values for parameters in calls to DrawTTFText(). It is very similar to the analogous GetBdfFontMetrics() API function.

4.7.8 DrawTTFText

Syntax:

```
drawttftext(x as integer, y as integer, ->
width as integer, height as integer, ->
xoffset as integer, yoffset as integer, ->
font as ttfont, facenum as integer, ->
ptsize as integer, {data}[]as reference? ->
to anytype, {flags} as font_flags)
```

Parameters:

{x}
The x-position of the upper left corner of the rendered region (relative to the calling object's parent).

{y}
The y-position of the upper left corner of the rendered region (relative to the calling object's parent).

{width}
The width of the region to use for rendering.

{height}
The height of the region to use for rendering.

{xoffset}
The horizontal offset from the upper left corner of the rendered region to the origin of the first character.

{yoffset}
The vertical offset from the upper left corner of the rendered region to the origin of the first character.

{font}
The ttfont resource.

{facenum}
The number for the desired face in the ttfont resource (typically 0).

{ptsize}
The desired point size for the rendered text.

{data}
The data to be rendered as text. The type can be a unibyte or byte array (unistring or string).

{flags}
A value of type "font_flags" (see section 4.7.1) that determines how the text is rendered.

Description:

This function renders the text contained in "{data}" and paints the rendered region to the display. The "{width}," "{height}," "{xoffset}," and "{yoffset}" values are typically obtained from a call to GetTTFTextSize() or inferred from metrics obtained from a call to GetTTFontMetrics(). The rendered text is clipped if the area of the region defined by "{width}" and "{height}" is not sufficient, or if the values of "{xoffset}" and "{yoffset}" place some or all of the text outside of the rendering region.

4.7.9 SetTTFAngle

Syntax:

```
setttfangle(theta as float)
```

Parameters:

{theta}
The angle of the rendered text (in radians, relative to horizontal).

Description:

This function sets the angle at which text is rendered in using a ttfont resource, relative to the horizontal plane. The default angle is 0, indicating standard left-to-right horizontal rendering. This function should be called before GetTTFTextSize(). The last angle setting persists until the current message handler returns.

4.7.10 GetSysFontCharacters

Syntax:

```
getsysfontcharacters(font as sysfont, ->
facenum as integer, bRange as unibyte, ->
eRange as unibyte) returns unibyte[]
```

Parameters:

{font}
The sysfont resource.

{facenum}
The number for the desired face in a ttfont resource (typically 0).

{bRange}
The first character to be checked for in a font. Character codes greater than "{bRange}" but less than "{eRange}" are also checked.

{eRange}
The last character to be checked for in a font. Character codes less than "{eRange}" but greater than "{bRange}" are also checked.

Description:

GetSysFontCharacters returns a unibyte array indicating the existing character codes between bRange and eRange (inclusive) in a font.

4.7.11 GetSysTextSize

Syntax:

```
getsystextsize(width as reference to ->
integer, height as reference to integer,->
xoffset as reference to integer, yoffset ->
as reference to integer, font as sysfont,->
facenum as integer, ptsize as integer,->
data[] as reference? to anytype, flags as ->
font_flags)
```

Parameters:

{width}
An integer to receive the width of the text box.

{height}
An integer to receive the width of the text box.

{xoffset}
An integer to receive the horizontal offset from the upper left corner of the rendered region to the origin of the first character.

{yoffset}
An integer to receive the vertical offset from the upper left corner of the rendered region to the origin of the first character.

{font}
The sysfont resource. This may be either a BDF or a TrueType font.

{facenum}
The number for the desired face. This value is only used with TrueType fonts and is typically zero (0).

{ptsize}
The desired point size of the rendered text. This value is ignored unless font specifies a TrueType font.

{data}
The data to be rendered as text. The type can be a unibyte or byte array (unistring, string or charstr).

{flags}
A value of type "font_flags" (see GetBDFTextSize) that determines how the text will be rendered.

Description:

This function finds the size of a given text string rendered in the given point size and typeface. The calculated values are returned in the variables passed as parameters. The "{width}" and "{height}" variables receive the width and height in pixels of the rendered region, respectively. The "{xoffset}" and "{yoffset}" variables receive the offset from the upper left corner of this region to the origin of the first rendered character.

The "{flags}" is of type "font_flags"

Typically, this function is called as a precursor to calling DrawSysText(). The "{width}," "{height}," "{xoffset},"

and "{yoffset}" values obtained from this function call may be used as parameters in the call to DrawSysText().

4.7.12 GetSysTextFit

Syntax:

```
getsystextfit(multiflags[] as reference ->
to multiline_flags, xpos[] as reference ->
to integer, ypos[] as reference to ->
integer, widths[] as reference to ->
integer, heights[] as reference to ->
integer, xoffsets[] as reference to ->
integer, yoffsets[] as reference to ->
integer, indices[] as reference to ->
integer, lengths[] as reference to ->
integer, font as sysfont, facenum as ->
integer, ptsize as integer, data[] as ->
reference? to sametype!, wordbrks[] as ->
reference? to sametype!, linebrks[] as ->
reference? to sametype!, flags as ->
font_flags)
```

Parameters:

{multiflags}
An array of type "multiline_flags" returned by the function giving information on how each line was fitted.

{xpos}
An array of integers returned by the function that tells the relative horizontal offset for where each line of text should be drawn.

{ypos}
An array of integers returned by the function that tells the relative vertical offset for where each line of text should be drawn.

{widths}
An integer array used as both an input and an output. For input, the 0-th element is used to indicate the total width (in pixels) of the box that the text needs to fit into (if FONT_HFIT is specified, this value is only used for alignment purposes). On return, the width for each line of text will be in the individual elements of the array, with the 0-th element always containing the final full width of all the text lines (if FONT_HFIT is specified then this is calculated, otherwise it is the same value as was passed in).

{heights}
An integer array used as both an input and an output. For input, the 0-th element is used to indicate the total height (in pixels) of the box that the text needs to fit into (if FONT_VFIT is specified, this value is only used for alignment purposes). On return, the height for each line of text will be in the individual elements of the array, with the 0-th element always containing the final full height of all the text lines (if FONT_VFIT is specified then this is calculated, otherwise it is the same value as was passed in).

{xoffsets}
An integer array used as both an input and an output. For input, if FONT_HLEFT, FONT_HRIGHT, or FONT_HCENTER is not specified (by default, this is the case), the 0-th element is used to determine how far from the left edge the origin of the first text character should be. On return, the elements indicate how far from the left edge each line's origin should be placed.

{yoffsets}
An integer array used as both an input and an output. For input, if FONT_VTOP, FONT_VBOTTOM, or FONT_VCENTER is not specified (by default, this is the case), the 0-th element is used to determine how far from the top edge the origin of the first text character should be. On return, the elements indicate how far from the top edge each line's origin should be placed.

{indices}
An integer array used as both an input and an output. For input, the 0-th element gives the first index into "{data}" indicating where to begin the text fitting. Characters before this index are not fitted. A value of 0 starts at the beginning. On return, the index to the first character for each line of text will be in the individual elements of the array, with the 0-th element containing the index where text fitting began.

{lengths}
An integer array used as both an input and an output. For input, The number of elements of "{data}" to fit, or -1 to indicate all elements from the starting index to the end of "{data}." On return, the number of elements that should be drawn for each line of text is indicated in the individual elements of the array, with the 0-th element containing the total length of data fitted.

{font}
The sysfont resource. This may be either a BDF or a TrueType font.

{facenum}
The number for the desired face. This value is only used with TrueType fonts and is typically zero (0).

{ptsize}
The desired point size of the rendered text. This value is ignored unless font specifies a TrueType font.

{data}
The data to be rendered as text. The type can be a unibyte or byte array (unistring, string or charstr).

{wordbrks}
An array of the same type as "{data}" that determines which characters to treat as word breaks (spaces, tabs, etc...).

{linebrks}
An array of the same type as "{data}" that determines which characters to treat as line breaks (newline, form feed, etc...).

{flags}
A value of type "font_flags" that determines how the text will be rendered.

Description:

This function tries to find the best text fit for "{data}." A good understanding of the GetSysTextSize and DrawSysText functions should prelude attempts to use this function.

The parameter "{flags}" is of type font_flags, which is a type that can have any combination of legal values.

4.7.13 GetSysFontMetrics

Syntax:

```
getsysfontmetrics(maxleft as reference to ->
integer, maxright as reference to integer,->
maxup as reference to integer, maxdown as ->
reference to integer, xnextline as ->
reference to integer, ynextline as ->
reference to integer, font as sysfont, ->
facenum as integer, ptsize as integer, ->
flags as font_flags)
```

Parameters:

{maxleft}
An integer to receive the maximum distance in pixels from a character origin to the left edge of any character in the font.

{maxright}
An integer to receive the maximum distance from a character origin to the right edge of any character in the font.

{maxup}
An integer to receive the maximum distance from a character origin to the top edge of any character in the font.

{maxdown}
An integer to receive the maximum distance from a character origin to the bottom edge of any character in the font.

{xnextline}
An integer to receive the horizontal offset from the origin of the first character on a line to the origin of the first character on the next line.

{ynextline}
An integer to receive the vertical offset from the origin of the first character on a line to the origin of the first character on the next line.

{font}
The sysfont resource. This may be either a BDF or a TrueType font.

{facenum}
The number for the desired face. This value is only used with TrueType fonts and is typically zero (0).

{ptsize}
The point size. This value is ignored unless font specifies a TrueType font.

{flags}
A value of type "font_flags" (See section 4.7.1, "GetBdfTextSize" for a description)

Description:

GetSysFontMetrics function returns important metrics for the "{font}" resource which can be used to determine values for parameters in calls to DrawSysText ().

4.7.14 DrawSysText

Syntax:

```
drawsystext(x as integer, y as integer, ->
width as integer, height as integer, ->
xoffset as integer, yoffset as integer, ->
font as sysfont, facenum as integer, ->
ptsize as integer, data[] as reference? ->
to anytype, flags as font_flags)
```

Parameters:

{x}
The x-position of the upper left corner of the rendered region (relative to the calling object's parent).

{y}
The y-position of the upper left corner of the rendered region (relative to the calling object's parent).

{width}
The width of the region to use for rendering.

{height}
The height of the region to use for rendering.

{xoffset}
The horizontal offset from the upper left corner of the rendered region to the origin of the first character.

{yoffset}
The vertical offset from the upper left corner of the rendered region to the origin of the first character.

{font}
The sysfont resource. This may be either a BDF or a TrueType font.

{facenum}
The number for the desired face. This value is only used with TrueType fonts and is typically zero (0).

{ptsize}
The desired point size of the rendered text. This value is ignored unless font specifies a TrueType font.

{data}
The data to be rendered as text. The type can be a unibyte or byte array (unistring, string or charstr).

{flags}
A value of type "font_flags" (See section 4.7.1, "GetBdfTextSize" for a description)

Description:

DrawSysText renders the text contained in "{data}" and paints the rendered region to the display. The "{width}," "{height}," "{xoffset}," and "{yoffset}" values are often obtained from a call to GetSysTextSize() or inferred from metrics obtained from a call to GetSysFontMetrics(). The rendered text is clipped if the area of the region defined by "{width}" and "{height}" is not sufficient, or if the values of "{xoffset}" and "{yoffset}" place some or all of the text outside of the rendering region.

4.7.15 DrawSysTextFit

Syntax:

```
drawsystextfit(multiflags[] as reference? ->
to multiline_flags, xpos[] as reference? ->
to integer, ypos[] as reference? to ->
integer, widths[] as reference? to ->
integer, heights[] as reference? to ->
integer, xoffsets[] as reference? to ->
integer, yoffsets[] as reference? to ->
integer, indices[] as reference? to ->
integer, lengths[] as reference? to ->
integer, font as sysfont, facenum as ->
integer, ptsize as integer, data[] as ->
reference? to anytype, flags as font_flags)
```

Parameters:

{multiflags}
An array of type "multiline_flags" indicating some information on how well the lines fit within the specified region.

{xpos}
An array of integers. The 0-th element indicates the x-position of the upper left corner of the rendered region (relative to the calling object's parent). Subsequent elements identify the relative horizontal offset from the initial x-position where each line of text should be drawn.

{ypos}
An array of integers. The 0-th element indicates the y-position of the upper left corner of the rendered region (relative to the calling object's parent). Subsequent elements identify the relative vertical offset from the initial y-position where each line of text should be drawn.

{widths}
An array of integers indicating the width for each line of text.

{heights}
An array of integers indicating the height for each line of text.

{xoffsets}
An array of integers indicating the horizontal offset from the upper left corner of the rendered region to the origin of the first character for each line.

{yoffsets}
An array of integers indicating the vertical offset from the upper left corner of the rendered region to the origin of the first character for each line.

{indices}
An array of integers the tells the index into "{data}" where each line of text begins.

{lengths}
An array of integers indicating the number of elements to draw in each line of "{data}."

{font}
The sysfont resource. This may be either a BDF or a TrueType font.

{facenum}
The number for the desired face. This value is only used with TrueType fonts and is typically zero (0).

{ptsize}
The desired point size of the rendered text. This value is ignored unless font specifies a TrueType font.

{data}
The data to be rendered as text. The type can be a unibyte or byte array (unistring or string).

{flags}
A value of type "font_flags" (see below) that determines how the text will be rendered.

Description:

This function renders the text contained in "{data}." The parameters are typically obtained by a call to GetSysTextFit(). The rendered text is clipped as necessary. A good understanding of the GetSysTextSize and DrawSysText functions should prelude attempts to use this function.

The parameter "{flags}" is of type font_flags , which is a type that can have any combination of legal values.

4.8 Controlling the Speaker

4.8.1 PlayNote

Syntax:

```
playnote(note as byte, duration as integer)
```

Parameters:

{note}
The pitch from 0 to 86.

{duration}
The duration of the note in milliseconds.

Description:

The PlayNote function plays a note at the specified pitch for the specified duration. The duration is rounded to the nearest multiple of 20 milliseconds. The function returns immediately and the rates are queued.

Values for each note number are shown in the table below.

Number	Note	Frequency
0	rest note	0 Hz
1	A#/Bb	58.25 Hz
2	B	61.875 Hz
3	C	65.5 Hz
4	C#/Db	69.25 Hz
5	D	73.5 Hz
6	D#/Eb	77.75 Hz
7	E	82.5 Hz
8	F	87.25 Hz
9	F#/Gb	92.5 Hz
10	G	98 Hz
11	G#/Ab	103.75 Hz
12	A	110 Hz
13	A#/Bb	116.5 Hz
14	B	123.75 Hz
15	C	131 Hz
16	C#/Db	138.5 Hz
17	D	147 Hz
18	D#/Eb	155.5 Hz
19	E	165 Hz
20	F	174.5 Hz
21	F#/Gb	185 Hz
22	G	196 Hz

Number	Note	Frequency
23	G#/Ab	207.5 Hz
24	A	220 Hz
25	A#/Bb	233 Hz
26	B	247.5 Hz
27	C	262 Hz
28	C#/Db	277 Hz
29	D	294 Hz
30	D#/Eb	311 Hz
31	E	330 Hz
32	F	349 Hz
33	F#/Gb	370 Hz
34	G	392 Hz
35	G#/Ab	415 Hz
36	A	440 Hz
37	A#/Bb	466 Hz
38	B	495 Hz
39	Midl C	524 Hz
40	C#/Db	554 Hz
41	D	588 Hz
42	D#/Eb	622 Hz
43	E	660 Hz
44	F	698 Hz
45	F#/Gb	740 Hz
46	G	784 Hz
47	G#/Ab	830 Hz
48	A	880 Hz
49	A#/Bb	932 Hz
50	B	990 Hz
51	C	1048 Hz
52	C#/Db	1108 Hz
53	D	1176 Hz
54	D#/Eb	1244 Hz
55	E	1320 Hz

Number	Note	Frequency
56	F	1396 Hz
57	F#/Gb	1480 Hz
58	G	1568 Hz
59	G#/Ab	1660 Hz
60	A	1760 Hz
61	A#/Bb	1864 Hz
62	B	1980 Hz
63	C	2096 Hz
64	C#/Db	2216 Hz
65	D	2352 Hz
66	D#/Eb	2488 Hz
67	E	2640 Hz
68	F	2792 Hz
69	F#/Gb	2960 Hz
70	G	3136 Hz
71	G#/Ab	3320 Hz
72	A	3520 Hz
73	A#/Bb	3728 Hz
74	B	3960 Hz
75	C	4192 Hz
76	C#/Db	4432 Hz
77	D	4704 Hz
78	D#/Eb	4976 Hz
79	E	5280 Hz
80	F	5584 Hz
81	F#/Gb	5920 Hz
82	G	6272 Hz
83	G#/Ab	6640 Hz
84	A	7040 Hz
85	A#/Bb	7456 Hz
86	B	7920 Hz

4.8.2 PlayNoteNotify

Syntax:

```
playnotenotify (obj as objref, parm as ->
integer, note as byte, duratio} as integer)
```

Parameters:

{obj}
The object to notify when the sound has finished.

{parm}
Optional parameter to send with the message produced when the sound has been played.

{note}
The pitch from 0 to 86.

{duration}
The duration of the note in milliseconds.

Description:

This function plays a note at the specified pitch for the specified duration. The duration is rounded to the nearest multiple of 20 milliseconds. The function returns immediately and the rates are queued. After the note has played, the MSG_SOUND_DONE message is sent directly to the specified object, with parm as an optional parameter.

4.8.3 PlaySound

Syntax:

```
playsound (sound as _audio)
```

Parameters:

{sound}
The audio resource to play.

Description:

This function plays the audio resource identified by "{sound}." This function will only play the desired audio resource on units that have the audio decoder option installed.

4.8.4 PlaySoundNotify

Syntax:

```
playsoundnotify (obj as objref, parm as ->
integer, sound as _audio)
```

Parameters:

{obj}
The object to notify when the sound has finished.

{parm}
Optional parameter to send with the message produced when the sound has been played.

{sound}
The audio resource to play.

Description:

This function plays the audio resource identified by "{sound}." After the sound has played, the MSG_SOUND_DONE message is sent directly to the specified object, with parm as an optional parameter

4.8.5 StopSpkr

Syntax:

```
stopspkr()
```

Description:

This function immediately silences the speaker, terminating any note that is currently playing, and clears the speaker queue.

4.8.6 SetVolume

Syntax:

```
setvolume(direction as volume_adjust)
```

Parameters:

{direction}
A value of type "volume_adjust" (see below) that determines how to adjust the volume.

Description:

This function adjusts the volume of the unit at runtime. The adjustment is temporary and is forgotten when power is removed from the terminal. Permanent changes to the volume setting must be made with the Power On Setup utility or the SetSystemSetting() API function.

The SetVolume() function takes an argument of type "volume_adjust," which is a defined type with the following legal values:

VOLUME_LOUDER
Adjust the volume up.

VOLUME_QUIETER
Adjust the volume down.

4.9 Array and String Functions

4.9.1 Len

Syntax:

```
len(arr[] as reference? to anytype) ->
returns integer
```

Parameters:

{arr}
An array of any type.

Description:

This function returns the number of elements in the array.

4.9.2 Left

Syntax:

```
left(arr[] as reference? to sametype!, ->
len as integer) returns sametype!
```

Parameters:

{arr}
An array of any type.

{len}
The number of data elements to extract.

Description:

This function returns an array that contains the first "{len}" elements of "{arr}."

4.9.3 Right

Syntax:

```
right(arr[] as reference? to sametype!, ->
len as integer) returns sametype!
```

Parameters:

{arr}
An array of any type.

{len}
The number of data elements to extract.

Description:

This function returns an array that contains the last "{len}" elements of "{arr}."

4.9.4 Mid

Syntax:

```
mid(arr[] as reference? to sametype!, ->
index as integer, len as integer) returns ->
sametype!
```

Parameters:

{arr}
An array of any type.

{index}
The location of the first data element to extract (0-based).

{len}
The number of data elements to extract, or -1 to indicate all elements from the element at "{index}" to the end of "{arr}."

Description:

This function returns an array containing "{len}" elements from "{arr}," starting with the element located at "{index}" into "{arr}."

4.9.5 Trim

Syntax:

```
trim(arr[] as reference? to byte) returns ->
string
```

Parameters:

{arr}
Byte array.

Description:

This function removes any leading or trailing white space (spaces, tabs, carriage returns, or line feeds) from the byte array, returning the remaining characters.

4.9.6 Find

Syntax:

```
find(match[] as reference? to sametype!, ->
start as integer, length as integer, ->
pattern[] as reference? to sametype!) ->
returns integer
```

Parameters:

{match}
An array of any type to examine.

{start}
The location in "{match}" to begin the search; elements before the "{start}" element are not searched. 0 starts at the beginning.

{length}
The number of elements after "{start}" to search in "{match}," or -1 to indicate all elements from "{start}" to the end of "{match}."

{pattern}
The pattern of data elements to search for in "{match}."

Description:

This function searches for "{pattern}" in "{match}." If "{pattern}" is found, then the index of the first element of "{pattern}" in "{match}" is returned. If "{pattern}" is not found in "{match}," -1 is returned.

4.9.7 Concat

Syntax:

```
concat(strA[] as reference? to sametype!, ->
strB[] as reference? to sametype!) ->
returns sametype!
```

Parameters:

{strA}
An array of any type to be placed first in the returned array.

{strB}
An array of the same type as "{strA}" to be placed last in the returned array.

Description:

This function concatenates the arrays "{strA}" and "{strB}" and returns the resulting array.

NOTE:
It is generally easier to use the plus (+) operator to concatenate arrays.

4.9.8 Redim

Syntax:

```
redim(arr[] as reference to anytype, ->
newsize as integer)
```

Parameters:

{arr}
An array of any type.

{newsize}
The new size of the array "{arr}."

Description:

This function resizes an array of any type to the size specified by "{newsize}." No data is lost unless the array is decreased in size. If the array is increased in size, then the new elements of the array are initialized to the default value for the data type assigned to "{arr}."

4.9.9 ArrayOperation

Syntax:

```
arrayoperation (arr1[] as reference to ->
anytype, arr2[] as reference? to anytype, ->
op as ArrayOp)
```

Parameters:

{arr1}
An array variable that will receive the result of the array operation. Depending on the operation to be performed, this parameter may also be used as an input parameter to the function

{arr2}
An array used in the operation. How this parameter will be used is determined by the op parameter.

{op}
Determines which operation to perform on the input array(s).

Description:

This function performs one of several array operations, based on the value of the op parameter. The op parameter can take on the following values:

ARRAY_CONVERT
This operation converts arr2[] to the same data type as arr1[] and stores the result of that conversion in arr1[]. Most types of arrays can be converted. This operation is particularly useful when you are developing a workspace that uses both unicode strings and traditional byte strings and you need to convert between the two formats.

ARRAY_PALETTE
Both arr2[] and arr1[] must be of type byte or type color. Arr2[] should contain 256 elements. For each element in arr1[], the following assignment is performed: arr1[i] = arr2[arr1[i]]. This performs, in essence a software palette translation on a pixmap. This is used for the translucency and plasma effects you see in some objects.

ARRAY_REVERSE
Reverses the order of elements in arr2[] and stores the result in arr1[].

ARRAY_PFIELD
Reserved for future use. Do not use.

4.9.10 FreeArrayHandle

Syntax:

```
freearrayhandle(handle as arrayhandle)
```

Parameters:

{handle}
An array handle that was previously allocated with a call to AllocateArrayHandle.

Description:

This function releases an array handle that was allocated by calling AllocateArrayHandle and reclaims the memory used by that handle. After the handle has been freed, it is invalid and should not be used any more.

This function is an advanced API function and if misused may cause the terminal to exhaust its memory resources.

4.9.11 ReadArrayHandle

Syntax:

```
readarrayhandle (data[] as reference to ->
anytype, handle as arrayhandle)
```

Parameters:

{data}
An array which will receive a copy of the array associated with handle

{handle}
An array handle that was previously allocated with a call to CreateArrayHandle

Description:

This function reads an array associated with an array handle and stores the array in data. You must call this function only with array handles that have been allocated with the AllocateArrayHandle function and which have not been freed by calling FreeArrayHandle.

4.9.12 AllocateArrayHandle

Syntax:

```
allocatearrayhandle(data[] as reference? ->
to anytype) returns arrayhandle
```

Parameters:

{data}
An array to be associated with the returned handle.

Description:

This function makes a copy of the array that is passed in as the data parameter and returns a handle which can be used to access the new copy via the ReadArrayHandle function.

When the array is no longer needed, you MUST call FreeArrayHandle to release the array handle. This function is an advanced API function and if misused may cause the terminal to exhaust its memory resources.

4.9.13 ReverseFind

Syntax:

```
reversefind (match[] as reference? to ->
sametype!, start as integer, len as ->
integer, pattern[] as reference? to ->
sametype!) returns integer
```

Parameters:

{match}
An array of any type to examine.

{start}
The location in "{match}" to begin the search; characters

after the "{start}" character are not searched. -1 starts at the emd.

{len}
The number of elements before "{start}" to search in "{match}," or -1 to indicate all elements from "{start}" to the beginning of "{match}.".

{pattern}
The pattern of data elements to search for in "{match}."

Description:

This function searches for "{pattern}" in "{match}." If "{pattern}" is found, then the index of the first element of "{pattern}" in "{match}" is returned. If "{pattern}" is not found in "{match}," -1 is returned.

4.9.14 Replace

Syntax:

```
replace (match[] as reference to ->
sametype!, start as integer, len as ->
integer, pattern[] as reference? to ->
sametype!, newdata[] as reference? to ->
sametype!, count as integer) returns integer
```

Parameters:

{match}
An array of any type to examine.

{start}
The location in "{match}" to begin the search; characters before the "{start}" character are not searched. 0 starts at the beginning.

{len}
The number of elements after "{start}" to search in "{match}," or -1 to indicate all elements from "{start}" to the end of "{match}."

{pattern}
The pattern of data elements to search for in "{match}."

{newdata}
What to replace an occurrence of "{pattern}" with in "{match}."

{count}
The maximum number of times "{pattern}" should be replaced by "{newdata}" in "{match}," or -1 to replace all occurrences of "{pattern}" by "{match}."

Description:

This function searches for "{pattern}" in "{match}" and replaces up to "{count}" occurrences with "{newdata}." The returned value indicates how many replacements actually occurred.

4.10 Conversion Functions

4.10.1 Str

Syntax:

```
str(value as reference? to anytype) ->
returns string
```

Parameters:

{value}
The value to convert to a string.

Description:

This function converts "{value}" to a string representing the value. The system software uses heuristic rules to determine how to convert different types of values. Integers and floating point numbers are converted to text representations of the passed value (with sign, if negative, and decimal point, if floating point). Bytes and unibytes are treated as unsigned integers. Booleans convert to the strings "true" or "false." Objrefs convert to a string containing the referenced object's name. Arrays convert to a comma-separated list of converted values enclosed in brackets. User-defined data types such as enumerations are treated as integers.

4.10.2 Val

Syntax:

```
val(value as reference to anytype, text[] ->
as reference? to byte)
```

Parameters:

{value}
A variable (of any type) to receive the converted value.

{text}
An array of bytes (or a string) to be converted.

Description:

This function converts the text in "{text}" to a variable of "{value}'s" type and stores the result in "value." The conversions follow a similar heuristic to the Str() API function. Enumerations cannot be assigned a value with Val().

If "{value}" is an array, then "{text}" should be a list of values separated by commas and enclosed within brackets ([]). Alternatively, if "{value}" is an array of bytes, unibytes, or integers, "{text}" may contain text enclosed in double quotes. Each character in the quoted string is converted and assigned to a corresponding array member.

4.10.3 FromBytes

Syntax:

```
frombytes(var as reference to anytype, ->
toset[] as byte, bigendian as boolean)
```

Parameters:

{var}
A variable (see description below) to receive the binary data.

{toset}
A byte array containing the binary data.

{bigendian}
A flag indicating the endianness of the data in "{toset}." A value of "true" indicates big endian byte order, while a value of "false" indicates little endian byte order.

Description:

This function stores the binary data in "{toset}" in variable "{var}." This is useful for extracting variable data (such as integers or floating point numbers) from a byte stream. Only certain built-in data types are supported, and the "{toset}" array must be the correct size for "{var}'s" data type, as follows:

Data Type	Size
Integer	4 bytes
Float	4 or 8 bytes (in IEEE 754 format)[1]
Boolean	1 byte
Byte	1 byte
Unibyte	2 bytes
Array (of one of the above data types)	Size of data type (see above) * Number of elements

1. When an 8-byte array is passed for storage in a float variable, the data in "{toset}" is assumed to be a double precision floating point number, and it is automatically converted to single precision before storage in "{var}."

4.10.4 GetBytes

Syntax:

```
getbytes(tobreak as anytype, bigendian as ->
boolean) returns byte[]
```

Parameters:

{tobreak}
The variable (see description below) containing the desired binary data.

{bigendian}
The desired byte order for the data stored in "{tobreak}." A value of "true" stores the data in big endian byte order, while a value of "false" stores the data in little endian byte order.

Description:

This function returns a byte array containing the binary data from the "{tobreak}" variable. Only certain built in data types for "{tobreak}" are supported, and the length of the returned byte array is as follows:

Data Type	Size
Integer	4 bytes
Float	4 bytes (in IEEE 754 format)
Boolean	1 byte
Byte	1 byte
Unibyte	2 bytes
Array (of one of the above data types)	Size of data type (see above) * Number of elements

4.10.5 LowerCase

Syntax:

```
lowercase(str[] as byte) returns string
```

Parameters:

{str}
The string to convert to lower case.

Description:

This function converts all capital letters in the "{str}" string to lower case letters and returns the resulting string.

4.10.6 UpperCase

Syntax:

```
uppercase(str[] as byte) returns string
```

Parameters:

{str}
The string to convert to upper case.

Description:

This function converts all lower case letters in "{str}" to upper case letters and returns the resulting string.

4.11 Math Functions

4.11.1 Sin

Syntax:

```
sin(x as float) returns float
```

Parameters:

{x}
The floating point operand.

Description:

This function returns the Sine of "{x}."

4.11.2 Cos

Syntax:

```
cos(x as float) returns float
```

Parameters:

{x}
The floating point operand.

Description:

This function returns the Cosine of "{x}."

4.11.3 Tan

Syntax:

```
tan(x as float) returns float
```

Parameters:

{x}
The floating point operand.

Description:

This function returns the Tangent of "{x}."

4.11.4 Asin

Syntax:

```
asin(x as float) returns float
```

Parameters:

{x}
The floating point operand.

Description:

This function returns the Arc Sine of "{x}."

4.11.5 Acos

Syntax:

```
acos(x as float) returns float
```

Parameters:

{x}
The floating point operand.

Description:

This function returns the Arc Cosine of "{x}."

4.11.6 Atan

Syntax:

```
atan(x as float) returns float
```

Parameters:

{x}
The floating point operand.

Description:

This function returns the Arc Tangent of "{x}."

4.11.7 Power

Syntax:

```
power(x as float, exp as float) returns float
```

Parameters:

{x}
The base operand (floating point).

{exp}
The exponent operand (floating point).

Description:

This function returns "{x}" raised to the "{exp}" power.

4.11.8 Exp

Syntax:

```
exp({x} as float) returns float
```

Parameters:

{x}
The exponent operand (floating point).

Description:

This function returns e raised to the "{x}" power.

4.11.9 Ln

Syntax:

```
ln(x as float) returns float
```

Parameters:

{x}
The floating point operand.

Description:

This function returns the natural logarithm of "{x}."

4.11.10 Sqrt

Syntax:

```
sqrt(x as float) returns float
```

Parameters:

{x}
The floating point operand.

Description:

This function returns the square root of "{x}."

4.12 User Message Functions

4.12.1 UserBroadcastMsg

Syntax:

```
userbroadcastmsg(startobj as objref, ->
msgnum as message, parm as integer, donow ->
as boolean)
```

Parameters:

{startobj}
First object in Z-order to receive the message. Only this object and its children will receive the message. Specify a value of "default" to send the message to all objects.

{msgnum}
The user message to send.

{parm}
The integer parameter to associate with the sent message.

{donow}
A flag to select preemption of the current message while the user message is processed.

Description:

This function sends the "{msgnum}" user message as a broadcast message; it will go to all objects (enabled or disabled) starting with "{startobj}" in Z-order. Any object with a handler for "{msgnum}" is eligible to handle the message. Messages sent with this function cannot be terminated by a handler. The return value of the handler is ignored. The "{parm}" parameter may be set to any integer value and may be used for any purpose. If "{donow}" is set to "true," processing of the current message is suspended while the user message is processed by the system. When the user message processing is complete, this function returns and the current message continues processing. Any pending exceptions are thrown in the calling function. Setting "{donow}" to "false" causes the user message to be enqueued and handled as any other message.

4.12.2 UserSendMsg

Syntax:

```
usersendmsg(startobj as objref, msgnum as ->
message, parm as integer, donow as boolean)
```

Parameters:

{startobj}
First object in Z-order to receive the message. Only this object and its children will receive the message. Specify a value of "default" to send the message to all objects.

{msgnum}
The user message to send.

{parm}
The integer parameter to associate with the sent message.

{donow}
A flag to select preemption of the current message while the user message is processed

Description:

This function sends the "{msgnum}" user message as a normal message; it will go only to enabled objects starting with "{startobj}" in Z-order. Any object with a handler for "{msgnum}" is eligible to handle the message. Messages sent with this function can be terminated by a handler returning "true." If "false" is returned, the message continues through Z-order.

The "{parm}" parameter may be set to any integer value and may be used for any purpose. If "{donow}" is set to "true," processing of the current message is suspended while the user message is processed by the system. When the user message processing is complete, this function returns and the current message continues processing. Any pending exceptions are thrown in the calling function. Setting "{donow}" to "false" causes the user message to be enqueued and handled as any other message.

4.12.3 UserDirectMsg

Syntax:

```
userdirectmsg(startobj as objref, msgnum ->
as message, parm as integer, donow as ->
boolean) returns boolean
```

Parameters:

{startobj}
The object to which the message is being sent.

{msgnum}
The user message to send.

{parm}
The integer parameter to associate with the sent message.

{donow}
A flag to select preemption of the current message while the user message is processed

Description:

This function sends a user message directly to the "{startobj}" object. If "{startobj}" has a handler for "{msgnum}" and "{donow}" is "true," the handler is immediately called and executed. Otherwise, the "{msgnum}" message is posted in the messaging queue as a direct message to "{startobj}." The return value is the value returned by the called handler (if "{donow}" is "true") or "false" if "{startobj}" has no handler for "{msgnum}" (or if "{donow}" is "false"). The message is delivered whether "{startobj}" is enabled or disabled.

4.12.4 FakeKeyMsg

Syntax:

```
fakekeymsg(msgtype as fake_key, keycode ->
as UNIBYTE)
```

Parameters:

{msgtype}
The type of key message to post.

{keycode}
The keycode of the key to post.

Description:

This function allows the programmer to introduce a simulated key message into the messaging system (simulating a physical key press or release). The variable "{msgtype}" must be one of the following:

KEY_PUSH
KEY_REPEAT
KEY_RELEASE

4.12.5 FakeScreenMsg

Syntax:

```
fakescreenmsg(msgtype as fake_screen, x1 ->
as integer, y1 as integer, x2 as integer, ->
y2 as integer)
```

Parameters:

{msgtype}
Type of screen message to post.

{x1}
The x-location of the simulated screen event (for SCREEN_PUSH and SCREEN_RELEASE messages). For SCREEN_MOVE events, this is the x-location at the start of the move event.

{y1}
The y-location of the simulated screen event (for SCREEN_PUSH and SCREEN_RELEASE messages). For SCREEN_MOVE events, this is the y-location at the start of the move event.

{x2}
For SCREEN_MOVE events, this is the x-location at the end of the move event. This parameter is ignored for other types of screen events.

{y2}
For SCREEN_MOVE events, this is the y-location at the end of the move event. This parameter is ignored for other types of screen events.

Description:

This function allows the programmer to introduce a fake screen message into the messaging system. The variable "{msgtype}" must be one of the following:

SCREEN_PUSH
SCREEN_MOVE
SCREEN_RELEASE

4.13 User Input Capture

Some objects in Qlarity may want to capture or pre-process user input messages for themselves. Input messages are touch screen, keyboard, and keypad messages. To provide this, a capture list identifies objects wishing to receive input messages before they are passed through the normal messaging tree. Any object may appear in the list, but it will only appear once.

Input messages are handled as follows:

- All input messages go to the objects in the capture list first.
- Each object in the capture list is treated as a root object (receiving all touch screen messages regardless of location).
- Container objects in the capture list behave normally by passing the message to their children and then receiving a second pass. If an object in the capture list is not enabled back to root, that object does not capture an input message (although it still appears in the capture list).
- If an input message is killed while being processed by the capture list, it does not pass to other objects in the capture list or to root for normal message processing. If an input message is not killed, normal message processing occurs on the message (it is passed to root, which may cause objects in the capture list to receive the message twice).

The following API functions are used to examine and modify the capture list.

4.13.1 SetCapture

Syntax:

```
setcapture(obj as objref)
```

Parameters:

{obj}
The object to appear in the capture list.

Description:

This function places "{obj}" in the capture list (if it is not already). If it is already in the list, the object is moved to the front of the list.

4.13.2 GetCapture

Syntax:

```
getcapture(obj as objref) returns integer
```

Parameters:

{obj}
The object to look for in the capture list.

Description:

This function checks the capture list to determine if "{obj}" appears in the list. The return value indicates the location of the object in the list with 0 being first, 1 second, and so on. A –1 indicates that the object does not appear in the list.

4.13.3 RemoveCapture

Syntax:

```
removecapture(obj as objref)
```

Parameters:

{obj}
The object to remove from the capture list.

Description:

This function looks for "{obj}" in the capture list. If it is found, it is removed from the list.

4.14 Exception Functions

Refer to section 2.15, "Exception Handling" for general information and to the *Qlarity Foundry User's Manual* for more specific information on the Qlarity exception handling system. Exception names (types) and description strings are listed in Appendix B.

4.14.1 Throw

Syntax:

```
throw(loc[] as reference? to byte, msg[] ->
as reference? to byte)
```

Parameters:

{loc}
An array of bytes (or a string) indicating where the exception occurred (i.e., in which function).

{msg}
An array of bytes (or string) detailing specific information about what exception occurred.

Description:

This function throws a user exception to the exception handling system. "{loc}" should give information on where the exception was thrown, and "{msg}" should give details on what actual exception occurred. The error type is always EXCEPT_USER, and the error level is always EXLEV_USER.

4.14.2 GetException

Syntax:

```
getexception(msg[] as reference to byte, ->
errtype as reference to unibyte, ->
errlevel as reference to unibyte) returns ->
boolean
```

Parameters:

{msg}
An array of bytes (or a string) to receive specific information about what exception occurred.

{errtype}
A unibyte to receive the error type.

{errlevel}
A unibyte to receive the error level.

Description:

This function retrieves information about the last exception that occurred and is typically called in error handling code (such as inside an on error block). Once information about an exception is retrieved via a call to GetException(), the exception is removed from the exception queue; system handling for the exception is terminated. If the user exception handling code is insufficient, the exception can be returned to the exception queue by calling Rethrow().

Important:
If you do not call GetException() in an "on error" clause, the exception remains in the exception queue. A MSG_ERROR message is sent through the messaging system if there are still errors in the queue at the end of a message.

4.14.3 Rethrow

Syntax:

```
rethrow()
```

Description:

This function returns an exception to the exception queue after it has been removed by a previous call to GetException(). Rethrow() is typically called inside error handling code (such as inside an on error block) when an exception is retrieved that is not handled in that block of code.

4.15 User Non-Volatile Configuration Functions

The user configuration block is a nonvolatile section of system memory that can be used for any purpose. Typically, it is used to store configuration data that is specific to the user application. Data written to this memory is persistent across power cycles. Up to 8 kilobytes of data can be stored in the user configuration block.

4.15.1 ReadUserConfig

Syntax:

```
readuserconfig(len as integer) returns ->
byte[]
```

Parameters:

{len}
The number of bytes to read from the user configuration block.

Description:

This function reads "{len}" number of bytes from the user configuration block and returns them in a byte array. If the bytes were not previously written with a call to WriteUserConfig(), the bytes will contain garbage.

4.15.2 WriteUserConfig

Syntax:

```
writeuserconfig(cfg[] as reference? to byte)
```

Parameters:

{cfg}
An array of bytes to write to the user configuration block.

Description:

This function writes the bytes contained in "{cfg}" to the user configuration block. These bytes may subsequently be retrieved with a call to ReadUserConfig(). Since flash memory will fatigue and eventually fail after many write cycles (100,000+), care should be taken to avoid excessive calls to this function.

4.16 File System Functions

The Qlarity-based terminal implements a simple file system for storage of user data in flash, non-volatile memory. The file system supports a directory structure (of any depth) with path directory names delimited by either a forward slash (/) or a backslash (\). Directory and file names are limited to 128 characters in length, and the maximum path length is 256 characters.

The file system root directory is designated by a single forward slash (/) or backslash (\). Any path beginning with / or \ starts at the root directory. If a path does *not* start with / or \, it is considered to be relative to the current directory. The root directory is the default initial current directory.

This section describes functions for creating and using the file system. Since flash memory will fatigue and eventually fail after many write cycles (100,000+), care should be taken to avoid excessive calls to these functions.

4.16.1 GetAvailFilespace

Syntax:

```
getavailfilespace() returns integer
```

Description:

This function returns an estimate of the file space available for storing user data. As there is some storage overhead in the file system, the amount that is available for storing user data is slightly less than the returned value.

4.16.2 MakeDir

Syntax:

```
makedir(name[] as reference? to byte)
```

Parameters:

{name}
A string containing the desired name (and possibly path) for the new directory.

Description:

This function creates a new directory in the file system at the specified location. If the path included in "{name}" begins with / or \, it starts with the root directory; otherwise it is relative to the current directory.

4.16.3 ChangeCurDir

Syntax:

```
changecurdir(name[] as reference? to byte)
```

Parameters:

{name}
A string containing the desired path and directory name to make the current directory.

Description:

This function changes the current directory to the specified directory.

4.16.4 GetCurDir

Syntax:

```
getcurdir() returns string
```

Description:

This function returns a string containing the current directory.

4.16.5 GetDirEntry

Syntax:

```
getdirentry(index as integer) returns string
```

Parameters:

{index}
The index of the desired directory entry (see below).

Description:

This function lists the contents of the current directory. If "{index}" is 0, the first entry in the current directory is returned. Subsequent calls to GetDirEntry() with "{index}" set to -1 return subsequent entries in the directory. An empty string ("") is returned if no more entries are available. If the file system is accessed with other API function calls that alter the file system or current directory (such as ChangeCurDir()), GetDirEntry(0) should always be called before GetDirEntry(-1) or the returned results will be invalid.

Calling GetDirEntry(N) will return the Nth entry in the directory or an empty string ("") if the Nth entry does not exist.

4.16.6 EraseFile

Syntax:

```
erasefile(name[] as reference? to byte)
```

Parameters:

{name}
A string containing the name (possibly including path) of the file to remove.

Description:

This function removes the specified file from the file system. If the path included in "{name}" begins with / or \, it starts with the root directory; otherwise it is relative to the current directory. This function also erases directories.

4.16.7 GetFileInfo

Syntax:

```
getfileinfo(name[] as reference? to byte, ->
size as reference to integer, ->
usedfilespace as reference to integer, ->
flags as reference to fileinfo_flags)
```

Parameters:

{name}
A string containing the name (possibly including path) of the desired file.

{size}
A variable to receive the size of the file in bytes (directories return a size of 0 bytes).

{usedfilespace}
A variable to receive the number of bytes allocated to the file by the file system (which includes file system overhead).

{flags}
A variable of type "fileinfo_flags" to receive information about the file (see below).

Description:

This function returns information about a file in the file system. The actual size of the specified file is returned in the "{size}" variable. The number of bytes that the file consumes in the file space is returned in the "{usedfilespace}" variable.

The "{flags}" parameter is a variable of type "fileinfo_flags," which can take on the following legal values (logically ORed together):

FILEINFO_ISTEXT
Indicates that the file was created in text mode.

FILEINFO_ISBIN
Indicates that the file was created in binary mode.

FILEINFO_ISDIR
Indicates that the file is a directory.

FILEINFO_ISOPEN
Indicates that the file has been opened with a previous call to OpenFile().

To check the attributes of the file, logically AND "{flags}" with the appropriate value, and check to see if it is equal to that flag (i.e., "{flags}" AND FILEINFO_ISOPEN == FILEINFO_ISOPEN).

If the path included in "{name}" begins with / or \, it starts with the root directory; otherwise it is relative to the current directory.

4.16.8 OpenFile

Syntax:

```
openfile(name[] as reference? to byte, ->
flags as file_flags) returns filedesc
```

Parameters:

{name}
A string containing the name (possibly including path) of the desired file.

{flags}
A combination of the following values (logically ANDed together):

FILE_READ
Open file for reading (if FILE_WRITE is not specified, FILE_NO_CREATE is automatic and FILE_APPEND cannot be specified).

FILE_WRITE
Open file for writing.

FILE_TEXT
Open file in text mode (FILE_BINARY may not be specified).

FILE_BINARY
Open file in binary mode (FILE_TEXT may not be specified).

FILE_NO_CREATE
Do not create the file or destroy old file contents (it must exist).

FILE_APPEND
When opening a file for writing, do not destroy old file contents, and adjust file position to be at the end of the file.

Description:

This function opens a file for accessing the data and returns a file handle used to reference the file in other function calls. When opening, at least FILE_READ or FILE_ WRITE must be specified. If neither FILE_TEXT nor FILE_BINARY is specified, the access mode is determined as follows: if the file exists, it is opened in the same mode in which it was created; if the file does not exist, it is opened in text mode by default (as if FILE_TEXT were specified). Files opened in text mode do not support overwriting data in the file. There are a limited number of files that can be open at any one time (currently 7).

4.16.9 CloseFile

Syntax:

```
closefile(fnum as filedesc)
```

Parameters:

{fnum}
File handle returned from an OpenFile() call.

Description:

This function closes a file. The "{fnum}" file handle is invalid after a call to this function.

4.16.10 ReadFile

Syntax:

```
readfile(fnum as filedesc, var as ->
reference to anytype)
```

Parameters:

{fnum}
File handle of desired file.

{var}
Variable (of any type) to receive data read from the file.

Description:

This function reads data from the specified file and stores the data in variable "{var}." Data is read starting at the current position in the file. If the access mode of the file is binary, the number of bytes needed for "{var}" is read from the file and stored in "{var}." For arrays in binary mode, the size of the array determines the number of items read from the file. In text mode, one line of data (bytes up to but not including the newline character 0x10) is read from the file and is converted to "{var}'s" data type, as with a call to the Val() API function. The resulting value is stored in "{var}."

4.16.11 WriteFile

Syntax:

```
writefile(fnum as filedesc, data as ->
reference? to anytype)
```

Parameters:

{fnum}
File handle of the desired file.

{data}
A variable (of any type) containing data to write to the file.

Description:

This function writes data to the specified file based on the file's access method. If the file is in binary mode, the data is written in bytes exactly as it is in "{data}." In binary mode, bytes in the file are overwritten if the current file position is not at the end of the file. In text mode, the current file position must be at the end of the file. Also, the data is converted to a string (as with a call to the Str() API function), and the resulting string is written to the file followed by a newline character (0x10).

4.16.12 SetFilePos

Syntax:

```
setfilepos(fnum as filedesc, offset as ->
integer, absolute as boolean)
```

Parameters:

{fnum}
File handle of the desired file.

{offset}
An integer containing the offset to be added to the current position (or the beginning of the file).

{absolute}
A flag determining the basis for the new position calculation. A value of "true" adds "{offset}" from the beginning of the file. A value of "false" adds "{offset}" to the current file position.

Description:

This function sets the current read/write position for the specified file. To reset the position to the beginning of the file, use an offset of 0 and absolute as "true." (An offset of -1 is the end of the file.) For files in text mode, the position and offset refer to the line number in the file. For files in binary mode, these refer to the byte offset in the file.

4.16.13 GetFilePos

Syntax:

```
getfilepos(fnum as filedesc) returns integer
```

Parameters:

{fnum}
File handle of the desired file.

Description:

This function returns the current position in the file. For files in text mode, this refers to the line number of the current line in the file. For files in binary mode, the current position is a byte offset into the file.

4.16.14 EndOfFile

Syntax:

```
endoffile(fnum as filedesc) returns boolean
```

Parameters:

{fnum}
File handle of the desired file.

Description:

This function determines if the current position in the file is at the end of the file. A value of "true" is returned if this is the case, other "false" is returned.

4.16.15 EraseFileSpace

Syntax:

```
erasefilespace()
```

Description:

This function unconditionally erases the entire file space (all files and directories). USE WITH CAUTION.

4.16.16 RenameFile

Syntax:

```
renamefile(name[] as reference? to byte, ->
newname[] as reference? to byte)
```

Parameters:

{name}
A string containing the old name (possibly including path) of the file to rename.

{newname}
A string containing the new name (possibly including path) to be used to refer to the file.

Description:

This function will rename a file that is contained in the flash file system.

4.17 Qlarity Foundry Functions

These functions are only available in Qlarity Foundry, and calls to them should be enclosed in a "#if _TOOL/ #endif" directive to prevent their inclusion in the runtime application.

4.17.1 Tool_Persist

Syntax:

```
tool_persist(x as reference to anytype)
```

Parameters:

{x}
The name of a variable modified in a tool message handler.

Description:

This function informs Qlarity Foundry that an object property has been modified in a tool message handler. Qlarity Foundry must be informed of the change in order to update the application properly. Only call this if you want a value permanently saved. See section 3.7, "Tool Messages" for more details.

4.17.2 Tool_Trace

Syntax:

```
tool_trace(str as string)
```

Parameters:

{str}
A message to display in Qlarity Foundry.

Description:

This function causes Qlarity Foundry to display the message contained in "{str}" when the function is called. It is useful for debugging code. Note that in Layout View, Qlarity Foundry usually only executes handlers for the MSG_INIT and MSG_DRAW messages (as well as MSG_ERROR and user messages that originate in these messages), so any calls to Tool_Trace() must originate from handlers for these messages.

4.18 Miscellaneous Functions

4.18.1 SetGPIO

Syntax:

```
setgpio(pins as unibyte, action as ->
gpio_action)
```

Parameters:

{pins}
A value of type unibyte indicating which outputs to set. The "gpio_pin" type has been created to simplify use of this API. Legal values are PIN0, PIN1, PIN2, etc. Multiple values may be OR'ed together to select any combination of output pins.

{action}
A value of type "gpio_action" (see below) indicating how the outputs should be set.

Description:

This function sets the output state of one or more GPIO pins. The pins must be set to outputs using the SetGPIODirection API function before this function can be used. The function sets the outputs on the pins selected by "{pins}" according to the desired "{action}." Legal values for "{action}" include:

GPIO_SET
Set the outputs (logic HIGH) on the selected pins.

GPIO_CLEAR
Clear the outputs (logic LOW) on the selected pins.

GPIO_TOGGLE
Toggle the outputs on the selected pins.

4.18.2 ReadGPIO

Syntax:

```
readgpio(pins as unibyte) returns gpio-pin
```

Parameters:

{pins}
A value of type unibyte indicating which inputs to read. The "gpio-pin" type has been created to simplify use of this API. Legal values are PIN0, PIN1, PIN2, etc. Multiple values may be OR'ed together to select any combination of input pins.

Description:

This function reads the input state of one or more GPIO pins. (Pins may be set to inputs using the SetGPIODirection API function). The function reads the inputs on the pins selected by "{pins}" and returns a unibyte containing the read values. The return value must be AND'ed with the appropriate "gpio_pin" to determine the value on that pin. A non-zero result indicates that the input is set (logic HIGH), while a zero result indicates that the input is clear (logic LOW).

4.18.3 SetGPIODirection

Syntax:

```
setgpiodirection(pins as unibyte, input ->
as boolean)
```

Parameters:

{pins}
A value of type unibyte indicating which gpio pins to set to the desired direction. The "gpio-pin" type has been created to simplify use of this API. Legal values are PIN0, PIN1, PIN2, etc. Multiple values may be OR'ed together to select any combination of pins.

{input}
A boolean flag that should be set to TRUE if the desired direction is input, and FALSE if the desired direction is output.

Description:

This function sets the input/output direction of one or more GPIO pins. The function sets the pins selected by "{pins}" to inputs if "{input}" is TRUE, otherwise the pins are set to outputs.

4.18.4 GetVersion

Syntax:

```
getversion() returns float
```

Description:

This function returns the version of the system software (firmware) currently programmed into the terminal.

4.18.5 GetHardwareInfo

Syntax:

```
gethardwareinfo(req as hwinfo) returns ->
byte[]
```

Parameters:

{req}
A value of type "hwinfo" (see below) indicating the desired hardware information.

Description:

This function returns information about the terminal hardware. It returns a byte array containing the requested information. The returned information is determined by the "{req}" argument. This argument is of type "hwinfo," which is an enumerated type with the following legal values:

HW_ETHERNET
Request information on the Ethernet interface. The returned byte array contains two bytes. The first byte is set to 1 if the Ethernet interface is present, otherwise it is set to 0. The second byte is set to 1 if the Ethernet link is currently active, otherwise it is set to 0.

HW_TOUCH
Request information on the touch screen. The returned byte array contains one byte, which is set to 1 if a touch screen is present, otherwise it is cleared to 0.

HW_KEYPAD
Request information on the keypad. The function returns a byte array of variable length. The first byte is set to 1 if a keypad is present, otherwise it is cleared to 0 (and the length of the returned byte array is 1). If a keypad is present, the remaining bytes in the array contain all possible keycodes for the keypad. The keycodes must by logically OR'ed with 0x8000 to obtain the actual keycodes that will be received by the application via keypad messages.

HW_KEYBOARD
Request information on the keyboard. The returned byte array contains one byte, which is set to 1 if a keyboard is present, otherwise it is cleared to 0.

HW_TEMPCONTROL
Request information on the temperature compensation controller. The returned byte array contains one byte, which is set to 1 if a temperature compensation controller is present, otherwise it is cleared to 0.

HW_CLOCK
Request information on the real time clock. The returned byte array contains one byte, which is set to 1 if a battery-backed real time clock is present, otherwise it is cleared to 0.

HW_DISPLAY
Request information on the display. The returned byte array combines four bytes with the display pixel width stored in the first two bytes (MSB first), and the display pixel height stored in the last two bytes (MSB first). Following the four bytes are three bytes indicating more about the type of display in the following order: color, tft, and transflective. A value of 0 for any of those three fields indicates that the display does not have that property.

HW_DEFAULTAPP
Request information on whether a default application is available in the unit. The returned byte array contains one byte, which is set to 1 if a default application is present, otherwise it is cleared to 0.

HW_MACADDRESS
Request information on what the Ethernet address of the unit is. The returned byte array contains a six byte Ethernet address.

HW_AUDIOCODEC
Request information on the audio capabilities of the terminal. The returned byte array contains one byte, which is set to 1 if an audio decoder is present, otherwise it is cleared to 0.

HW_GPIO
Request information on the general purpose digital input/output (GPIO) capabilities of the terminal. The returned byte array contains one byte, which is set to 1 if GPIO is present, otherwise it is cleared to 0.

HW_CPU
Request information on the type of CPU present in the unit, and its execution frequency. The returned byte array contains the text name of the CPU, a '/' character, and some text indicating the instruction execution frequency in Hertz.

HW_COMLIST
Request information on which communications ports are present. The returned byte array will contain a 1 in every location that a com exists (COM1 would correlate with the 0-th element in the array). The size of the array will depend on how many serial ports are installed. Serial ports do not exist if there is no corresponding element in the array.

HW_MEMORY
Request statistics on unit memory. The first four elements of this byte array indicate the total amount of RAM that the unit has (MSB). The next four elements indicate the total amount of flash memory available (MSB). The last four elements indicate an estimate of the current amount of RAM the unit still has available (MSB).

4.18.6 SetContrast

Syntax:

```
setcontrast(direction as contrast_adjust)
```

Parameters:

{direction}
A value of type "contrast_adjust" (see below) that determines how to adjust the contrast.

Description:

This function adjusts the contrast of the display at runtime (i.e., not with Power On Setup). The adjustment is temporary and is forgotten when power is removed from the terminal. Permanent changes to the contrast setting must be made with the Power On Setup utility or the SetSystemSetting() API function.

The SetContrast() function takes an argument of type "contrast_adjust," which is a defined type with the following legal values:

CONTRAST_LIGHTER
Adjust the contrast one step lighter.

CONTRAST_DARKER
Adjust the contrast one step darker.

4.18.7 SetBacklight

Syntax:

```
setbacklight({command} as backlight_adjust)
```

Parameters:

{command}
A value of type "backlight_adjust" (see below) that determines how to adjust the backlight setting.

Description:

This function adjusts the backlight setting of the display at runtime (i.e., not with Power On Setup). The adjustment is temporary and is forgotten when power is removed from the terminal. Permanent changes to the backlight setting should be made with the Power On Setup facility or the SetSystemSetting() API function.

The SetBacklight() function takes an argument of type "backlight_adjust," which is a defined type with the following legal values:

BACKLIGHT_ON
Turn backlight on (set to previous "on" setting).

BACKLIGHT_OFF
Turn backlight off (power saving mode).

BACKLIGHT_LIGHTER
Adjust the backlight one step lighter.

BACKLIGHT_DARKER
Adjust the backlight one step darker.

4.18.8 EnableKeypadBacklight

Syntax:

```
enablekeypadbacklight(enable as boolean)
```

Parameters:

{enable}
A boolean value that sets the state of the keypad backlight. A setting of TRUE turns on the backlight, while a setting of FALSE turns it off.

Description:

The EnableKeypadBacklight function sets the current state of the keypad backlight (if one is available). The setting is temporary and is forgotten when power is removed from the terminal. Permanent changes to the keypad backlight setting should be made with the Power On Setup facility or the SetSystemSetting() API function.

4.18.9 SetLED

Syntax:

```
setled(cmd as ledcmd, lednum as integer)
```

Parameters:

{cmd}
A value of type "led_cmd" (see below) that determines how the LED setting is adjusted.

{lednum}
An integer indicating which LED is adjusted.

Description:

This function changes the state (on or off) of one of the keypad LEDs. The "{cmd}" parameter of type "led_cmd" is a defined type with the following legal values:

LED_ON
Turn on the specified LED.

LED_OFF
Turn off the specified LED.

LED_TOGGLE
Toggle the state of the specified LED.

4.18.10 GetTime

Syntax:

```
gettime(day as reference to integer, ->
month as reference to integer, dig2year ->
as reference to integer, dotw as reference ->
to weekday, hour as reference to integer, ->
minute as reference to integer, second as ->
reference to integer)
```

Parameters:

{day}
An integer to receive the current day of the month.

{month}
An integer to receive the current month.

{dig2year}
An integer to receive the last two digits of the current year.

{dotw}
A variable of type "weekday" (see below) to receive the current day of the week.

{hour}
An integer to receive the current hour (military time).

{minute}
An integer to receive the current minute.

{second}
An integer to receive the current second.

Description:

This function returns the current time as maintained by the real time clock hardware.

The variable "{dotw}" is of type "weekday," which is a defined type with the follow (self-explanatory) legal values:

SUNDAY
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY

4.18.11 SetTime

Syntax:

```
settime(day as integer, month as integer, ->
dig2year as integer, dotw as weekday, hour->
as integer, minute as integer, second as ->
integer)
```

Parameters:

{day}
An integer indicating the current day of the month.

{month}
An integer indicating the current month.

{dig2year}
An integer indicating the last two digits of the current year.

{dotw}
A variable of type "weekday" (see below) indicating the current day of the week.

{hour}
An integer indicating the current hour (military time).

{minute}
An integer indicating the current minute.

{second}
An integer indicating the current second.

Description:

This function sets the current time which is maintained by the real time clock hardware. The variable "{dotw}" is of type "weekday," which is a defined type with the following (self-explanatory) legal values:

SUNDAY
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY

4.18.12 GetTemperature

Syntax:

```
gettemperature() returns integer
```

Description:

This function returns the current temperature (degrees Celsius) inside the terminal case. The Qlarity-based hardware includes a temperature sensor that is used to implement temperature compensation of display contrast. The sensor is accurate to approximately ±2° C, and is calibrated by a setting in the Power On Setup utility.

4.18.13 TypeOf

Syntax:

```
typeof(unique as reference to integer, ->
obj as objref, {name}[] as reference? to ->
byte) returns typeval
```

Parameters:

{unique}
An integer to receive a type identifier if the property or variable to be typed is user-defined. Built-in types cause 0 to be stored in this parameter.

{objref}
Objref of the object (default for global) whose property type will be returned.

{name}
Name of the property/variable whose type will be returned.

Description:

This function determines the type of an object property or variable. The returned value is of type "TYPEVAL" with legal values as shown below.

Value	Type
INTEGER_TYPE	Integer
FLOAT_TYPE	Floating Point
BOOLEAN_TYPE	Boolean
BYTE_TYPE	Byte
UNIBYTE_TYPE	Unibyte
ARRAY_TYPE	Array
OBJREF_TYPE	Object Reference

If the object property or variable is a user-defined type, a unique, non-zero value corresponding to the type will be stored in "{unique}."

4.18.14 SetSystemSetting

Syntax:

```
setsystemsetting({cmd} as syscmd, ->
newvalue as anytype, action as ->
syscmd_action)
```

Parameters:

{cmd}
A value of type "syscmd" (see below) that determines what terminal control setting will be modified.

{newvalue}
The new value for the selected terminal control setting. This value expects an integer if not otherwise noted in the description of the "syscmd."

{action}
A value of type "syscmd_action" (see below) that determines the time at which the new settings take effect.

Description:

This function modifies most of the hardware terminal settings for the Qlarity-based terminal. These settings can also be saved in the non-volatile flash memory of the terminal, making the settings effective across power cycles. When writing changes that are to be saved to the flash, settings are first written to a pending save area. The pending changes are only saved to the flash when this function is called with the command SYS_SAVE.

The "{action}" argument is of type SYSCMD_ACTION and can take the following values:

SYSACT_DONOW
The new setting will take effect immediately but not persist after the terminal power is cycled. A value set using this action will not be saved when this function is called with the SYS_SAVE command.

SYSACT_ONBOOT
The new setting will not take effect immediately but should take effect after the terminal power is cycled. This value is placed into a list of pending changes that is saved to the non-volatile flash memory when this function is called with the SYS_SAVE command.

SYSACT_ALWAYS
The new setting will take effect immediately and after the terminal power is cycled. The current setting is changed and this value is also placed into a list of pending changes that is saved to the non-volatile flash memory when this function is called with the SYS_SAVE command.

The "{cmd}" argument specifies which setting will be modified. It is of type "syscmd," which is an enumerated type with the following legal values:

SYS_CONTRAST
Sets the display contrast using a scale from 0 to 255, with 255 the brightest. The value -1 may also be passed in to cause the current contrast value to be used. **Note that incrementing by one in the scale will not necessarily change the contrast value, and values returned by GetSystemSetting() may not match what was passed into SetSystemSetting()!** To change the contrast by one contrast increment, use the SetContrast() function and then pass in -1 for the contrast setting in this function to save to the flash.

SYS_BACKLIGHT
Sets the display backlight using a scale from 0 to 255, with 255 the brightest. The value -1 may also be passed in to cause the current backlight value to be used. **Note that incrementing by one in the scale will not necessarily change the backlight value, and values returned by GetSystemSetting() may not match what was passed into SetSystemSetting()!** To change the backlight by one backlight increment, use the SetBacklight() function and then pass in -1 for the backlight setting in this function to save to the flash.

SYS_VOLUME
For units that have an audio decoder, this sets the unit volume using a scale from 0 to 255, with 255 the loudest. The value -1 may also be passed in to cause the current volume to be used. Note that incrementing by one in the scale will not necessarily change the volume, and values returned by GetSystemSetting() may not match what was passed into SetSystemSetting()! To change the volume by one volume increment, use the SetVolume() function and then pass in -1 for the volume setting in this function to save to the flash.

SYS_NOTEAMPLITUDE
For units that have an audio decoder, this sets the amplitude of the generated note waveforms using a scale from 0 to 255, with 255 the loudest.

SYS_MODE
Change the current terminal mode (development mode on, development mode off, and default application). In this case, "{newvalue}" is of type "syscmd_mode," which is an enumerated type with the following legal values:

MODE_DEVELOFF
MODE_DEVELON
MODE_DEFAULTAPP

If the terminal does not have a default application, selecting MODE_DEFAULTAPP will be the same as MODE_DEVELOFF. As there is no "current" value for this setting, using SYSACT_ DONOW will not do anything, and using SYSACT_ ALWAYS will only set the pending save setting.

SYS_ORIENT
Change the display orientation (portrait or landscape). In this case, "{newvalue}" is of type "syscmd_orient," which is an enumerated type with four legal values:

ORIENT_PORTRAIT
ORIENT_LANDSCAPE
ORIENT_PORTRAIT2
ORIENT_LANDSCAPE2

SYS_KBDRPTDELAY
For systems with a keyboard interface, change the keyboard repeat delay, which is the amount of time after the key is pressed before the first repeat character is generated. In this case, "{newvalue}" is of type "syscmd_kbdrptdelay," which is an enumerated type with the following legal values:

KBDRPTDELAY_250	(Delay = 250 ms)
KBDRPTDELAY_500	(Delay = 500 ms)
KBDRPTDELAY_750	(Delay = 750 ms)
KBDRPTDELAY_1000	(Delay = 1000 ms)

SYS_KBDRPTRATE
For systems with a keyboard interface, change the keyboard repeat rate, which is the number of characters generated per second after repeat begins. In this case, "{newvalue}" is of type "syscmd_kbdrptrate," which is an enumerated type with the following legal values:

KBDRPTRATE_30_0	(30.0 cps)
KBDRPTRATE_26_7	(26.7 cps)
KBDRPTRATE_24_0	(24.0 cps)
KBDRPTRATE_21_8	(21.8 cps)
KBDRPTRATE_20_0	(20.0 cps)
KBDRPTRATE_18_5	(18.5 cps)
KBDRPTRATE_17_1	(17.1 cps)
KBDRPTRATE_16_0	(16.0 cps)
KBDRPTRATE_15_0	(15.0 cps)
KBDRPTRATE_13_3	(13.3 cps)
KBDRPTRATE_12_0	(12.0 cps)
KBDRPTRATE_10_9	(10.9 cps)
KBDRPTRATE_10_0	(10.0 cps)
KBDRPTRATE_9_2	(9.2 cps)
KBDRPTRATE_8_5	(8.5cps)
KBDRPTRATE_8_0	(8.0cps)
KBDRPTRATE_7_5	(7.5cps)
KBDRPTRATE_6_7	(6.7cps)
KBDRPTRATE_6_0	(6.0 cps)
KBDRPTRATE_5_5	(5.5 cps)
KBDRPTRATE_5_0	(5.0 cps)
KBDRPTRATE_4_6	(4.6 cps)
KBDRPTRATE_4_3	(4.3 cps)
KBDRPTRATE_4_0	(4.0 cps)
KBDRPTRATE_3_7	(3.7 cps)
KBDRPTRATE_3_3	(3.3 cps)

KBDRPTRATE_3_0 (3.0 cps)
KBDRPTRATE_2_7 (2.7 cps)
KBDRPTRATE_2_5 (2.5 cps)
KBDRPTRATE_2_3 (2.3 cps)
KBDRPTRATE_2_1 (2.1 cps)
KBDRPTRATE_2_0 (2.0 cps)

SYS_KEYRPTDELAY
For systems with a keypad interface, change the keypad repeat delay, which is the amount of time after the key is pressed before the first repeat character is generated. "{newvalue}" is an integer indicating the number of milliseconds for the delay. The minimum delay is 20 ms and the maximum delay is 2000 ms. The value is rounded to the nearest 20 ms increment.

SYS_KEYRPTPERIOD
For systems with a keypad interface, change the keypad repeat period, which is the amount of time between repeat characters after repeat has begun. "{newvalue}" is an integer indicating the number of milliseconds for the delay. The minimum delay is 20 ms and the maximum delay is 2000 ms. The value is rounded to the nearest 20 ms increment.

SYS_KEYCLICK
For systems with a keypad interface, enable or disable the audible key click, which is a short beep generated whenever a key is pressed. "{newvalue}" must be a boolean; TRUE enables the key click and FALSE disables it.

SYS_KEYRPT
For systems with a keypad interface, enable or disable the key repeat feature. "{newvalue}" must be a boolean; TRUE enables key repeat and FALSE disables it.

SYS_COM1BAUD, SYS_COM2BAUD,
SYS_COM3BAUD, SYS_COM4BAUD,
SYS_COM5BAUD, SYS_COM6BAUD,
SYS_COM7BAUD, SYS_COM8BAUD,
SYS_COM9BAUD, SYS_COM10BAUD
Set the baud rate for the selected serial interface. "{newvalue}" is of type "syscmd_baud," which is an enumerated type with the following legal values:

BAUD_115200
BAUD_57600
BAUD_38400
BAUD_19200
BAUD_14400
BAUD_9600
BAUD_4800
BAUD_2400
BAUD_1200
BAUD_600

SYS_COM1DATABITS, SYS_COM2DATABITS,
SYS_COM3DATABITS, SYS_COM4DATABITS,
SYS_COM5DATABITS, SYS_COM6DATABITS,
SYS_COM7DATABITS, SYS_COM8DATABITS,
SYS_COM9DATABITS, SYS_COM10DATABITS
Set the number of data bits for the selected serial interface. "{newvalue}" is of type "syscmd_databits," which is an enumerated type with two legal values:

DATABITS_7
DATABITS_8

SYS_COM1PARITY, SYS_COM2PARITY,
SYS_COM3PARITY, SYS_COM4PARITY,
SYS_COM5PARITY, SYS_COM6PARITY,
SYS_COM7PARITY, SYS_COM8PARITY,
SYS_COM9PARITY, SYS_COM10PARITY
Set the parity for the selected serial interface. "{newvalue}" is of type "syscmd_parity," which is an enumerated type with three legal values:

PARITY_ NONE
PARITY_ODD
PARITY_ EVEN

SYS_COM1STOPBITS, SYS_COM2STOPBITS,
SYS_COM3STOPBITS, SYS_COM4STOPBITS,
SYS_COM5STOPBITS, SYS_COM6STOPBITS,
SYS_COM7STOPBITS, SYS_COM8STOPBITS,
SYS_COM9STOPBITS, SYS_COM10STOPBITS
Set the number of stop bits for the selected serial interface. "{newvalue}" is of type "syscmd_stopbits," which is an enumerated type with two legal values:

STOPBITS_1
STOPBITS_2

SYS_COM1FLOWCONTROL,
SYS_COM2FLOWCONTROL,
SYS_COM3FLOWCONTROL,
SYS_COM4FLOWCONTROL,
SYS_COM5FLOWCONTROL,
SYS_COM6FLOWCONTROL,
SYS_COM7FLOWCONTROL,
SYS_COM8FLOWCONTROL,

SYS_COM9FLOWCONTROL,
SYS_COM10FLOWCONTROL
Set the flow control for the selected serial interface. "{newvalue}" is of type "syscmd_flowcontrol," which is an enumerated type with three legal values:

FLOWCONTROL_NONE
FLOWCONTROL_XON _OFF
FLOWCONTROL_RTS_CTS

RTS/CTS flow control is only supported for the EIA-232 interface.

SYS_COM1FLOWTIMEOUT,
SYS_COM2FLOWTIMEOUT
Set the flow control transmit timeout for the selected serial interface. "{newvalue}" is of type integer and can be any number from 0 to 65535 (zero meaning no timeout). It tells how many 20 ms intervals to wait (if necessary) to send any character before timing out.

SYS_IPADDRESS
For terminals with an Ethernet interface, set the IP address of the terminal. "{newvalue}" is a 4-byte array containing the new IP address.

SYS_IPSUBNET
For terminals with an Ethernet interface, set the subnet mask. "{newvalue}" is a 4-byte array containing the new subnet mask.

SYS_IPGATEWAY
For terminals with an Ethernet interface, set the gateway IP address. "{newvalue}" is a 4-byte array containing the new gateway IP address.

SYS_USEDRAWCACHE
Change the status of whether the unit uses draw caching to increase draw performance. "{newvalue}" is of type "drawcache_level". The values are:

CACHE_ALL
CACHE_OFF
CACHE_ENABLED
CACHE_EFFECTIVE_ENABLED

SYS_PASSWORD
Change the password used to enter the Power On Setup utility. "{newvalue}" is a unibyte array containing exactly sixteen elements. To allow the use of multiple input methods to access Power On Setup (on units with some combination of keypad, keyboard, and touch screen), use the following constants in place of the corresponding Power On Setup keys on the input source (i.e., for the Enter key on the keyboard, use POSKEY_ENTER rather than KEY_ENTER). Otherwise, key codes may be used to define the appropriate key sequence. However, all sequences must have a POSKEY_ENTER in them, and keys following this key are ignored. Note that for the keyboard, only the bits in KEY_ASCII_MASK are used. Available POSKEY keys on all input devices are as follows:

POSKEY_UP
POSKEY_DOWN
POSKEY_LEFT
POSKEY_RIGHT
POSKEY_ENTER

The following POSKEY keys are also available on keyboards and some keypads:

POSKEY_ESC
POSKEY_0
POSKEY_1
POSKEY_2
POSKEY_3
POSKEY_4
POSKEY_5
POSKEY_6
POSKEY_7
POSKEY_8
POSKEY_9

As there is no "current" value for this setting, using SYSACT_DONOW will not do anything, and using SYSACT_ALWAYS will only set the pending save setting.

SYS_USEPASSWORD
Specify whether or not a password is required to get full functionality in the Power On Setup utility. "{newvalue}" must be of type "syscmd_usepassword." As there is no "current" value for this setting, using SYSACT_DONOW will not do anything, and using SYSACT_ALWAYS will only set the pending save setting. The values are:

USEPASSWORD_OFF
USEPASSWORD_ON

SYS_TEMPERATURE
Set this value to the current temperature to calibrate the temperature reported by the terminal. "{newvalue}" must be a float.

SYS_FEEDBACK_TYPE
Specify the method of feedback that the unit will use to report development information as well as any severe runtime errors through. "{newvalue}" must be of type "syscmd_feedback" and must at least have one of the methods selected. The values are:

FBTYPE_SERIAL
FBTYPE_VIDEO
FBTYPE_UDP

The values can be combined together to produce any combination desired (using the "AND" operator as in "FBTYPE_SERIAL AND FBTYPE_VIDEO").

SYS_FEEDBACK_IPADDRESS
If the unit is Ethernet enabled and using UDP feedback, this option specifies the IP address to send all UDP feedback information to. "{newvalue}" is therefore a 4-byte array containing the IP address.

SYS_FEEDBACK_FPORT
If the unit is Ethernet enabled and using UDP feedback, this option specifies the foreign port number to send all UDP feedback packets to. "{newvalue}" in this case is an integer and can be any number between 0 and 65535.

SYS_FATALREBOOTTIMEOUT
This is the number of 20 ms increments to wait after hitting a major system error before rebooting. A value of 0 (the default) is interpreted as indefinitely. Hopefully, you will never need this.

Two additional values are also legal for "{cmd}:"

SYS_KEYPADBACKLIGHT
For systems with a keypad interface and keypad backlight, turn the backlight on or off. "{newvalue}" must be a boolean; TRUE turns the backlight on and FALSE turns it off.

SYS_AUTOSHIFT
For systems with a keypad interface and Auto Shift capability (such as the QTERM-G55), enable or disable the Auto Shift feature. When Auto Shift is off, the Shift Key behaves like any other key. "{newvalue}" must be a boolean; TRUE enables the Auto Shift feature and FALSE turns it off.

SYS_SAVE
Causes the terminal settings that are pending to be saved to the terminal's non-volatile flash memory. "{newvalue}" is a boolean value that determines whether the system is reset after the new terminal settings are saved. A value of TRUE will cause the system to reset (RESET_NORMAL). The parameter "{action}" is ignored.

SYS_CLEAR
Clears the terminal settings that are pending without saving them to the terminal's non-volatile flash memory. "{newvalue}" must be set to 0. The parameter "{action}" is ignored.

SYS_KEYPADGATEDELAY
For systems with a keypad interface, change the keypad hardware gate delay in the keypad scanning routine. This is the delay between writing the row latch and reading the column latch. Some keypads which are highly resistive and/or capacitive may require longer delays to read the proper row and column. The actual value is dimensionless, but higher values will yield longer delays. If your keypad is not working, try setting to a higher value. SETTING THIS TO AN INCORRECT VALUE (USUALLY TOO LOW) MAY PREVENT THE KEYPAD FROM FUNCTIONING PROPERLY! "{value}" is an integer indicating the delay. Values from 1 to 100 are acceptable.

SYS_AUTOPOWER
For systems with a keypad interface and Auto Power capability (such as the QTERM-G55), set the mode of the Auto Power feature. Value is of type syscmd_powerkeymode. The values are:

POWERKEY_NORMALKEY
POWERKEY_AUTOPOWER

IMPORTANT NOTE:
Changes made to the terminal settings via SetSystemSetting() using SYSACT_ALWAYS and SYSACT_ONBOOT are not permanent (saved to the non-volatile flash memory) until SetSystemSetting() is called with the SYS_SAVE command. Many changes may be made via multiple calls to SetSystemSetting() before they are saved to the flash using the SYS_SAVE command.

The intent of this command is two-fold: to allow the creation of a custom terminal configuration facility at the application level (similar to the built-in Power On Setup facility) and to allow changing settings during run-time operation. By creating a custom terminal configuration facility in an application, any desired subset of the terminal settings and other application specific settings can be made available to the user.

4.18.15 GetSystemSetting

Syntax:

```
getsystemsetting(cmd as syscmd, var as ->
reference to anytype, which as ->
syscmd_readfrom)
```

Parameters:

{cmd}
A value of type "syscmd" (see section 4.18.14, "SetSystemSetting.") that determines which terminal control setting will be returned in "{var}."

{value}
The variable that will receive the value of the system setting being retrieved.

{which}
A value of type "syscmd_readfrom" that determines the mode of the setting that is read.

Description:

This function retrieves a value for the terminal setting specified by "{cmd}." This parameter is an enumerated type described in section 4.18.14, "SetSystemSetting."

There are three versions of each setting; the current setting, the pending setting, and the saved setting. The current setting is the setting that the terminal is using. The pending setting is any setting that was changed using SetSystemSetting() and is awaiting a SYS_SAVE command in order to be saved to the flash. The pending setting may or may not be the current setting. The saved setting is the value of the setting that is saved in the non-volatile flash memory. Each of these values is accessible through the "{which}" parameter. This parameter must be of type "syscmd_readfrom" and has the following possible values:

SYSREAD_CURRENT
The current value of the setting being used by the firmware.

SYSREAD_PENDINGSAVE
The value of the setting that is pending a save command.

SYSREAD_SAVED
The value of the setting that is actually saved.

The variable "{value}" is passed as a reference and receives the value of the setting. The "{value}" variable must be of the correct data type for each value of "{cmd}" as follows (see also section 4.18.14, "SetSystemSetting"):

SYS_CONTRAST	integer
SYS_BACKLIGHT	integer
SYS_VOLUME	integer
SYS_NOTEAMPLITUDE	integer
SYS_MODE	syscmd_mode
SYS_ORIENT	syscmd_orient
SYS_KBDRPTDELAY	syscmd_kbdrptdelay
SYS_KBDRPTRATE	syscmd_kbdrptrate
SYS_KEYRPTDELAY	integer
SYS_KEYRPTPERIOD	integer
SYS_KEYCLICK	boolean
SYS_KEYRPT	boolean
SYS_COM1BAUD	syscmd_baud
SYS_COM1DATABITS	syscmd_databits
SYS_COM1PARITY	syscmd_parity
SYS_COM1STOPBITS	syscmd_stopbits
SYS_COM1FLOWCONTROL	syscmd_flowcontrol
SYS_COM1FLOWTIMEOUT	integer
SYS_COM2BAUD	syscmd_baud
SYS_COM2DATABITS	syscmd_databits
SYS_COM2PARITY	syscmd_parity
SYS_COM2STOPBITS	syscmd_stopbits
SYS_COM2FLOWCONTROL	syscmd_flowcontrol
SYS_COM2FLOWTIMEOUT	integer
SYS_IPADDRESS	byte array
SYS_IPSUBNET	byte array
SYS_IPGATEWAY	byte array
SYS_USEDRAWCACHE	drawcache_level
SYS_PASSWORD	unibyte array
SYS_USEPASSWORD	syscmd_usepassword
SYS_TEMPERATURE	float
SYS_FEEDBACK_TYPE	syscmd_feedback
SYS_FEEDBACK_IPADDRESS	byte array
SYS_FEEDBACK_FPORT	integer
SYS_FATALREBOOTTIMEOUT	integer
SYS_KEYPADBACKLIGHT	boolean
SYS_AUTOSHIFT	boolean
SYS_KEYPADGATEDELAY	integer
SYS_AUTOPOWER	syscmd_powerkeymode

The following "{cmd}" values are not legal for this function:

SYS_SAVE
SYS_CLEAR

Note that the following values for "{cmd}" have no defined current value:

SYS_MODE
SYS_PASSWORD
SYS_USEPASSWORD

4.18.16 SoftReset

Syntax:

```
softreset(rst as rstmode)
```

Parameters:

{rst}
A value of type "rstmode" indicating how the reset is to be performed.

Description:

This function causes the terminal to undergo a warm reboot cycle. "{rst}" can take the following values:

RESET_NORMAL
Causes the unit to perform a reset as if the power had been cycled. If the unit is in development mode, it waits for an application to be loaded.

RESET_LOADAPP
Causes the unit to perform a reset and go directly into the download application mode.

RESET_ENTER_BL
Causes the unit to perform a reset, entering into the bootloader where the firmware can be upgraded serially.

RESET_ENTER_POS
Causes the unit to perform a reset, entering the "Power On Setup" utility upon reboot.

RESET_TOUCH_CAL
Causes the unit to perform a reset, entering the screens used to calibrate the touch screen.

4.18.17 GetRandomNum

Syntax:

```
getrandomnum() returns float
```

Description:

This function returns a pseudo-random number. The return value is always a floating point number between 0 to 1.

4.18.18 SeedRandomNum

Syntax:

```
seedrandomnum()
```

Description:

This function seeds the random number generator with a value from an internal hardware timer. Calling this function at a non-deterministic time (such as a user input event) produces the best pseudo-random numbers (obtained with calls to GetRandomNum()).

4.18.19 SetSeedRandomNum

Syntax:

```
setseedrandomnum(seed as integer)
```

Description:

This function seeds the random number generator with a specific value, enabling "pseudo-random" sequences to be produced (and reproduced). This is most useful for debugging.

4.18.20 WatchdogEnable

Syntax:

```
watchdogenable(enable as boolean, timeout ->
as integer)
```

Parameters:

{enable}
A boolean value specifying whether to enable (use "true") or disable (use "false") the watchdog timer.

{timeout}
The desired number of 20 ms intervals that should elapse before a system reset occurs (1 = 20ms).

Description:

This function enables/disables the software watchdog timer. The purpose of the watchdog timer is to cause a system reset if something goes fatally wrong in the system software. The time will cause a system reset if the timeout period expires. For normal operation, the timer should be periodically reset using the WatchdogReset() API function.

4.18.21 WatchdogReset

Syntax:

```
watchdogreset()
```

Description:

This function restarts the timeout period in the watchdog timer. If the watchdog timer is enabled (via the WatchdogEnable() API function), this function must be called periodically to reset the timer or the system will be reset when the timeout period elapses. This function has no effect if the watchdog timer is not enabled.

4.18.22 GetProfileTick

Syntax:

```
getprofiletick() returns integer
```

Description:

This function is used to profile execution speed. When this function is called, it returns the number of profiling ticks that have occurred since the last time the function was called. One profile tick is 1/32768 s.

4.18.23 DelayMS

Syntax:

```
delayms(delay as integer)
```

Parameters:

{delay}
The desired delay in milliseconds.

Description:

This function delays the requested number of milliseconds before returning.

4.18.24 GetBinaryResource

Syntax:

```
getbinaryresource (resourceID as integer)->
returns byte[]
```

Parameters:

{resourceID}
`The identifier of the binary resource.`

Description:

This function returns a byte array containing the data of a binary resource that was included with the application.

4.18.25 SetArrayData

Syntax:

```
setarraydata(arr[] reference? to ->
sametype!, index as integer, srcdata[] ->
as reference? to sametype!, srcindex as ->
integer, len as integer)
```

Parameters:

{arr}
An array variable already holding data.

{index}
The location in "{arr}" to start inserting the "{srcdata}."

{srcdata}
An array containing the data to be inserted into "{arr}."

{srcindex}
The index into "{srcdata}" indicating the start of the data to be inserted into "{arr}."

{len}
The length of the data (or -1 for all) to be copied from "{srcdata}" to "{arr}."

Description:

This function copies data from "{srcdata}" into "{arr}." It is mostly used to improve efficiency when only parts of an array change.

4.18.26 CreateCRCTable

Syntax:

```
createcrctable(width as integer, poly as ->
integer, reflect as boolean) returns ->
integer[]
```

Parameters:

{width}
The number of bits (minus one) used in the polynomial for the desired CRC.

{poly}
The polynomial to be used in calculating the CRC (the most significant bit is always implicitly set, for example; x^16+x^12+x^5+x^0 is 0x1021).

{reflect}
Whether or not to use a reflected algorithm (reflects incoming data).

Description:

This function is used to generate a CRC table, which is used in calculating CRCs. More information on specifying CRC algorithms can be found in "*A Painless Guide to CRC Error Detection Algorithms*" by Ross N. Williams. Following are examples of some common algorithms:

CRC-16/CITT : "{width}" = 16, "{poly}" = 0x1021, "{reflect}" = FALSE

CRC-32: "{width}" = 32, "{poly}" = 0x04C11DB7, "{reflect}" = TRUE

CRC-16: "{width}" = 16, "{poly}" = 0x8005, "{reflect}" = TRUE

Modbus : "{width}" = 16, "{poly}" = 0x8005, "{reflect}" = TRUE

4.18.27 CalculateCRC

Syntax:

```
calculatecrc(table[] as reference? to ->
integer, width as integer, reflect as ->
boolean, crcin as integer, reflectout as ->
boolean, xoronout as integer, data[] as ->
reference? to byte) returns integer
```

Parameters:

{table}
The CRC table calculated from CreateCRCTable.

{width}
The number of bits (minus one) used in the polynomial for the desired CRC.

{reflect}
Whether or not to use a reflected algorithm (reflects incoming data).

{crcin}
The initial CRC value.

{reflectout}
Whether or not to reflect the final CRC before xor-ing it with "{xoronout}."

{xoronout}
The value to xor the final CRC to before returning.

{data}
The array of byte data for which to calculate the CRC.

Description:

This function calculates the CRC for a set of data using the parameters to describe the algorithm. More information on specifying CRC algorithms can be found in "*A Painless Guide to CRC Error Detection Algorithms*" by Ross N. Williams.

To continue calculating the CRC for a set of data in multiple parts, call the function subsequently with the initial CRC ("{crcin}") set to the calculated CRC from the previous call. In all but the final call, "{xoronout}" should be 0 and "{reflectout}" should be FALSE.

Following are examples of some common algorithms:

CRC-16/CITT : "{width}" = 16, "{reflect}" = FALSE, "{crcin}" = 0xFFFF, "{reflectout}" = FALSE, "{xoronout}" = 0x0000

CRC-32 : "{width}" = 32, "{reflect}" = TRUE, "{crcin}" = 0xFFFFFFFF, "{reflectout}" = TRUE, "{xoronout}" = 0xFFFFFFFF

CRC-16 : "{width}" = 16, "{reflect}" = TRUE, "{crcin}" = 0x0, "{reflectout}" = TRUE, "{xoronout}" = 0x0

Modbus : "{width}" = 16, "{reflect}" = TRUE, "{crcin}" = 0xFFFF, "{reflectout}" = TRUE, "{xoronout}" = 0x0

4.18.28 ZlibCompress

Syntax:

```
zlibcompress(out[] as reference to byte, ->
in[] as reference? to byte)
```

Parameters:

{out}
A data array to accept the compressed data. This array is sized to fit the compressed data.

{in}
The uncompressed data.

Description:

This function is used to compress data using the zlib library. The zlib format is not fully compatible with other zip formats although it is possible to convert a zlib file to one that is compatible with the gzip file format. For more information visit the zlib web site.

4.18.29 ZlibDecompress

Syntax:

```
zlibdecompress(out[] as reference to byte,->
in[] as reference? to byte)
```

Parameters:

{out}
A data array to accept the compressed data. This array is sized to fit the compressed data.

{in}
The uncompressed data.

Description:

This function is used to compress data using the zlib library. The zlib format is not fully compatible with other zip formats although it is possible to convert a zlib file to one that is compatible with the gzip file format. For more information visit the zlib web site.

4.18.30 SetPalette

Syntax:

```
setpalette(red[] as reference? to byte, ->
green[] as reference? to byte, blue[] as ->
reference? to byte)
```

Parameters:

{red}
Eight bit red color palette (256 byte array).

{green}
Eight bit green color palette (256 byte array).

{blue}
Eight bit blue color palette (256 byte array).

Description:

This function sets the color palette for many terminal displays. Not all terminals support this API. This function is considered an advanced function. Please contact Technical Support if you need more details on this API.

APPENDIX A

BUILT-IN CONSTANTS AND DEFINED TYPES

A.1 Constants

(constant pi := 3.1415926536 as float)

Defined Type	Values
arrayOp	ARRAY_CONVERT ARRAY_PALETTE ARRAY_REVERSE ARRAY_PFIELD
arrayHandle	NULL_HANDLE
backlight_adjust	BACKLIGHT_ON BACKLIGHT_OFF BACKLIGHT_LIGHTER BACKLIGHT_DARKER
comm	COM1 COM2 COM3 COM4 COM5 COM6 COM7 COM8 COM9 COM10 COM_INVALID
contrast_adjust	CONTRAST_LIGHTER CONTRAST_DARKER
drawcache_level	CACHE_OFF CACHE_ALL CACHE_ENABLED CACHE_EFFECTIVE_ENABLED
ellipse_flags	ELLIPSE_NORMAL ELLIPSE_FILL ELLIPSE_CONNECT_CENTER ELLIPSE_CONNECT_ENDS
enable_info	GET_ENABLED GET_ZENABLED
fake_key	KEY_PUSH KEY_REPEAT KEY_RELEASE
fake_screen	SCREEN_PUSH SCREEN_MOVE SCREEN_RELEASE
filedesc	FILE_NONE
file_flags	FILE_NOFLAGS FILE_READ FILE_WRITE FILE_TEXT FILE_BINARY FILE_NO_CREATE FILE_APPEND
fileinfo_flags	FILEINFO_ISNOT FILEINFO_ISTEXT FILEINFO_ISBIN FILEINFO_ISDIR FILEINFO_ISOPEN
font_flags	FONT_NORMAL FONT_VERTICAL FONT_INVERSE FONT_DRAW_HPARTIAL FONT_DRAW_VPARTIAL FONT_DRAWSPACE FONT_DRAWLINEBREAKS FONT_DRAWWORDBREAKS FONT_NOSOFTBREAKS FONT_HBASELINE FONT_HFIT FONT_HABS FONT_HLEFT FONT_HCENTER FONT_HRIGHT FONT_VBASELINE FONT_VFIT FONT_VABS FONT_VTOP FONT_VCENTER FONT_VBOTTOM
gpio_action	GPIO_SET GPIO_CLEAR GPIO_TOGGLE

Defined Type	Values
gpio_pin	PIN0 PIN1 PIN2 PIN3 PIN4 PIN5 PIN6 PIN7 PIN_ALL
hwinfo	HW_ETHERNET HW_TOUCH HW_KEYPAD HW_KEYBOARD HW_TEMPCONTROL HW_CLOCK HW_DISPLAY HW_DEFAULTAPP HW_MACADDRESS HW_AUDIOCODEC HW_GPIO HW_CPU HW_COMLIST HW_MEMORY HW_KEYPADBACKLIGHT HW_POWER_OVER_ETHERNET
keyboardcontrol	KEYBOARD_PRESENT KEYBOARD_WRITECOMMAND BYTE
ledcmd	LED_ON LED_OFF LED_TOGGLE
multiline_flags	MULTILINE_MASK_BREAK MULTILINE_NOBREAK MULTILINE_SOFTBREAK MULTILINE_WORDBREAK MULTILINE_LINEBREAK MULTILINE_MASK_WIDTH MULTILINE_PARTIAL_WIDTH MULTILINE_NONE_WIDTH MULTILINE_MASK_HEIGHT MULTILINE_PARTIAL_HEIGHT MULTILINE_NONE_HEIGHT
netprotocol	NET_TCP NET_UDP NET_RAW

Defined Type	Values
poly_flags	POLY_NORMAL POLY_FILL POLY_NOCONNECT
position_info	GET_X GET_Y GET_WIDTH GET_HEIGHT GET_ORIGIN_X GET_ORIGIN_Y GET_XGLOBAL GET_YGLOBAL
rstmode	RESET_NORMAL RESET_ENTER_BL RESET_LOADAPP RESET_ENTER_POS RESET_TOUCH_CAL
servercomm	NULL_SERVER
syscmd	SYS_SAVE SYS_CLEAR SYS_CONTRAST SYS_BACKLIGHT SYS_MODE SYS_ORIENT SYS_KBDRPTDELAY SYS_KBDRPTRATE SYS_KEYRPTDELAY SYS_KEYRPTPERIOD SYS_KEYCLICK SYS_KEYRPT SYS_COM1BAUD SYS_COM1DATABITS SYS_COM1PARITY SYS_COM1STOPBITS SYS_COM1FLOWCONTROL SYS_COM2BAUD SYS_COM2DATABITS SYS_COM2PARITY SYS_COM2STOPBITS SYS_COM2FLOWCONTROL SYS_IPADDRESS SYS_IPSUBNET SYS_IPGATEWAY SYS_USEDRAWCACHE SYS_PASSWORD SYS_USEPASSWORD SYS_TEMPERATURE SYS_COM3BAUD

Defined Type	Values
syscmd (continued)	SYS_COM8STOPBITS SYS_COM8FLOWCONTROL SYS_COM9BAUD SYS_COM9DATABITS SYS_COM9PARITY SYS_COM9STOPBITS SYS_COM9FLOWCONTROL SYS_COM10BAUD SYS_COM10DATABITS SYS_COM10PARITY SYS_COM10STOPBITS SYS_COM10FLOWCONTROL SYS_FEEDBACK_TYPE SYS_FEEDBACK_IPADDRESS SYS_FEEDBACK_FPORT SYS_COM1FLOWTIMEOUT SYS_COM2FLOWTIMEOUT SYS_COM3DATABITS SYS_COM3PARITY SYS_COM3STOPBITS SYS_COM3FLOWCONTROL SYS_COM4BAUD SYS_COM4DATABITS SYS_COM4PARITY SYS_COM4STOPBITS SYS_COM4FLOWCONTROL SYS_COM5BAUD SYS_COM5DATABITS SYS_COM5PARITY SYS_COM5STOPBITS SYS_COM5FLOWCONTROL SYS_COM6BAUD SYS_COM6DATABITS SYS_COM6PARITY SYS_COM6STOPBITS SYS_COM6FLOWCONTROL SYS_COM7BAUD SYS_COM7DATABITS SYS_COM7PARITY SYS_COM7STOPBITS SYS_COM7FLOWCONTROL SYS_COM8BAUD SYS_COM8DATABITS SYS_COM8PARITY SYS_COM1FLOWTIMEOUT SYS_COM2FLOWTIMEOUT SYS_COM3FLOWTIMEOUT SYS_COM4FLOWTIMEOUT SYS_COM5FLOWTIMEOUT SYS_COM6FLOWTIMEOUT

Defined Type	Values
syscmd (continued)	SYS_COM7FLOWTIMEOUT SYS_COM8FLOWTIMEOUT SYS_COM9FLOWTIMEOUT SYS_COM10FLOWTIMEOUT SYS_VOLUME SYS_NOTEAMPLITUDE SYS_FATALBOOTTIMEOUT SYS_KEYPADBACKLIGHT SYS_AUTOSHIFT SYS_COM11BAUD SYS_COM11DATABITS SYS_COM11PARITY SYS_COM11STOPBITS SYS_COM11FLOWCONTROL SYS_COM11FLOWTIMEOUT SYS_KEYPADGATEDELAY SYS_AUTOPOWER
syscmd_action	SYSACT_DONOW SYSACT_ONBOOT SYSACT_ALWAYS
syscmd_baud	BAUD_115200 BAUD_57600 BAUD_38400 BAUD_19200 BAUD_14400 BAUD_9600 BAUD_4800 BAUD_2400 BAUD_1200 BAUD_600 BAUD_300
syscmd_databits	DATABITS_7 DATABITS_8
syscmd_feedback	FBTYPE_SERIAL FBTYPE_VIDEO FBTYPE_UDP
syscmd_flowcontrol	FLOWCONTROL_NONE FLOWCONTROL_XON_XOFF FLOWCONTROL_RTS_CTS FLOWCONTROL_DTR_DSR
syscmd_kbdrptdelay	KBDRPTDELAY_250 KBDRPTDELAY_500 KBDRPTDELAY_750 KBDRPTDELAY_1000

Defined Type	Values
syscmd_kbdrptrate	KBDRPTRATE_30_0 KBDRPTRATE_26_7 KBDRPTRATE_24_0 KBDRPTRATE_21_8 KBDRPTRATE_20_0 KBDRPTRATE_18_5 KBDRPTRATE_17_1 KBDRPTRATE_16_0 KBDRPTRATE_15_0 KBDRPTRATE_13_3 KBDRPTRATE_12_0 KBDRPTRATE_10_9 KBDRPTRATE_10_0 KBDRPTRATE_9_2 KBDRPTRATE_8_5 KBDRPTRATE_8_0 KBDRPTRATE_7_5 KBDRPTRATE_6_7 KBDRPTRATE_6_0 KBDRPTRATE_5_5 KBDRPTRATE_5_0 KBDRPTRATE_4_6 KBDRPTRATE_4_3 KBDRPTRATE_4_0 KBDRPTRATE_3_7 KBDRPTRATE_3_3 KBDRPTRATE_3_0 KBDRPTRATE_2_7 KBDRPTRATE_2_5 KBDRPTRATE_2_3 KBDRPTRATE_2_1 KBDRPTRATE_2_0
syscmd_mode	MODE_DEVELOFF MODE_DEVELON MODE_DEFAULTAPP
syscmd_orient	ORIENT_LANDSCAPE ORIENT_PORTRAIT ORIENT_LANDSCAPE2 ORIENT_PORTRAIT2
syscmd_parity	PARITY_NONE PARITY_ODD PARITY_EVEN

Defined Type	Values
syscmd_poskeys	POSKEY_UP POSKEY_DOWN POSKEY_LEFT POSKEY_RIGHT POSKEY_ENTER POSKEY_ESC POSKEY_0 POSKEY_1 POSKEY_2 POSKEY_3 POSKEY_4 POSKEY_5 POSKEY_6 POSKEY_7 POSKEY_8 POSKEY_9
syscmd_powerkeymode	POWERKEY_NORMALKEY POWERKEY_AUTOPOWER
syscmd_readfrom	SYSREAD_CURRENT SYSREAD_PENDINGSAVE SYSREAD_SAVED
syscmd_stopbits	STOPBITS_1 STOPBITS_2
syscmd_usedrawcache	DRAWCACHE_ON DRAWCACHE_OFF
syscmd_usepassword	USEPASSWORD_OFF USEPASSWORD_ON
typeval	INTEGER_TYPE FLOAT_TYPE BOOLEAN_TYPE BYTE_TYPE UNIBYTE_TYPE ARRAY_TYPE OBJREF_TYPE
volume_adjust	VOLUME_LOUDER VOLUME_QUIETER
weekday	SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY

A.2 Tool Types

Defined Type	Values
GuiCursors	CSR_ALL CSR_UPDOWN CSR_LEFTRIGHT CSR_UPLEFT CSR_UPRIGHT CSR_NONE CSR_DOWNRIGHT CSR_DOWNLEFT CSR_BLOCK CSR_SELECT CSR_DELETE CSR_PLUS CSR_NOLINETO CSR_OBJECTFIXED

A.3 Colors

Colors	
RGB_BLACK	RGB_BLUE
RGB_BROWN	RGB_BURNTORANGE
RGB_CYAN	RGB_DKBROWN
RGB_DKGRAY	RGB_DKSTEELGRAY
RGB_FORESTGREEN	RGB_GRAY
RGB_GREEN	RGB_KELLYGREEN
RGB_LTBROWN	RGB_MAGENTA
RGB_MAROON	RGB_MIDNIGHTBLUE
RGB_MOSSGREEN	RGB_NAVY
RGB_ORANGE	RGB_PURPLE
RGB_RED	RGB_STEELGRAY
RGB_VIOLET	RGB_WARMGRAY
RGB_WHITE	RGB_YELLOW
COL_0 through COL_255	

A.4 Key Codes

Key Codes
KEY_ANY
KEY_NONE
KEY_KEYPAD
KEY_CAPS_LOCK
KEY_NUM_LOCK
KEY_SCROLL_LOCK
KEY_SHIFT
KEY_CTRL
KEY_ALT
KEY_SPECIAL
KEY_ASCII_MASK
KEY_SPACE
KEY_BACKSPACE
KEY_TAB
KEY_UPARROW
KEY_DOWNARROW
KEY_LEFTARROW
KEY_RIGHTARROW
KEY_ENTER
KEY_ESCAPE
KEY_DELETE
KEY_INSERT
KEY_HOME
KEY_END
KEY_PAGEUP
KEY_PAGEDOWN
KEY_NUMPAD_5
KEY_PRINTSCREEN
KEY_PAUSE
KEY_OS
KEY_MENU
KEY_F1

Key Codes
KEY_F2
KEY_F3
KEY_F4
KEY_F5
KEY_F6
KEY_F7
KEY_F8
KEY_F9
KEY_F10
KEY_F11
KEY_F12
KEY_EXCLAMATION
KEY_QUOTE
KEY_POUND
KEY_DOLLAR
KEY_PERCENT
KEY_AMPERSTAND
KEY_APOSTROPHE
KEY_OPEN_PAREN
KEY_CLOSE_PAREN
KEY_ASTERISK
KEY_PLUS
KEY_COMMA
KEY_HYPHEN
KEY_PERIOD
KEY_SLASH
KEY_0
KEY_1
KEY_2
KEY_3
KEY_4
KEY_5
KEY_6

Key Codes
KEY_7
KEY_8
KEY_9
KEY_COLON
KEY_SEMICOLON
KEY_LESS_THAN
KEY_EQUALS
KEY_GREATER_THAN
KEY_QUESTION
KEY_AT
KEY_A
KEY_B
KEY_C
KEY_D
KEY_E
KEY_F
KEY_G
KEY_H
KEY_I
KEY_J
KEY_K
KEY_L
KEY_M
KEY_N
KEY_O
KEY_P
KEY_Q
KEY_R
KEY_S
KEY_T
KEY_U
KEY_V
KEY_W

Key Codes
KEY_X
KEY_Y
KEY_Z
KEY_OPEN_BRACKET
KEY_BACKSLASH
KEY_CLOSE_BRACKET
KEY_CARET
KEY_UNDERSCORE
KEY_GRAVE
KEY_LCASE_A
KEY_LCASE_B
KEY_LCASE_C
KEY_LCASE_D
KEY_LCASE_E
KEY_LCASE_F
KEY_LCASE_G
KEY_LCASE_H
KEY_LCASE_I
KEY_LCASE_J
KEY_LCASE_K

Key Codes
KEY_LCASE_L
KEY_LCASE_M
KEY_LCASE_N
KEY_LCASE_O
KEY_LCASE_P
KEY_LCASE_Q
KEY_LCASE_R
KEY_LCASE_S
KEY_LCASE_T
KEY_LCASE_U
KEY_LCASE_V
KEY_LCASE_W
KEY_LCASE_X
KEY_LCASE_Y
KEY_LCASE_Z
KEY_OPEN_BRACE
KEY_PIPE
KEY_CLOSE_BRACE
KEY_TILDE

APPENDIX B

EXCEPTION LIST

This appendix lists the names and associated descriptions of all possible exceptions. All non-fatal exception names begin with the prefix EXCEPT_.

B.1 Special Exceptions

EXCEPT_NONE	No String
EXCEPT_USER	No String

B.2 Memory Exceptions

EXCEPT_NOMEM	“Unable to allocate memory”
EXCEPT_NODCMEM	“Draw cache memory exhausted - reducing level”

B.3 Message System Exceptions

EXCEPT_NOPOST	“Unable to post message - queue is full”
EXCEPT_NOSYSMSG	“Unable to initiate a system message”
EXCEPT_BADREGMSG	“Not a registerable message”

B.4 Font Exceptions

EXCEPT_NOMODEVERT	“Vertical text unsupported by font”
EXCEPT_NOMODEHORZ	“Horizontal text unsupported by font”
EXCEPT_GLYPHRETR	“Unable to retrieve TTF glyph”
EXCEPT_GLYPHTRANS	“Unable to transform TTF glyph”
EXCEPT_GLYPHBMP	“Unable to render TTF glyph”
EXCEPT_CLOSEFACE	“Unable to release TTF face”
EXCEPT_OPENFACE	“Unable to initialize requested TTF face”
EXCEPT_MISSCMAP	“No appropriate TTF character map”
EXCEPT_CHANGEPT	“Unable to set requested TTF point size”
EXCEPT_GLYPHCOPY	“Unable to copy TTF glyph”
EXCEPT_NOSCALABLE	“TTF not scalable”
EXCEPT_NOSPACE	“Text does not fit as specified in area”

B.5 Drawing Exceptions

EXCEPT_NOTDRAWING	“Unable to perform while not drawing”
EXCEPT_DRAWING	“Unable to perform while drawing”
EXCEPT_CORRUPT_BMP	“Corrupted bitmap”
EXCEPT_UNSUP_BMP	“Unsupported bitmap”
EXCEPT_NONDRAWN_OBJ	“Object is non-drawable”
EXCEPT_ROOT_OBJ	“Unable to change root container information”
EXCEPT_FOCALDIST	“Illegal focal distance/ edge distance ratio”
EXCEPT_NOSTARTPT	“Unable to find ellipse start point”
EXCEPT_NOENDPT	“Unable to find ellipse end point”
EXCEPT_SCANFILLERROR	“Scanfill of area unsuccessful”
EXCEPT_MISMATCHPTS	“Mismatch in number of points”
EXCEPT_NOGETPIXMAP	“Unable to capture children in object pixel map”

B.6 Array Exceptions

EXCEPT_ARR_NOREDIM	"Particular array is not redimmable"
EXCEPT_NOARRSUPPORT	"No support for action on array"
EXCEPT_ARRINDEX	"Illegal index into an array"

B.7 Z-Order Exceptions

EXCEPT_NOATTACHABLE	"Impossible attachment request"
EXCEPT_NOZACTION	"Unable to queue up Z-order change"
EXCEPT_OBJNOTCONTAINER	"Object is not container"

B.8 Miscellaneous Exceptions

EXCEPT_INVAL_SIZE	"Invalid size"
EXCEPT_PLAYNOTE	"Unable to play note"
EXCEPT_BADPERSIST	"Unable to persist a non-variable"
EXCEPT_BADLEDNUM	"Invalid LED number"
EXCEPT_NORETHROW	"No exception to rethrow"
EXCEPT_NOFINDOBJ	"Unable to find object by name"
EXCEPT_NOFINDPROP	"Unable to find property by name"
EXCEPT_NOVARSTRING	"Unable to stringify given variable"
EXCEPT_BADPROPVAL	"Illegal value given by string"
EXCEPT_INVAL_OBJREF	"Invalid objref (possibly empty?)"
EXCEPT_BADSET	"Attempt to set non-setable variable"
EXCEPT_HWARE_UNAVAIL	"Hardware unavailable"
EXCEPT_BAD_FORMAT	"Bad format string"
EXCEPT_BADVALUE	"Value is not within expected range"
EXCEPT_BAD_ARRSIZE	"Array has incorrect size"
EXCEPT_MISSING_ENTER	"No enter key specified in password"
EXCEPT_HWARE_INUSE	"Hardware currently in use"
EXCEPT_INVAL_TYPE	"Invalid type"
EXCEPT_INVAL_HANDLE	"Invalid array handle"
EXCEPT_INVAL_ARROP	"Invalid array operation"
EXCEPT_DATA_CORRUPT	"Corrupted data"
EXCEPT_ZLIBFAULT	"ZLib library error"
EXCEPT_ZLIB_ADLER	"Missing or partial adler for compressed data"
EXCEPT_TFT_CONTRAST	"TFT displays do not support a contrast value"

B.9 Communications/Networking Exceptions

EXCEPT_NOCHANNEL	"Transmission channel not setup"
EXCEPT_NOFREECHANNEL	"No free transmission channel"
EXCEPT_CHANNELUSED	"Transmission channel is in use"
EXCEPT_SEND	"Transmission error"
EXCEPT_BADIP	"Bad IP address"
EXCEPT_NOTALLOWED	"Permission Denied"
EXCEPT_BADSERIAL	"Bad serial port identifier"
EXCEPT_BADSERIALSET	"Unable to set serial port"
EXCEPT_UNSUP_ON_PORT	"Not supported for this serial port"

EXCEPT_NOT_COMM	"Current message is not communications related"
EXCEPT_SERVER_COMM	"Cannot transmit on server comm resource"
EXCEPT_TCPSETUPFAIL	"Cannot open the requested TCP connection"
EXCEPT_NO_UDP_SUPPORT	"UDP protocol does not support this feature"
EXCEPT_INVAL_COM	"Invalid comm resource"
EXCEPT_BADPORT	"Bad port number"
EXCEPT_TCPRESET	"TCP connection reset"
EXCEPT_TCPREFUSED	"TCP connection refused"
EXCEPT_TCPTOOBIG	"TCP transmission buffer overflow"
EXCEPT_TCPTIMEOUT	"TCP transmission timed out"

B.10 Math Exceptions

EXCEPT_BADARCMATH	"Illegal arc function value"
EXCEPT_BADLOG	"Illegal log function value"
EXCEPT_BADSQRT	"Illegal sqrt function value"
EXCEPT_DIVBYZERO	"Division by zero"
EXCEPT_NOBYTES	"No byte representation for given type"
EXCEPT_BADBYTES	"Bad byte representation for given type"
EXCEPT_DOMAINERROR	"Math function domain error"

B.11 Flash Write Exceptions

EXCEPT_PBLK_TOO_BIG	"Block is too large"
EXCEPT_FLASHMEMORIZE	"Unable to place necessary routine in RAM"
EXCEPT_FLASHERASE	"Error occurred erasing block"
EXCEPT_FLASHWRITE	"Error occurred writing to flash"
EXCEPT_FLASHBADVERIFY	"Verification of written data to flash failed"
EXCEPT_FLASHBADREAD	"Unable to read flash"

B.12 File System Exceptions

EXCEPT_FFSNOSPACE	"Not enough space left in file system"
EXCEPT_FFSNOEXIST	"File does not exist"
EXCEPT_FFSNODELROOT	"Unable to delete root directory"
EXCEPT_FFSBADFD	"File resource number is invalid"
EXCEPT_FFSENDOFFILE	"End of file reached without sufficient data"
EXCEPT_FFSALREADYOPEN	"File is already open"
EXCEPT_FFSTOOMANYOPEN	"Too many open files"
EXCEPT_FFSBADOPEN	"Unable to open file as indicated"
EXCEPT_FFSDUPNAME	"File with given name already exists"
EXCEPT_FFSBADOP	"File not set up to allow requested operation"
EXCEPT_FFSPATHNOTFOUND	"Path not found"
EXCEPT_FFSBADNAME	"Illegal name specified"
EXCEPT_FFSTOOLONG	"Path or name too long"
EXCEPT_FFSDELOPEN	"Unable to delete open file"
EXCEPT_FFSNOINIT	"File system not inited (perhaps low on space)"

EXCEPT_FFSBADDIR	"Indicated file not directory"
EXCEPT_FFSBADOFFSET	"Invalid offset"
EXCEPT_FFSNOOVERWRITE	"No overwrite support for given mode"
EXCEPT_NOREAD	"Unable to read item of given type"
EXCEPT_NOWRITE	"Unable to write item of given type"
EXCEPT_FFSCORRUPT	"Corrupt file system detected"
EXCEPT_FFSBADMOVE	"Illegal rename requested"

B.13 Compiler Error Exceptions

EXCEPT_MISALIGNED	"Data misalignment problem"
EXCEPT_PARMERR	"Parameter error"
EXCEPT_BADFONTNUM	"Non-existent font reference"
EXCEPT_BADBMPNUM	"Non-existent bitmap reference"
EXCEPT_TYPEMISMATCH	"Incorrect types"
EXCEPT_BADOPCODE	"Illegal instruction opcode"
EXCEPT_BADDEREFTYPE	"Unable to dereference item"
EXCEPT_BADMATHOP	"Invalid operation for given type"
EXCEPT_BADPROMTYPE	"Unable to promote to desired type"
EXCEPT_STACKPOPPED	"Interpreter stack decimated"
EXCEPT_BLOWNSTACK	"Interpreter stack blown"
EXCEPT_STACKNOTEMPTY	"Returning with interpreter stack not empty"
EXCEPT_XCPTSTACKPOPPED	"Exception stack decimated"
EXCEPT_XCPTSTACKMORE	"Returning with exception stack not empty"
EXCEPT_BADAPI	"Attempting to call a non-existent API function"
EXCEPT_BADSNDNUM	"Non-existent sound reference"
EXCEPT_INVALRES	"Invalid resource"
EXCEPT_BADBASEFUNC	"Invalid call to default function"

The following are fatal system exceptions. These exceptions typically cause the system to stop processing after the description string is transmitted to the primary serial port.

B.14 Fatal Memory Exceptions

FERR_NOMEM	"Unable to allocate memory"
FERR_NO_STACK_MEM	"Unable to allocate memory on stack"
FERR_HEAPPTR	"Invalid heap pointer found"
FERR_HEAPCORRUPT	"Heap corruption detected"

B.15 Fatal Flash Exceptions

FERR_FLASHWRITE	"Error writing to flash"
FERR_FLASH	"Flash programming error"
FERR_NODEFRAG	"Missing required defragmentation area"
FERR_FFSCORRUPT	"Corrupted flash file system detected"
FERR_BADFFSSTART	"Current flash file system start is invalid"
FERR_BADSECNUM	"Bad sector number"
FERR_BADFFSWRITE	"Illegal write attempted"

FERR_FFSCANTSETUP	"Unable to setup flash file system"
FERR_FLASHREAD	"Error reading from flash"
FERR_FLASHERASE	"Error erasing flash"

B.16 Fatal Initialization Exceptions

FERR_GONEHWAREFIG	"Missing hardware configuration - contact QSI"
FERR_HAVE_PREV_BFF	"Previous BFF not freed"
FERR_NO_PREV_BFF	"No BFF to be freed"
FERR_BAD_BFF	"BFF bad or missing"
FERR_BAD_OBJVERSION	"Incompatible instance and template versions"
FERR_TTFENGINE_DEAD	"TTF engine failed to start"
FERR_TTFENGINE_NODIE	"TTF engine failed to terminate"
FERR_TASKSTART	"Unable to start required task"
FERR_MSGSYSSTART	"Unable to start messaging system"
FERR_FFSSTART	"Unable to start flash file system"
FERR_FONTERR	"Unable to use built in font"
FERR_DECOMPRESS	"Decompression error"
FERR_NATIVEOBJECT	"Native objects not suported"
FERR_BADINIT	"Initialization error"

B.17 Fatal Message System Exceptions

FERR_BADINITPOST	"Unable to post MSG_INIT"
FERR_BADSEMCOUNT	"Unexpected message queue error"
FERR_BADMESSAGE	"Unknown message in message queue"
FERR_MSGQSTALLED	"Message queue has stalled"

B.18 Network Fatal Exceptions

EXCEPT_OUTOFBUF	"Out of network packet buffers"

B.19 Miscellaneous Fatal Exceptions

FERR_MAXEXCEPT	"Too many exceptions"

B.20 Fatal Qlarity Foundry Exceptions

FERR_POSTWINMSG	"Unable to post Windows message"
FERR_WINSYNC	"Unable to synchronize to design tool"
FERR_NOFINDOBJ	"Unable to find object given by design tool"
FERR_BADGUI	"Illegal design tool request"
FERR_SOFTTERM	"Executing soft termination request"
FERR_APIRESET	"Application requested soft termination"
FERR_FAILEDREINIT	"Failed to re-init object"

APPENDIX C

QLARITY COMMAND LINE COMPILER

You can use the Qlarity compiler to compile your BASIC file rather than using Qlarity Foundry to compile it. This is useful if you created your user application in a text editor.

The compiler looks for a file called *natives.lib.* This file contains information message prototypes and functions for the API. This file must be in the same directory as the compiler or the program cannot be compiled.

NOTE:
The command line compiler is available as a Win32 or Linux executable file. The Win32 compiler is distributed with Qlarity Foundry. Contact QSI to obtain the Linux compiler. Please be aware that the Linux compiler is often in flux and may only be available as a snapshot of current Qlarity developement.

To compile a Qlarity BASIC file, at the command prompt, type the following:

```
qlarify <file_name>
```

To specify the name of the output file type, add the following parameters:

```
qlarify -o <output_file_name> <file_name>
```

If any errors occur, they are output to the display. To output the compiler errors to a file, type the following:

```
qlarify -e <error_file_name> <file_name>
```

To display Help on the command line syntax, type the following:

```
qlarify -h
```

A typical call to the compiler to compile an application in Qlarity Foundry might look like this:

```
qlarify -t keypaddef! charstr -t ->
   aggregate! string -s 19 -s 25 -a -o ->
   <output_file_name> <file_name>
```

This would have the compiler define the pseudotypes keypaddef% as a charstr and aggregate% as string. It would suppress warnings #19 and #25 (dangerous conversion and font definition warning), remove font encodings for Unicode characters from BDF fonts, and output the resultant .bff file to <output_file_name>.

Qlarify supports the following command line switches:

`-e <error_file>`	Redirect error output to <error_file>. If this option is not specified all error information is sent to the display
`-o <output_file>`	Generate the <output_file> as the compiled Qlarity application. If this is not specified, then output will be sent to a file with the same name as the input file where the .qly extension (if present) is replaced with .bff.
`-c`	Case sensitive. This makes the compiler case sensitive. By default the compiler is not case sensitive. Do not use this option with QSI provided code or libraries which assume case insensitivity. This is not recommended and should only be used if you have authored your entire application (including libraries and object templates) from scratch.
`-h, -?`	Print out help information
`-v`	Print out the compiler version only. Do not compile anything

`-a`	Remove character encodings greater than 255 (i.e. Unicode characters) from BDF fonts. This can save a lot of memory in applications that do not use Unicode. This is the default setting for applications compiled with Qlarity Foundry. Normally, if you use this option you will not specify -u
`-u`	Unicode support. Specifying this option will cause the char data type to be 16 bits wide (like a unibyte) and characters in a charstr to be 16 bits each. This effectively enables Unicode support. Normally you do not use this option in conjunction with -a
`-t <new_data_type> <existing_data_type>`	Create a new pseudo data type name <new_data_type> with the same attributes as <existing_data_type>. The KeypadDef% and Aggregate% data types often used in applications developed in Qlarity Foundry are examples of pseudo data types. Since specifying the percent(%) character in a command line can be difficult, you may substitute the bang (!) character wherever you need to use a percent. For applications developed in Qlarity Foundry, you should usually specify `-t keypaddef! charstr -t aggregate! string`
`-s <warning_number>`	Suppress warnings with the number <warning_number>. Some warnings may be overly repetitive and difficult to avoid. You may wish to suppress those warnings from the output. For applications developed in Qlarity Foundry, you should usually specify `-s 19 -s 25` which will suppress superfluous conversion warnings and BDF font warnings
`-p <option>`	Defines an option as if you had used #option <option> at the beginning of the source file.

APPENDIX D

QLARITY API FUNCTIONS QUICK REFERENCE LIST

This appendix provides a list of the Qlarity API functions in alphabetical order for quick reference. The syntax for each function's code is also listed. For more information and a description of each function, refer to the section and page listed in the "Manual Reference" column.

Each API function statement is defined as a single line of code. In the following list, when a statement wraps to the next line, -> appears at the end of the line to indicate that the statement is continued.

Function	Syntax	Reference
Acos	`acos(x as float) returns float`	section 4.11.5, page 73
AllocateArrayHandle	`allocatearrayhandle(data[] as reference? to -> anytype) returns ArrayHandle`	section 4.9.12, page 70
ArrayOperation	`arrayoperation (arr1[] as reference to anytype, -> arr2[] as reference? to anytype, op as ArrayOp)`	section 4.9.9, page 69
Asin	`asin(x as float) returns float`	section 4.11.4, page 73
Atan	`atan(x as float) returns float`	section 4.11.6, page 73
Attach	`attach(obj as objref, parent as objref)`	section 4.4.1, page 43
CalculateCRC	`calculatecrc(table[] as reference? to integer, -> width as integer, reflect as boolean, crcin as -> integer, reflectout as boolean, xoronout as -> integer, data[] as reference? to byte) returns -> integer`	section 4.18.27, page 95
ChangeCurDir	`changecurdir(name[] as reference? to byte)`	section 4.16.3, page 78
ChangePort	`changeport(channel as comm, newport as unibyte)`	section 4.1.9, page 38
CloseFile	`closefile(fnum as filedesc)`	section 4.16.9, page 80
Concat	`concat(strA[] as reference? to sametype!, -> strB[] as reference? to sametype!) returns sametype!`	section 4.9.7, page 69
Cos	`cos(x as float) returns float`	section 4.11.2, page 73
CreateCRCTable	`createcrctable(width as integer, poly as integer, -> reflect as boolean) returns integer[]`	section 4.18.26, page 95
DelayMS	`delayms(delay as integer)`	section 4.18.23, page 94
DrawBdfText	`drawbdftext(x as integer, y as integer, width as -> integer, height as integer, xoffset as integer, -> yoffset as integer, font as bdffont, data[] as -> reference? to anytype, {flags} as font_flags)`	section 4.7.4, page 57

Function	Syntax	Reference
DrawBDFTextFit	`drawbdftextfit(counts[] as reference? to integer, -> multiflags[] as reference? to multiline_flags, -> xpos[] as reference? to integer, ypos[] as -> reference? to integer, widths[] as reference? to -> integer, heights[] as reference? to integer, -> xoffsets[] as reference? to integer, yoffsets[] -> as reference? to integer, font as bdffont, data[] -> as reference? to anytype, start as integer, -> length as integer, flags as font_flags)`	section 4.7.5, page 57
DrawBitmap	`drawbitmap(x as integer, y as integer, bmp as -> bitmap)`	section 4.6.8, page 46
DrawBitmapRegion	`drawbitmapregion(x as integer, y as integer, -> xoffset as integer, yoffset as integer, width as -> integer, height as integer, bmp as bitmap)`	section 4.6.9, page 46
DrawBorder	`drawborder (x1 as integer, y1 as integer, x2 as -> integer, y2 as integer, style as integer, -> drawFlags as unibyte)`	section 4.6.21, page 52
DrawBox	`drawbox(left as integer, top as integer, righ} as -> integer, bottom as integer)`	section 4.6.14, page 48
DrawEllipse	`drawellipse(xoffset as integer, yoffset as -> integer, a as float, xfocal as float, yfocal as -> float, theta as float, gamma as float, flags as -> ellipse_flags)`	section 4.6.16, page 49
DrawLine	`drawline(x1 as integer, y1 as integer, x2 as -> integer, y2 as integer)`	section 4.6.7, page 46
DrawPixmap	`drawpixmap(x as integer, y as integer, pixmap[] -> as reference? to byte, mapwidth as integer, -> mapheight as integer)`	section 4.6.11, page 47
DrawPixmapRegion	`drawpixmapregion(x as integer, y as integer, -> xoffset as integer, yoffset as integer, width as -> integer, height as integer, pixmap[] as -> reference? to byte, mapwidth as integer, -> mapheight as integer)`	section 4.6.12, page 47
DrawPolygon	`drawpolygon(xpoints[] as integer, ypoints[] as -> integer, flags as poly_flags)`	section 4.6.15, page 48
DrawSysText	`drawsystext(x as integer, y as integer, width as -> integer, height as integer, xoffset as integer, -> yoffset as integer, font as sysfont, facenum as -> integer, ptsize as integer, data[] as reference? -> to anytype, flags as font_flags)`	section 4.7.14, page 63

Function	Syntax	Reference
DrawSysTextFit	`drawsystextfit(multiflags[] as reference? to -> multiline_flags, xpos[] as reference? to integer, -> ypos[] as reference? to integer, widths[] as -> reference? to integer, heights[] as reference? to -> integer, xoffsets[] as reference? to integer, -> yoffsets[] as reference? to integer, indices[] as -> reference? to integer, lengths[] as reference? to -> integer, font as sysfont, facenum as integer, -> ptsize as integer, data[] as reference? to -> anytype, flags as font_flags)`	section 4.7.15, page 64
DrawTTFText	`drawttftext(x as integer, y as integer, width as -> integer, height as integer, xoffset as integer, -> yoffset as integer, font as ttfont, facenum as -> integer, ptsize as integer, data[] as reference? -> to anytype, {flags} as font_flags)`	section 4.7.8, page 60
Enable	`enable(obj as objref, flag as boolean)`	section 4.3.4, page 41
EnableKeypadBacklight	`enablekeypadbacklight(enable as boolean)`	section 4.18.8, page 85
EndOfFile	`endoffile(fnum as filedesc) returns boolean`	section 4.16.14, page 81
EraseFile	`erasefile(name[] as reference? to byte)`	section 4.16.6, page 79
EraseFileSpace	`erasefilespace()`	section 4.16.15, page 81
Exp	`exp(x as float) returns float`	section 4.11.8, page 74
FakeKeyMsg	`fakekeymsg(msgtype as fake_key, keycode as UNIBYTE)`	section 4.12.4, page 75
FakeScreenMsg	`fakescreenmsg(msgtype as fake_screen, x1 as -> integer, y1 as integer, x2 as integer, y2 as -> integer)`	section 4.12.5, page 75
Find	`find(match[] as reference? to sametype!, start as -> integer, length as integer, pattern[] as -> reference? to sametype!) returns integer`	section 4.9.6, page 69
FreeArrayHandle	`freearrayhandle (handle as ArrayHandle)`	section 4.9.10, page 70
FromBytes	`frombytes(var as reference to anytype, toset[] as -> byte, bigendian as boolean)`	section 4.10.3, page 72
GetAvailFilespace	`getavailfilespace() returns integer`	section 4.16.1, page 78
GetBdfFontMetrics	`getbdffontmetrics(maxleft as reference to integer,-> maxright as reference to integer, maxup as -> reference to integer, maxdown as reference to -> integer, xnextline as reference to integer, -> ynextline as reference to integer, font as -> bdffont, flags as font_flags)`	section 4.7.3, page 56

Function	Syntax	Reference
GetBDFTextFit	`getbdftextfit(counts[] as reference to integer, -> multiflags[] as reference to multiline_flags, -> xpos[] as reference to integer, ypos[] as -> reference to integer, widths[] as reference to -> integer, heights[] as reference to integer, -> xoffsets[] as reference to integer, yoffsets[] as -> reference to integer, font as bdffont, data[] as -> reference? to sametype!, start as integer, length -> as integer, wordbrks[] as reference? to sametype!,-> linebrks[] as reference? to sametype!, flags as -> font_flags)`	section 4.7.2, page 54
GetBdfTextSize	`getbdftextsize(width as reference to integer, -> height as reference to integer, xoffset as -> reference to integer, yoffset as reference to -> integer, font as bdffont, data[] as -> reference? to anytype, flags as font_flags)`	section 4.7.1, page 53
GetBinaryResource	`getbinaryresource (resourceID as integer)returns -> byte[]`	section 4.18.24, page 94
GetBitmapSize	`getbitmapsize(width as reference to integer, -> height as reference to integer, bmp as bitmap)`	section 4.6.13, page 48
GetBytes	`getbytes(tobreak as anytype, bigendian as -> boolean) returns byte[]`	section 4.10.4, page 72
GetCapture	`getcapture(obj as objref) returns integer`	section 4.13.2, page 76
GetChildren	`getchildren(contobj as objref) returns objref[]`	section 4.3.6, page 42
GetComMessageSource	`getcommessagesource()returns comm`	section 4.1.4, page 36
GetContainer	`getcontainer(obj as objref) returns objref`	section 4.3.5, page 42
GetCurDir	`getcurdir() returns string`	section 4.16.4, page 79
GetDirEntry	`getdirentry(index as integer) returns string`	section 4.16.5, page 79
GetEllipseSize	`getellipsesize(maxleft as reference to integer, -> maxright as reference to integer, maxup as -> reference to integer, maxdown as reference to -> integer, a as float, xfocal as float, {yfocal} -> as float, theta as float, gamma as float, flags -> as ellipse_flags)`	section 4.6.17, page 50
GetEnableInfo	`getenableinfo(obj as objref, eval as enable_info) -> returns boolean`	section 4.3.7, page 42
GetException	`getexception(msg[] as reference to byte, errtype -> as reference to unibyte, errlevel as reference to -> unibyte) returns boolean`	section 4.14.2, page 77
GetFileInfo	`getfileinfo(name[] as reference? to byte, size as -> reference to integer, usedfilespace as reference -> to integer, {flags} as reference to fileinfo_flags)`	section 4.16.7, page 79
GetFilePos	`getfilepos(fnum as filedesc) returns integer`	section 4.16.13, page 81
GetHardwareInfo	`gethardwareinfo(req} as hwinfo returns byte[]`	section 4.18.5, page 83

Function	Syntax	Reference
GetNetChannelInfo	`getnetchannelinfo(channel as comm, prot as -> reference to netprotocol, 1port as reference to -> unibyte, fport as reference to unibyte, ipaddr[] -> as reference to byte)`	section 4.1.11, page 38
GetObjPixmap	`getobjpixmap(width as reference to integer, height-> as reference to integer) returns color[]`	section 4.6.10, page 47
GetObjPixmapRegion	`getobjpixmapregion(x as integer, y as integer, -> width as integer, height as integer)`	section 4.6.22, page 52
GetObjProp	`getobjprop(obj as objref, name[] as reference? to -> byte) returns string`	section 4.3.2, page 41
GetObjref	`getobjref(name[] as reference? to byte) returns -> objref`	section 4.3.1, page 41
GetPosInfo	`getposinfo(obj as objref, pval as position_info)-> returns integer`	section 4.3.8, page 42
GetProfileTick	`getprofiletick() returns integer`	section 4.18.22, page 94
GetRandomNum	`getrandomnum() returns float`	section 4.18.17, page 93
GetScreenPixmap	`getscreenpixmap(x as integer, y as integer, width -> as integer, height as integer) returns color[]`	section 4.6.18, page 51
GetSysFontCharacters	`getsysfontcharacters(font as sysfont,facenum as -> integer, bRange as unibyte, eRange as unibyte) -> returns unibyte[]`	section 4.7.10, page 61
GetSysFontMetrics	`getsysfontmetrics(maxleft as reference to integer,-> maxright as reference to integer,maxup as -> reference to integer, maxdown as reference to -> integer, xnextline as reference to integer, -> ynextline as reference to integer, font as -> sysfont, facenum as integer, ptsize as integer, -> flags as font_flags)`	section 4.7.13, page 63
GetSystemSetting	`getsystemsetting(cmd as syscmd, var as reference -> to anytype, which as syscmd_readfrom)`	section 4.18.15, page 92
GetSysTextFit	`getsystextfit(multiflags[] as reference to -> multiline_flags, xpos[] as reference to integer, -> ypos[] as reference to integer, widths[] as -> reference to integer, heights[] as reference to -> integer, xoffsets[] as reference to integer, -> yoffsets[] as reference to integer, indices[] as -> reference to integer, lengths[] as reference to -> integer, font as sysfont, facenum as integer, -> ptsize as integer, data[] as reference? to -> sametype!, wordbrks[] as reference? to sametype!, -> linebrks[] as reference? to sametype!, flags as -> font_flags)`	section 4.7.12, page 62

Function	Syntax	Reference
GetSysTextSize	`getsystextsize(width as reference to integer, -> height as reference to integer, xoffset as -> reference to integer, yoffset as reference to -> integer, font as sysfont, facenum as integer, -> ptsize as integer, data[] as reference? to -> anytype, flags as font_flags)`	section 4.7.11, page 61
GetTemperature	`gettemperature() returns integer`	section 4.18.12, page 86
GetTime	`gettime(day as reference to integer, month as -> reference to integer, dig2year as reference to -> integer, dotw as reference to weekday, hour as -> reference to integer, minute as reference to -> integer, second as reference to integer)`	section 4.18.10, page 85
GetTTFFontMetrics	`getttfontmetrics(maxleft as reference to integer, -> maxright as reference to integer, maxup as -> reference to integer, maxdown as reference to -> integer, xnextline as reference to integer, -> ynextline as reference to integer, font as -> ttfont, facenum as integer, ptsize as integer, -> flags as font_flags)`	section 4.7.7, page 59
GetTTFTextSize	`getttftextsize(width as reference to integer, -> height as reference to integer, xoffset as -> reference to integer, yoffset as reference to -> integer, font as ttfont, facenum as integer, -> ptsize as integer, data[] as reference? to -> anytype, flags as font_flags)`	section 4.7.6, page 59
GetVersion	`getversion() returns float`	section 4.18.4, page 83
IgnoreDrawCache	`ignoredrawcache(obj as objref, ignore as boolean)`	section 4.6.20, page 51
Left	`left(arr[] as reference? to sametype!, len as -> integer) returns sametype!`	section 4.9.2, page 68
Len	`len(arr[] as reference? to anytype) returns integer`	section 4.9.1, page 68
Ln	`ln(x as float) returns float`	section 4.11.9, page 74
Lower	`lower(obj as objref)`	section 4.4.5, page 44
LowerCase	`lowercase(str[] as byte) returns string`	section 4.10.5, page 72
MakeDir	`makedir(name[] as reference? to byte)`	section 4.16.2, page 78
Mid	`mid(arr[] as reference? to sametype!, index as -> integer, len as integer) returns sametype!`	section 4.9.4, page 68
NetClose	`netclose(channel as comm)`	section 4.1.7, page 37
NetSendDatagram	`netsenddatagram(localport as unibyte, foreignport -> as unibyte, ipaddr[]as reference? to byte, data[] -> as reference? to byte)`	section 4.1.19, page 40
NetServerClose	`netserverclose(channel as servercomm)`	section 4.1.8, page 37

Function	Syntax	Reference
NetOpen	`netopen(obj as objref, prot as netprotocol, -> localport as unibyte, foreignport as unibyte, -> ipaddr[] as reference? to byte)`	section 4.1.5, page 36
NetServerOpen	`netserveropen(obj as objref, prot as netprotocol, -> localport as unibyte) returns servercomm`	section 4.1.6, page 37
OpenFile	`openfile(name[] as reference? to byte, flags as -> file_flags) returns filedesc`	section 4.16.8, page 80
PlayNote	`playnote(note as byte, duration as integer)`	section 4.8.1, page 65
PlayNoteNotify	`playnotenotify (obj as objref, parm as integer, -> note as byte, duration as integer)`	section 4.8.2, page 67
PlaySound	`playsound (sound as audio)`	section 4.8.3, page 67
PlaySoundNotify	`playsoundnotify (obj as objref, parm as integer, -> sound as _audio)`	section 4.8.4, page 67
Power	`power(x as float, exp as float) returns float`	section 4.11.7, page 73
Raise	`raise(obj as objref)`	section 4.4.4, page 44
ReadArrayHandle	`readarrayhandle (data[] as reference to anytype, -> handle as ArrayHandle)`	section 4.9.11, page 70
Read DCD	`readdcd(resource as comm) returns boolean`	section 4.1.18, page 40
ReadDTR	`readdtr(resource as comm) returns boolean`	section 4.1.17, page 39
ReadFile	`readfile(fnum as filedesc, var as reference to -> anytype)`	section 4.16.10, page 80
ReadGPIO	`readgpio(pins as unibyte) returns GPIO_PIN`	section 4.18.2, page 82
ReadRTS	`readrts(resource as comm) returns boolean`	section 4.1.15, page 39
ReadUserConfig	`readuserconfig(len as integer) returns byte[]`	section 4.15.1, page 78
Redim	`redim(arr[] as reference to anytype, newsize as -> integer)`	section 4.9.8, page 69
RegisterKey	`registerkey(obj as objref, keycode as unibyte)`	section 4.2.3, page 41
RegisterMsgHandler	`registermsghandler(obj as objref, msgnum as -> message, {msgparm} as unibyte)`	section 4.2.1, page 40
Relocate	`relocate(obj as objref, x as integer, y as integer)`	section 4.5.3, page 44
RemoveCapture	`removecapture(obj as objref)`	section 4.13.3, page 76
RenameFile	`renamefile(name[] as reference? to byte, newname[]-> as reference? to byte)`	section 4.16.16, page 81
Replace	`replace(match[] as reference to sametype!,start -> as integer, len as integer, pattern[] as -> reference? to sametype!, newdata[] as reference? -> to sametype!, count as integer)returns integer`	section 4.9.14, page 71
Rerender	`rerender(obj} as objref)`	section 4.5.1, page 44

Function	Syntax	Reference
Resize	`resize(obj} as objref, width as integer, height -> as integer)`	section 4.5.2, page 44
Rethrow	`rethrow()`	section 4.14.3, page 77
ReverseFind	`reversefind(match[] as reference? to sametype!, -> start as integer, len as integer, pattern[] as -> reference? to sametype!) returns integer`	section 4.9.13, page 70
RGB	`rgb(red as byte, green as byte, blue as byte) -> returns color`	section 4.6.5, page 45
Right	`right(arr[] as reference? to sametype!, len as -> integer) returns sametype!`	section 4.9.3, page 68
SeedRandomNum	`seedrandomnum()`	section 4.18.18, page 93
Send	`send(resource as comm,data[] as anytype)`	section 4.1.1, page 35
SendtoBack	`sendtoback(obj as objref)`	section 4.4.3, page 43
SendtoFront	`sendtofront(obj as objref)`	section 4.4.2, page 43
SetArrayData	`setarraydata(arr[] reference? to sametype!, index -> as integer, srcdata[] as reference? to sametype!, -> srcindex as integer, len as integer)`	section 4.18.25, page 94
SetBacklight	`setbacklight(command as backlight_adjust)`	section 4.18.7, page 85
SetBgColor	`setbgcolor(newcolor as byte)`	section 4.6.4, page 45
SetBreak	`setbreak(resource as comm, {state} as boolean)`	section 4.1.3, page 35
SetCapture	`setcapture(obj as objref)`	section 4.13.1, page 76
SetContrast	`setcontrast(direction as contrast_adjust)`	section 4.18.6, page 84
SetCTS	`setcts(resource as comm, {outValue} as boolean`	section 4.1.14, page 39
SetDSR	`setdsr(resource as comm, outValue as boolean)`	section 4.1.16, page 39
SetFgColor	`setfgcolor(newcolor as byte)`	section 4.6.3, page 45
SetFilePos	`setfilepos(fnum as filedesc, offset as integer, -> absolute as boolean)`	section 4.16.12, page 81
SetGPIO	`setgpio(pins as unibyte, action as gpio_action)`	section 4.18.1, page 82
SetGPIODirection	`setgpiodirection(pins as unibyte, input as boolean)`	section 4.18.3, page 83
SetLED	`setled(cmd as ledcmd, lednum as integer)`	section 4.18.9, page 85
SetObjProp	`setobjprop(obj as objref, name[] as reference? to -> byte, value[] as reference? to byte)`	section 4.3.3, page 41
SetOrigin	`setorigin(cont as objref, {originX} as integer, -> originY as integer)`	section 4.3.9, page 43
SetPalette	`setpalette(red[] as reference? to byte, green[] -> as reference? to byte, blue[] as reference? to byte)`	section 4.18.30, page 96
SetPixel	`setpixel(x as integer, y as integer)`	section 4.6.6, page 46
SetSeedRandomNum	`setseedrandomnum(seed as integer)`	section 4.18.19, page 93

Function	Syntax	Reference
SetSerialRecvSize	`setserialrecvsize(res as comm, newsize as integer)`	section 4.1.12, page 38
SetSerialTimeout	`setserialtimeout(res as comm, newtimeout as integer)`	section 4.1.13, page 39
SetSystemSetting	`setsystemsetting(cmd as syscmd, newvalue as -> anytype, action as syscmd_action)`	section 4.18.14, page 87
SetTime	`settime(day as integer, month as integer, -> dig2year as integer, dotw as weekday, hour as -> integer, minute as integer, second as integer)`	section 4.18.11, page 86
SetTransparent	`settransparent(newcolor as byte)`	section 4.6.1, page 45
SetTTFAngle	`setttfangle(theta as float)`	section 4.7.9, page 60
SetVolume	`setvolume(direction as volume_adjust)`	section 4.8.6, page 67
Sin	`sin(x as float) returns float`	section 4.11.1, page 73
SoftReset	`softreset(rst as rstmode)`	section 4.18.16, page 93
Sqrt	`sqrt(x as float) returns float`	section 4.11.10, page 74
StopSpkr	`stopspkr()`	section 4.8.5, page 67
Str	`str(value as reference? to anytype) returns string`	section 4.10.1, page 71
Tan	`tan(x as float) returns float`	section 4.11.3, page 73
Throw	`throw(loc[] as reference? to byte, msg[] as -> reference? to byte)`	section 4.14.1, page 77
Tool_Persist	`tool_persist(x as reference to anytype)`	section 4.17.1, page 82
Tool_Trace	`tool_trace(str as string)`	section 4.17.2, page 82
Transmit	`transmit(resource as comm, data[] as reference? -> to anytype, block as boolean)`	section 4.1.2, page 35
TransmitUrgent	`transmiturgent(channel as comm, data[] as -> reference? to anytype)`	section 4.1.10, page 38
Trim	`trim(arr[] as reference? to byte) returns string`	section 4.9.5, page 68
TypeOf	`typeof(unique as reference to integer, obj as -> objref, name[] as reference? to byte) returns -> typeval`	section 4.18.13, page 86
UnregisterMsgHandler	`unregistermsghandler(obj as objref, msgnum as -> message, msgparm as unibyte)`	section 4.2.2, page 40
UpperCase	`uppercase(str[] as byte) returns string`	section 4.10.6, page 73
UseDrawCache	`usedrawcache(useit as boolean)`	section 4.6.19, page 51
UseTransparent	`usetransparent(flag as boolean)`	section 4.6.2, page 45
UserBroadcastMsg	`userbroadcastmsg(startobj as objref, msgnum as -> message, parm as integer, donow as boolean)`	section 4.12.1, page 74
UserSendMsg	`usersendmsg(startobj as objref, msgnum as message,-> parm as integer, donow as boolean)`	section 4.12.2, page 74

Function	Syntax	Reference
UserDirectMsg	`userdirectmsg(startobj as objref, msgnum as -> message, parm as integer, donow as boolean) -> returns boolean`	section 4.12.3, page 75
Val	`val(value as reference to anytype, text[] as -> reference? to byte)`	section 4.10.2, page 71
WatchdogEnable	`watchdogenable(enable as boolean, timeout as -> integer)`	section 4.18.20, page 93
WatchdogReset	`watchdogreset()`	section 4.18.21, page 94
WriteFile	`writefile(fnum} as filedesc, data as reference? -> to anytype)`	section 4.16.11, page 81
WriteUserConfig	`writeuserconfig(cfg[] as reference? to byte)`	section 4.15.2, page 78
ZlibCompress	`zlibcompress(out[] as reference to byte, in[] as -> reference? to byte)`	section 4.18.28, page 96
ZlibDecompress	`zlibdecompress(out[] as reference to byte, in[] -> as reference? to byte)`	section 4.18.29, page 96