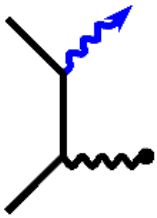
Handbuch für den Simplify Technologies GUI-Interpreter

Version 4.8.0



Simplify Technologies GmbH
Steinbühlstraße 15
35578 Wetzlar

$Rev: 6957 $

Simplify Technologies GmbH
Steinbühlstraße 15
35578 Wetzlar
Tel.: (+49) (0)6441-210390
FAX.: (+49) (0)6441-210399
Internet: www.simplify-technologies.de
Email: info@simplify-technologies.de

Inhaltsverzeichnis

1 Einleitung

1.1 Übersicht

Grafikfähige LCD und Touchdisplays werden in *embedded systems* zunehmend eingesetzt, um dem Benutzer eine ergonomische Bedienung zu ermöglichen. Simplify Technologies bietet Ihnen hierfür die Produkte, Teilkomponenten und fertigen Lösungen, die Sie für Ihr Projekt benötigen.
Die Simplify Technologies GUI–Bibliothek ist eine Sammlung von aufeinander abgestimmten Unterprogrammen und Funktionen für die Realisierung grafischer Benutzeroberflächen für Entwickler, die ihre eigene Bedieneinheit entwickeln. Sie erlaubt Ihnen, die Vorteile grafikfähiger Displays zu nutzen, ohne daß Sie die aufwendigen Ansteuerungsroutinen selber realisieren müssen. Dies bedeutet eine wesentliche Vereinfachung und Beschleunigung Ihrer Entwicklung.

Wenn Sie keine eigene Hardware entwickeln möchten, so bieten wir Ihnen unsere fertigen Module der LCM–Serie an. Diese können Sie über die serielle Schnittstelle ansteuern (je nach Ausführung RS–232, RS–422 oder RS–485), so daß eine Programmierung auf Seite des Display–Modules nicht erforderlich ist. Sie können Ihr gewohntes System verwenden und von dort Befehle an die LCM–Module senden, wo sie dann vom *GUI–Interpreter* ausgeführt werden. Die Befehle lehnen sich an die Funktionen der GUI–Bibliothek an, so daß eine Anwendung relativ einfach portiert werden kann, wenn Sie zu einem späteren Zeitpunkt eine “eigene” Hardware realisiern wollen.

Die vorliegende Dokumentation beschreibt den Simplify Technologies GUI-Interpreter und erläutert, wie Sie die mit dem GUI–Interpreter ausgestatteten LCM-Module verwenden können.

Nach einem schnellen Start und Test Ihres LCM in Abschnitt 1.2 werden in Kapitel 1.3 des Handbuches die grundlegenden Konzepte für die Ansteuerung der LCM–Module und des GUI–Interpreters vorgestellt. Kapitel 3 behandelt die Funktionsmodule des GUI–Interpreters und die Ihnen zur Verfügung gestellten Programmiermöglichkeiten im einzelnen. In Kapitel 2 wird das serielle Übertragungsprotokoll für die Befehle besprochen. Ein ausführliches Programmierbeispiel zur Verdeutlichung und als Ausgangsmodell für Ihre eigenen Anwendungen wird in Kapitel 5 beschrieben. Es ergänzt die Beschreibungen der vorangehenden Kapitel und zeigt wichtige Details der Programmierung der Software für das ansteuernde Gerät. Kapitel 7 gibt Hinweise über Sicherheitsfragen, die beim Einsatz der Geräte und des GUI–Interpreters zu beachten sind. In Anhang D finden Sie Antworten auf häufig auftretende Fragen.

Sollte dieses Handbuch Fragen offen lassen, werden wir uns bemühen, diese zügig und direkt zu beantworten. Wir wünschen Ihnen viel Spaß und Erfolg bei der Entwicklung Ihrer Anwendung!

1.2 Erster Start

Bevor Sie die leistungsfähigeren Eigenschaften und Kommunikationsmöglichkeiten mit dem LCM nutzen, schlagen wir vor, einen ersten Test des GUI-Interpreters im "ASCII-Modus" vorzunehmen. Gehen Sie dazu wie folgt vor:

1. Verbinden Sie zunächst das LCM57 mit der Stromversorgung und über einen geeigneten Adapter (je nach der von Ihnen bestellten Hardwareausstattung des LCMs reicht ein einfaches Kabel) mit einer seriellen Schnittstelle eines PCs.

2. Starten Sie auf dem PC ein beliebiges Terminal–Programm und stellen Sie dieses auf die serielle Schnittstelle ein, an der das LCM angeschlossen ist. Die Übertragungsparameter des Terminal–Programms müssen hier auf die gewünschte Baudrate (unterstützte Werte sind: 9600, 19200, 38400, 57600, 76800, 115200), 8 Datenbits, 1 Stopbit, keine Parität eingestellt werden. Das Terminal–Programm muß beim Drücken der Returntaste ein *Carrige Return*–Zeichen (CR = 0x0D) senden (im Folgenden durch <cr> bezeichnet).

3. Das LCM ermittelt nun die Baudrate der seriellen Übertragung (vgl. Abschnitt 2.1). Hierzu geben Sie zunächst das `@` Zeichen solange ein, bis das LCM mit dem Wort `OK` antwortet. Geben Sie nun als Bestätigung `zPING` <cr> ein.

4. Von nun an sollte das LCM weitere Befehle im ASCII–Modus verstehen. Jeder Befehl im ASCII–Modus beginnt mit einem kleinen `z` und endet mit einem CR–Zeichen. Die einzelnen Worte bzw. Parameter eines Befehls werden durch Leerzeichen voneinander getrennt.

   Versuchen Sie zunächst eine Linie vom Koordinatenursprung nach (50, 50) zu Zeichnen. Geben Sie ein: `zLINE 50 50` <cr>. Einen Kreis mit dem Radius 40 können Sie am Ende dieser Linie mit dem Befehl `zCIRCLE 40` <cr> erzeugen.

Weitere einfache Befehle im ASCII-Modus sind z. B.:
`zSET_CURSOR_POSITION 0 0` <cr>: setzt den (unsichtbaren) Zeichencursor auf (0, 0).
`zCOLORED_CIRCLE 75` <cr>: malt ausgefüllten Kreis mit Radius 75.
`zTEXT ''hallo''` <cr>: schreibt den Text "hallo".

`zRECTANGLE 100 40 <cr>`: zeichnet Rechteck der Breite 100 und der Höhe 40.
`zSET_LINEWIDTH 5 <cr>`: setzt die Liniendicke auf 5 (Standard ist 1).
`zINVERT_AREA 60 70 <cr>`: invertiert rechteckigen Bereich der Breite 60 und der Höhe 70
`zRESET <cr>`: setzt die Hardware zurück und startet den GUI–Interpreter neu.

Im weiteren Verlauf dieses Handbuches werden Sie weitere Grafikbefehle und effizientere Möglichkeiten kennenlernen, mit dem LCM zu kommunizieren.

1.3 Konzept der Programmierung mit dem GUI-Interpreter

Der GUI–Interpreter stellt, neben spezifischen zusätzlichen Befehlen, im wesentlichen dieselben Möglichkeiten zur Verfügung wie die Simplify Technologies GUI–Bibliothek. Der wesentliche Unterschied besteht darin, daß Benutzeroberfläche bzw. Anwendung bei Verwendung der GUI–Bibliothek direkt auf der Hardware des Display–Moduls in ANSI–C programmiert wird, während bei Verwendung des GUI–-Interpreters die Programmierung auf Seiten der ansteuernden Hardware erfolgt. Der GUI–Interpreter ist fest auf den Display–Modulen der LCM–Serie installiert und nimmt dann Befehle über die serielle Schnittstelle entgegen, so daß die Module für beliebige Anwendungen verwendet werden können, ohne daß sie selber direkt programmiert werden müßten. Für den Anwendungsprogrammierer ergibt sich so der Vorteil, daß die gesamte Programmierung auf dem ihm vertrauten System stattfindet. Abbildung 1 zeigt eine mögliche Systemkonfiguration.

Grundsätzlicher Ablauf: Das Display–Modul empfängt Datenpakete, interpretiert deren Inhalt und führt dann die gewünschten Befehle und Grafikfunktionen aus (Details hierzu können in Kapitel 2 auf Seite 8 nachgelesen werden). Erst nachdem das Display–Modul ein Antwortpaket gesendet hat ist es bereit weitere Befehle zu empfangen. Die Programmierung einer grafischen Benutzeroberfläche besteht also darin, die passenden Datenpakete zusammenzustellen und an das Modul zu senden. Diese Datenpakete werden zweckmäßigerweise von Funktionen zusammengestellt und versendet, die dann ihrerseits vom Anwendungsprogramm aufgerufen werden, so daß es nicht nötig ist, sich auf Anwendungsebene mit der Erzeugung der Datenpakete zu beschäftigen.

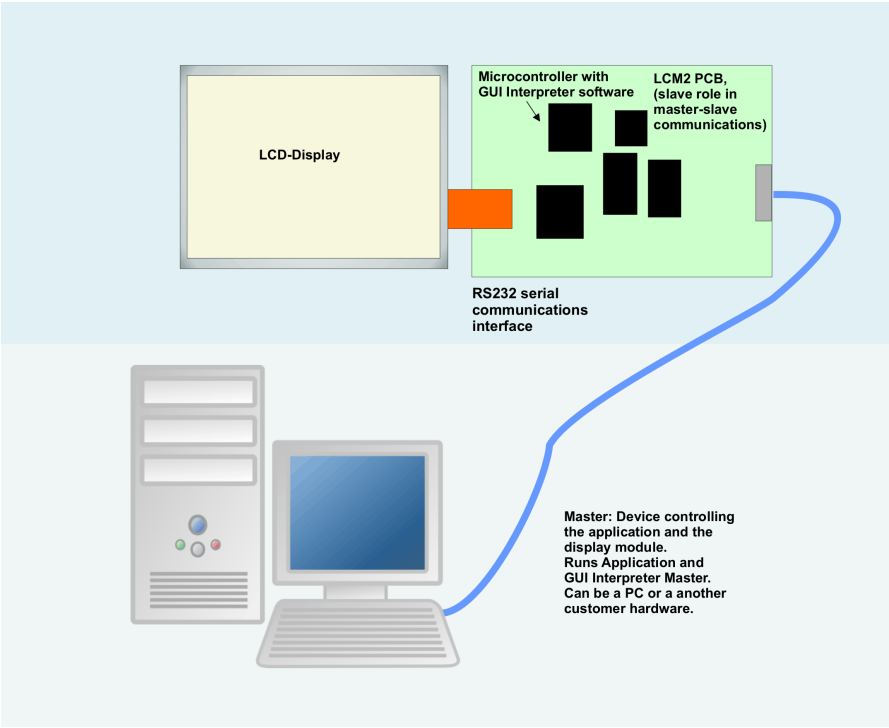


Abbildung 1: Mögliche Anwendung

2 Serielle Kommunikation

Die Kommunikation zwischen dem ansteuernden Gerät und dem LCM–Display–Modul erfolgt über das wechselseitige Zusenden von Datenpaketen. Dabei wird die Kommunikation grundsätzlich vom ansteuernden Gerät (im folgenden *Master* genannt) initiiert, das LCM–Modul sendet (als *Slave*) ausschließlich als Antwort auf die Anforderungen des Masters. In einer Vielzahl der Fälle wird der Master nur ein einzelnes LCM–Modul ansteuern. Dies ist der einfachste Fall. Eine Möglichkeit, mehrere LCMs anzuschließen, ist ebenfalls vorhanden, der Betrieb ist dann *Master / multi Slave*, wobei den einzelnen Slaves Adressen zugeordnet werden.
Die Kommunikationssequenz zwischen Master und Slave sieht grundsätzlich so aus, wie in Abbildung 2 dargestellt.

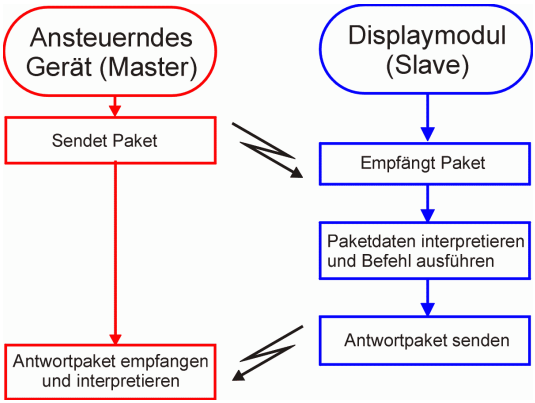


Abbildung 2: Grundsätzlicher Ablauf der Kommunikation

Neben einem leistungsfähigen binären Übertragungsprotokoll gibt es zusätzlich die Möglichkeit, das LCM-Display-Modul in einem *ASCII–Modus* anzusteuern (siehe hierzu Abschnitt 2.2 auf der anderen Seite).
Zuerst wird beschrieben, wie das LCM sich automatisch auf die serielle Übertragungsrate des Masters einstellt.

2.1 Autobauding

Das LCM kann seine serielle Übertragungsrate beim Systemstart an die des Masters anpassen. Etwa 500 ms nach dem Einschalten sind die LCMs bereit zum Autobauding. Hierbei werden folgende Baudraten unterstützt: 9600, 19200, 38400, 57600, 76800 und 115200. Das LCM berechnet die Übertragungsgeschwindigkeit aus der Zeit, die der Master zum Senden des “@”–Zeichens benötigt. Danach antwortet

es mit dem String “OK<cr>” , wenn eine der obigen Standardbaudraten ermittelt wurde. Andernfalls wird weiter auf “@”–Zeichen gewartet. Anschließend muß der Master nochmals mit einem “zPING<cr>” bestätigen. Falls das LCM dieses Kommando nicht mit einem “0<cr>” beantwortet, konnte es die letzte Zeichenkette nicht verstehen und erwartet einen neuen Durchlauf des Autobaudings, beginnend mit dem “@”–Zeichen. (Zur Verdeutlichung des Autobauding–Protokolls siehe auch Abbildung 3.)

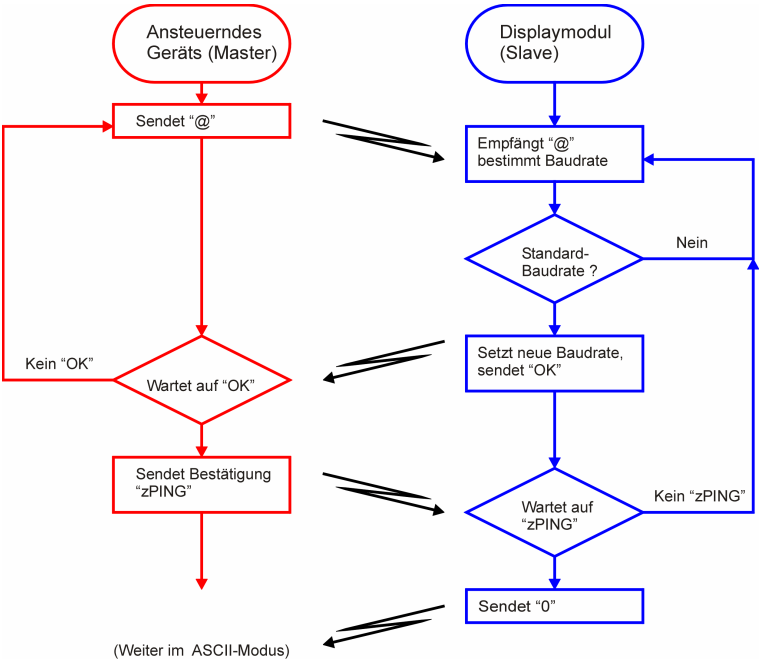


Abbildung 3: Ablauf des Autobaudings

2.2 Betrieb im ASCII–Modus

Der GUI-Interpreter interpretiert auch Befehle im ACII–Modus. Da der ASCII–Modus nicht über die Möglichkeiten zur Sicherstellung einer fehlerfreien Kommunikation verfügt wie der Binärmodus, ist er nicht für den Einsatz im fertigen Produkt vorgesehen, sondern ausschließlich für Testzwecke. Er unterstützt daher keine Funktionen zur Auswertung von berührungsempfindlichen Touch–Gläsern (optional), so dass auch die damit verbundenen Buttons und Keyboards nicht durch den ASCII–Modus kontrolliert werden können.

Im ASCII–Modus bestehen die Datenpakete aus einzelnen Zeichenketten, die sogar von einem einfachen Terminalprogramm auf einem PC an das LCM gesendet bzw. von diesem empfangen werden können. Jeder Befehl im ASCII–Modus beginnt mit einem kleinen “z” und endet mit einem CR–Zeichen. Die einzelnen Worte bzw. Parameter eines Befehls werden durch Leerzeichen voneinander getrennt. Das LCM antwortet auf Befehle ebenfalls mit einer Zeichenkette, an deren Anfang die Fehlernummer des ausgeführten Befehls steht. Trat kein Fehler auf (Fehlernummer $= 0$), so folgen je nach Kommando die verschiedenen Rückgabeparameter, die jeweils wieder durch Leerzeichen separiert sind. In der Funktionsreferenz werden die einzelnen Kommandos und ihre Anwendung erläutert. In Anhang A auf Seite 71 werden die Fehlercodes erläutert.

2.3 Bestandteile des binären Übertragungsprotokolls

An die Datenübertragung werden folgende Anforderungen gestellt:

1. Die Datenübertragung soll zuverlässig und effizient erfolgen. Es sollen Mechanismen zur Erkennung und Behebung von Fehlern vorhanden sein.
2. Es soll die Möglichkeit bestehen, mehrere Displays an den Master anzuschließen (z.B. über eine RS–485–Schnittstelle).
3. Es soll optional sowohl möglich sein, daß der Master ausführliche und aussagekräftige Antwortpakete auf seine Anforderungen erhält, als auch daß der Slave nur eine minimale Antwort zurücksendet, wenn z.B. der Master keine umfangreiche Interpretation der Antwort vornehmen kann.
4. Die Implementierung der für den Master erforderlichen Routinen soll möglichst einfach sein.

Um diese Anforderungen zu erfüllen, wird vorgesehen:

Zu 1.: Für eine effiziente Datenübertragung werden die Daten binär übertragen. Für eine sichere Datenübertragung wird jedes Paket mit einer Checksumme versehen, die eine Überprüfung des korrekten Empfangs ermöglicht. Diese Checksumme kann entweder eine einfache 16–Bit–Summe oder eine CRC16–Prüfsumme über alle sonstigen im Paket vorhandenen Bytes sein.

Zusätzlich kann eingestellt werden, ob jedes Paket mit einer *Sequenznummer* versehen wird, die dann auch im Antwortpaket verwendet wird. Das nächste vom Master abgesendete Paket wird dann mit einer um eins erhöhten Sequenznummer versehen. Dies ermöglicht, zu überprüfen, ob Pakete verlorengehen, bzw. ob ein Antwortpaket nicht zu dem vorher gesendeten Befehlspaket paßt. Die Sequenznummer liegt zwischen 0 und 15 und wird, nachdem sie 15 angenommen hat, wieder auf 0 zurückgesetzt.

Zu 2.: Damit mehrere Slaves angesteuert werden können, wird im Bedarfsfall eine Adresse verwendet, mit der ein einzelnes Gerät angesteuert wird. Diese Adressen haben einen Bereich von 1–255 (0 ist für den *Broadcast*–Modus reserviert, siehe Kapitel 2.6 auf Seite 17).

Zu 3.: Jedes Datenpaket enthält ein *Antwortflag*, aus dem hervorgeht, ob der Master ein ausführliches Antwortpaket erwartet (das dann ebenfalls die hier besprochenen Eigenschaften hat), oder ob er nur ein einzelnes Byte[1] als Antwort erhält.

Wenn der Master als Antwort nur ein einzelnes Byte verarbeiten möchte, so bedeutet eine empfangene 0x00: "Kein Fehler aufgetreten, Slave ist bereit, neues Paket zu empfangen". Im Fall, daß ein Fehler aufgetreten ist, sendet der Slave "0xFF" und ist ebenfalls danach bereit, neue Pakete zu empfangen.

Zu 4.: Zu Beginn der Anwendungsentwicklung oder für einfache Anwendungen auf dem Master kann das hier beschreibene Protokoll in einer sehr einfachen Variante implementiert werden. Die zusätzlichen Möglichkeiten, die das Protokoll bietet, können dann nach Bedarf nachgerüstet werden. Um das LCM–Modul noch einfacher ansteuern zu können, gibt es die Möglichkeit, einen *ASCII*–Modus zu verwenden. Damit lassen sich z.B. einfache Terminalprogramme zur Ansteuerung verwenden. Der ASCII–Modus ist jedoch nicht so effizient wie der Binärmodus und bietet auch verschiedene Vorteile des Binärmodus (wie z.B. Checksummen, Kontrolle der Sequenznummern, Multislavebetrieb) nicht. Beispiele in ANSI–C finden Sie im Kapitel 5 auf Seite 60.

Die Datenpakete, die zur Übermittlung der Daten zwischen Master und Slave verwendet werden, haben den in Tabelle 1 beschriebenen Aufbau.

Um dieses allgemeine Protokoll für die Grafikfunktionen des LCM–Moduls zu nutzen, enthalten die Nutzdaten Informationen darüber, welche Funktion ausgeführt werden soll. Welche Daten für welche Funktion zu übertragen sind, ist in der Funktionsreferenz beschrieben. Im folgenden werden diese Daten auch als "Befehle" bezeichnet. Es ist zweckmäßig, die Zusammenstellung der Daten für die Befehle im ansteuernden Gerät von Funktionen ausführen zu lassen. Solche Funktionen in ANSI–C sind als Beispiel in Kapitel 5 auf Seite 60 beschrieben und befinden sich auch auf der CD des Starter–Kits. Da sich diese Funktionen mit ihren Parametern gut für die Repräsentation der Befehle eignen, verwenden wir die Namen der Funktionen synonym für die Befehle.

[1] Im Kurzantwortmodus, bei dem nur 0x00 bzw. 0xFF als Antwort gesendet wird, machen alle Befehle, die ansonsten einen Status oder andere Ergebnisse im Antwortpaket zurück senden, keinen Sinn.

	Bestandteil	Länge	Beschreibung
1.	Paketlänge	1 Byte	Länge des gesamten Pakets in Bytes (max. 105)
2.	Paketstatusbyte	1 Byte	siehe Tabelle 2
3.	Slave Adresse	1 Byte	Adresse des angesprochenen Moduls, entfällt, wenn keine Adressierung erfolgt
4.	Daten	bis 100 Byte	Nutzdaten
5.	Checksumme	2 Byte	einfache 16-Bit-Checksumme oder CRC16, Byte1 = MSB, Byte2 = LSB

Tabelle 1: Aufbau eines Datenpakets

Bit	Bestandteil	
7	Paketlänge	1 = Antwortpakete ausführlich, 0 = Antwort aus nur 1 Byte
6	Adressmode	1 = Adressierung ein, 0 = aus
5	Art der Checksumme	1 = CRC16, 0 = einfache Checksumme
4	Sequenzkontrolle	1 = Sequenzkontrolle ein, 0 = aus
0-3.	Sequenznummer	wird mit jedem Paket hochgezählt

Tabelle 2: Aufbau des Paketstatusbyte

Befehle für und Antworten vom LCM: Die in den Nutzdaten übertragenen Bytes werden vom GUI–Interpreter des LCM als Befehl interpretiert. Dabei entsprechen die ersten zwei Bytes immer der Befehlskennung (*Kommando-ID*) (Byte 1: MSB, Byte 2: LSB). Die folgenden Bytes sind die Parameter des Befehls, wobei bei Parametern, die mehrere Bytes beanspruchen (16– oder 32–Bit–Werte) immer die höherwertigen Bytes zuerst übertragen werden.
Antwortpakete, die das LCM zurück an den Master sendet, enthalten mindestens den Antwortpakettyp (zwei Bytes) und danach eventuell Rückgabewerte der aufgerufenen Kommandos, wie z.B. den Error–Code (siehe Kapitel A auf Seite 71).
Die *Antworttypen* beinhalten Statusinformationen über das Kommunikationsprotokoll und werden hauptsächlich für Änderungen der Protokolleinstellungen verwendet und wenn Fehler auftreten. Im normalen Betrieb sind die Pakete vom Typ

LCM_OK_RESPONSE. Die möglichen Antwortpakettypen sind in Tabelle 3 beschrieben; die Werte sind in “serial_commands.h” definiert.

Typbezeichnung	Wert	Bemerkung
LCM_OK_RESPONSE	1	vorheriges Paket korrekt empfangen
LCM_ADDRESS_CHANGED	2	LCM–Adresse wurde geändert (siehe Multi–Slave–Modus)
LCM_CHECKSUM_CHANGED	3	Art der Checksumme wurde geändert
LCM_SEQUENCE_CHANGED	4	Art der Sequenz-Behandlung wurde geändert
LCM_BAD_CHECKSUM	5	Falsche Checksumme im letzten Paket
LCM_TIMEOUT	6	Zeitüberschreitung für erwartetes Paket im LCM
LCM_SEQUENCE_ERROR	7	Abfolge der Sequenzen nicht in Ordnung

Tabelle 3: Typen der Antwortpakete

2.4 Nutzung der Protokolleigenschaften

Im folgenden werden die Optionen des Protokolls für eine Konfiguration mit *einem* Display-Modul im Binärmodus beschrieben. Die Besonderheiten, die für einen Betrieb mit mehreren Modulen zu beachten sind, werden gesondert in Kapitel 2.6 auf Seite 17 behandelt.

Ausgangszustand der LCM-Module nach dem Einschalten: Nach dem Einschalten sind die LCM–Module so eingestellt, daß sie Pakete folgendermaßen erwarten:

- Einstellung der Schnittstelle: Baudrate (abhängig vom Autobauding) bis zu 115200 Baud, 8 Datenbits, 1 Stopbit, keine Parität.
- ASCII–Modus. Vom ASCII–Modus wechselt man in den Binär–Modus durch Senden des Strings “zBIN”, abgeschlossen mit *Carriage Return* (0x0d). Dies

wird vom LCM im Erfolgsfall mit `0x00` quittiert. Wird kein Zeichen oder etwas anderes als `0x00` zurückgeliefert, so ist ein Fehler aufgetreten.

- Antworten des LCM, sofern vorgesehen, bestehen nur aus einem Byte.
- Es findet keine Adressierung statt (Single–Slave–Betrieb).
- Es wird zur Überprüfung der Übertragung lediglich eine 16–Bit–Summe verwendet, keine CRC16–Prüfsumme.
- Es findet keine Sequenzkontrolle statt, d.h. die Pakete werden nicht auf fortlaufende Sequenznummern überprüft.
- Timeout-Einstellung = 0.1 s (siehe Abschnitt 2.5 auf Seite 17)

Abbildung 4 zeigt eine Gesamtübersicht über die Kommunikation im Binärmodus. Bitte beachten Sie: Auf Seiten des ansteuernden Gerätes (Master) sind nur die Protokollbestandteile zu programmieren, soweit sie in der Anwendung verwendet werden sollen. Empfehlenswert ist oft, zunächst mit der einfachsten Protokollversion zu beginnen und dann die weiterführenden Komponenten des Protokolls zu implementieren.
Um den Modus, wie das Übertragungsprotokoll arbeiten soll, zu bestimmen, stehen die folgenden Befehle zur Verfügung. Nachdem ein solcher, das Übertragungsprotokoll bestimmende Befehl an das Display–Modul gesendet wurde, sendet dieses *keine* Antwort zurück.

`LCMReset:` Führt einen Software–Reset des LCM aus.

`LCMServiceMode:` Versetzt das LCM in einen Service–Mode, um z.B. eine neue Version des GUI–Interpreters einzuspielen.

`LCMAutobauding:` Versetzt das LCM in den Autobauding-Modus, so daß die Übertragungsrate neu festgelegt werden kann.

`LCMcrc16on:` Schaltet die Paketüberprüfung per CRC16 ein.

`LCMcrc16off:` Schaltet die Paketüberprüfung per CRC16 aus. Es wird dann nur noch eine einfache 16–Bit–Summe der sonstigen im Paket vorhandenen Bytes als Checksumme verwendet.

`LCMAddressModeOn:` Schaltet die Adressierung für LCM–Module ein, d.h. Pakete werden von den Modulen nur interpretiert, wenn die jeweils vereinbarte Adresse übereinstimmt oder auf der Broadcastadresse 0 gesendet wird (notwendig für Multi–Slave–Betrieb, siehe auch Kapitel 2.6 auf Seite 17, funktioniert aber auch im Betrieb mit nur einem Display–Modul).

`LCMSequenceControlOn:` Schaltet die Überprüfung der Sequenznummern ein.

`LCMSequenceControlOff:` Schaltet die Überprüfung der Sequenznummern aus.

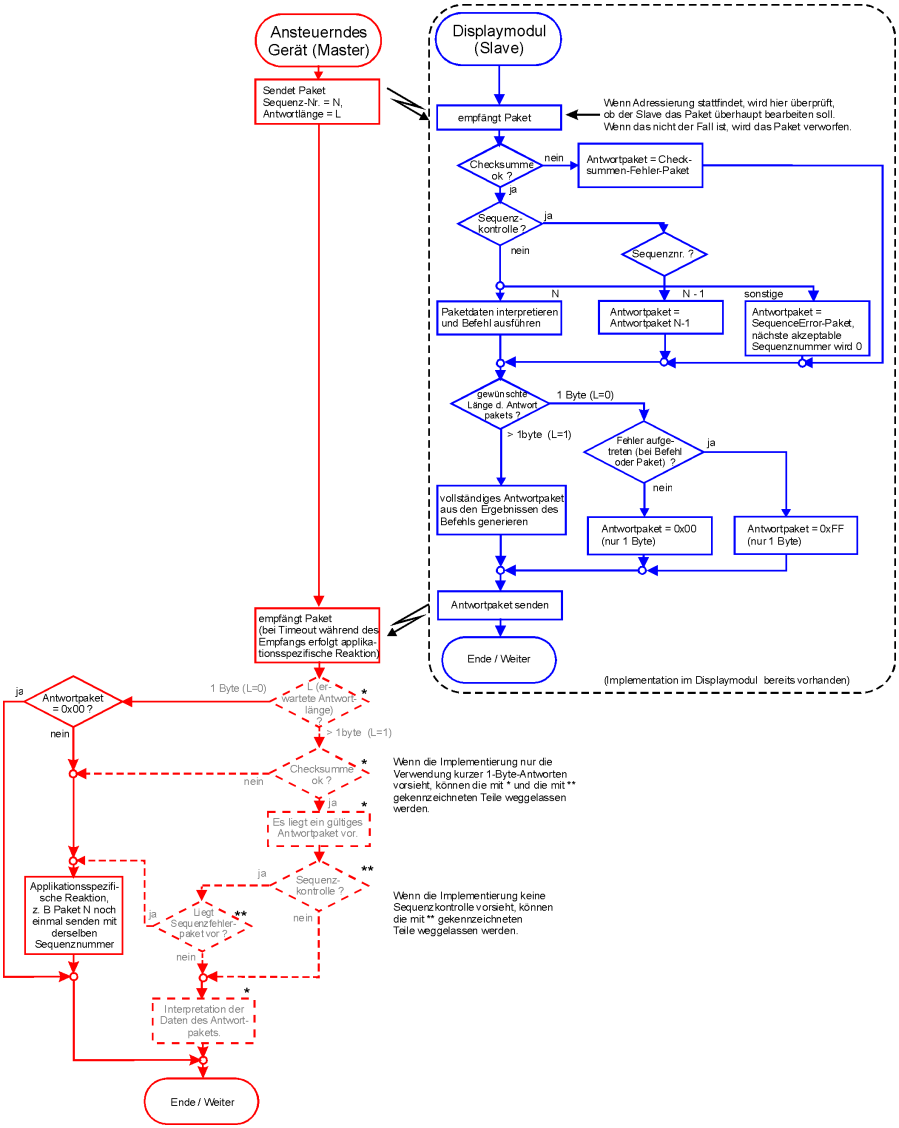


Abbildung 4: Kommunikationsprotokoll

2.4.1 Protokoll – einfachste Version

In der einfachsten Ausführung brauchen die weiterführenden Möglichkeiten des Binärmodus nicht implementiert zu werden. Es wird daher nur folgende Funktionalität auf der Seite Ihres ansteuernden Gerätes (Master) benötigt, wie auch aus Abbildung 4 auf der vorhergehenden Seite entnommen werden kann (dort brauchen die gestrichelten Funktionen nicht realisiert zu werden):

- Senden der Befehlspakete. Paketstatusbyte = 0x00, Checksumme wird durch einfaches Aufsummieren sämtlicher Bytes des Paketes gewonnen.
- Empfangen eines Bytes als Antwortpaket.
- Interpretation dieses Bytes als “OK” für 0x00 oder als “Error” für 0xFF.

In Kapitel 5 auf Seite 60 ist beispielhaft ein solcher einfacher Master implementiert, den Sie für Ihre Plattform anpassen können, um zügig zu einem funktionsfähigen System zu kommen.
Beachte: Viele Befehle liefern Rückgabeparameter, die in der einfachen Protokollversion nicht berücksichtigt werden.

2.4.2 Protokoll – Nutzung aller Möglichkeiten

Um alle Möglichkeiten des Protokolls nutzen zu können, muß die gesamte auf der Master–Seite in Abbildung 4 auf der vorhergehenden Seite gezeigte Funktionalität implementiert werden. Hierzu zählt:

- Berechnung und Auswertung von CRC16–Checksummen.
- Auswertung der Sequenznummern für die Verwendung der Sequenzkontrolle und entsprechende Reaktionen für den Fehlerfall (hier empfiehlt es sich typischerweise, das Paket, auf das mit einer fehlerhaften Sequenznummer geantwortet wurde, noch einmal zu senden).
- Interpretation der Daten des Antwortpaketes. Diese liefern Informationen oder Parameter, die für den vorangegangenen Befehl spezifisch sind (Status, evtl. Rückgabe von Parametern). Die befehlsspezifischen Antworten sind in der Funktionsreferenz aufgeführt.

Auch für diese erweiterten Möglichkeiten finden Sie in Kapitel 5 eine Beispielimplementation, so daß Sie diese relativ leicht für Ihr System anpassen können.

2.5 Übersicht über die Interpreter-Funktionen für die Kommunikation

LCM_CRC16_OFF: Schaltet die CRC-Checksummen der Pakete um auf einfache Checksumme (Summe über alle Bytes).
LCM_CRC16_ON: Die Paketübermittlung wird mit CRC16-Checksummen gesichert.
LCM_ADDRESS_MODE_ON: Schaltet den Adress-Mode des GUI-Interpreters ein (wird noch nicht unterstützt).
LCM_SEQUENCE_CONTROL_OFF: Schaltet die Sequenzierung von Paketen aus.
LCM_SEQUENCE_CONTROL_ON: Schaltet die Sequenzierung von Paketen ein.
LCM_AUTOBAUDING: Versetzt das Gerät in den Autobauding–Mode.
LCM_WARMSTART: Setzt das LCM zurück und startet den GUI–Interpreter neu.
LCM_SET_TIMEOUT: Setzt den Timeout des GUI–Interpreters beim Empfang von Daten über die serielle Schnittstelle.
LCM_ASCII: Stellt den GUI-Interpreter auf den ASCII-Mode um.
LCM_GET_LAST_RESPONSE: Fordert das Antwortpaket des letzten Befehls nocheinmal an.
LCM_GET_LAST_COMMAND_CHECKSUM: Holt die Checksumme des letzten erfolgreichen Kommandos, welches an den GUI–Interpreter gesendet wurde.
LCM_PING: Veranlaßt den GUI–Interpreter lediglich eine positive Antwort zu senden.
LCM_GET_SYSTEM_INFO: Holt einige Daten des angeschlossenen Slave-Systems (LCM–Modul).

2.6 Betrieb mit mehreren Slaves

Diese Funktionalität ist vorgesehen z. Zt. aber noch nicht implementiert

3 Module des GUI-Interpreters

Hier werden die einzelnen Funktionsmodule des GUI-Interpreters beschrieben, um eine allgemeine Übersicht über die Funktionsweise und Leistungsfähigkeit dieser Module sowie über die verfügbaren Funktionen zu geben. Detaillierte Informationen über die einzelnen Funktionen und ihre Verwendung erhalten Sie in der Funktionsreferenz.
Für jede Funktionsgruppe existiert ein eigenes Modul. Für das Auslösen der einzelnen Befehle über die serielle Schnittstelle durch Ihr ansteuerndes Gerät (*Master*) finden Sie im Kapitel 5 auf Seite 60 eine Beispielimplementationen in ANSI-C. Sie können natürlich auch jede andere Programmiersprache für Ihr Gerät verwenden, um die Ansteuersequenzen zu erzeugen.
Im folgenden werden die einzelnen Funktionsmodule anhand der Beispielimplementation des Masters aus Kapitel 5 beschrieben, die in diesem Kapitel verwendeten Header-Dateien sind diejenigen für diese Implementation. Diese Header-Dateien bieten ebenfalls Detailinformationen zu den einzelnen Funktionen.
Die einzelnen Module bestehen jeweils aus einer C-Datei mit den Funktionsimplementationen und einer Header-Datei. Die Header-Dateien der Module sind im einzelnen:

display.h: Include-File mit Funktionen zur Grafikausgabe.

lcd.h: Include-File für die spezifischen Funktionen zur Steuerung des LCD-Panels.

system.h: Definitionen für Zeichensätze und ihre Funktionen.

sound.h: Include-File für die Ansteuerung eines optionalen Piezo-Lautsprechers.

touch.h: Include-File für Funktionen des Touchglases. Diese ermöglichen Positionseingaben auf dem Display.

button.h: Include-File für Funktionen zur Realisierung von „Buttons“(Bedienknöpfe für das Anwendungsprogramm).

tglass.h: Include-File für die Kalibrierfunktionen des verwendeten Touch-Glases.

keyboard.h: Include-File mit Funktionen zur Kontrolle von Tastaturen im allgemeinen (vgl. `keyboard`-Channel in Abschnitt 3.8).

t_keyb.h: Include-File mit Funktionen zur Darstellung und Kontrolle von Touch-Tastaturen (vgl. `t_keyb`-Device in Abschnitt 3.9 auf Seite 31).

lcm.h: Include-File für Funktionen, die keine Entsprechung in der GUI-Bibliothek finden, sondern spezifisch für das LCM-Modul sind. Dies sind die Funktionen zur Einstellung des Kommunikationsprotokolls und Funktionen zur Verwaltung binärer Objekte, die in das LCM geladen werden können.

Darüberhinaus gibts es sogenannte *Addon*-Module, die komplexere Funktionen auf Basis der schon beschriebenen Funktionsgruppen bereitstellen.

event_handling.h: Include-File mit Funktionen zur Kontrolle von "Touch-Ereignissen". Hiermit können Buttons, Tastaturen und ähnliche Steuerelemente auf eine einheitliche Art und Weise abgefragt werden (siehe Abschnitt 3.11 auf Seite 33).

bar_indicator.h: Include-File mit Funktionen zur Anzeige von Fortschrittsbalken.

clip.h: Funktionen zum Testen und beschneiden von Linien und Punkten gegenüber einem rechteckigen Bereich.

inputline.h: Include-File mit Funktionen, die eine Texteingabezeile bereitstellen.

popup_menu.h: Funktionen zur Darstellung und Kontrolle von Popup-Menüs.

menu.h: Funktionen mit denen sich komplexere Menüs zusammenbauen und steuern lassen.

seven_ segment.h: Funktionen zur Dartstellung von Sieben-Segment-Anzeigen.

slider.h: Include-File mit Funktionen zur Anzeige und Steuerung von Schieberegelern.

selection_list.h: Include-File mit Funktionen zur Anzeige und Steuerung von Auswahllisten.

logger_data.h: Funktionen zum Verwalten von Datensätzen, die später in Diagrammen dargestellt werden sollen.

logger_disp.h: Funktionen zum Zeichnen von Diagrammen und Daten.

encoder.h: Funktionen zur Abfrage eines optischen Encoders (optional).

guii_animation.h: Funktionen zur Erstellung von piktogrammartigen Animationen, die im Hintergrund ablaufen können.

Die Funktionsnamen für die Programmierung der grafischen Benutzeroberfläche (GUI) beginnen mit dem Namen des jeweiligen Moduls (z.B. mit `disp_...` für Funktionen für das Display, z.B. `disp_set_pixel`).

Zusätzlich zu den Modulen sind noch folgende wichtige Dateien vorhanden:

portab.h, portab_guii.h: Definition portabler Dateitypen und wichtiger Konstanten.

errors.h: Definition der Fehlercodes für die Grafikfunktionen.

errsys.h: Definition der Fehlercodes für die Kommunikation und sonstige Fehler des LCM.

errors_guii.h: Definition der Fehlercodes, die spezifisch für den GUI-Interpreter sind.

config_master.h, global_settings_guii.h: Konfigurationsdateien mit Definitionen für die Kommunikation mit dem LCM.

serial_commands.h: Definiton aller Befehlskennungen des GUI-Interpreters im binären Modus.

serial_command_buffer.h: Include-File mit Funktionen zur Bearbeitung der Kommandopakete im binären Modus.

serial_transfer.h: Include-File mit Funktionen zum Übertragen und Empfangen von Paketen im binären Modus und zur Steuerung des Übertragungsprotokolls.

serial_local.h: Include-File mit Funktionen zum Öffnen und Schließen der seriellen Schnittstelle des Masters und zum Übertragen und Empfangen von Daten. (Dieses Modul muß entsprechend der Hardware Ihres Masters angepaßt werden.)

3.1 Eigenschaften von Funktionen und Variablen

Um möglichst hardwareunabhängig programmieren zu können, wurden die verschiedenen implementationsabhängigen Integer-Typen von ANSI-C nicht verwendet. An ihre Stelle treten die in der Datei “portab.h” definierten Typen, wie z.B. `int16` oder `uint8` (der erste ist eine vorzeichenbehafteter Integer-Typ mit einer Länge von 16 bit, der zweite ein Integer ohne Vorzeichen mit 8 Bit). In der Datei “portab.h” sind außerdem weitere Variablentypen vereinbart, wie z.B. `color`, `font` und `err_code`. **Um Teile des Beispielcodes für Ihren Master zu verwenden, ist es wichtig, daß Sie die Definitionen in “portab.h” entsprechend der Datentypen des von Ihnen verwendeten Compilers einstellen. Diese finden Sie dort in der Datei ”limits.h”, die zum Umfang des C-Compilers gehört.**
Der Typ `err_code` bezeichnet den Rückgabetyp der Funktionen. Alle Funktionen geben einen Error-Code vom Typ `err_code` zurück, an dem festgestellt werden kann, ob die Funktion erfolgreich durchgeführt werden konnte, oder welche Art von Fehler vorliegt. Die Definitionen der Error-Codes erfolgen in den Dateien “errors.h”, “errsys.h” und “errors_guii.h”. Insbesondere bedeutet der Rückgabewert `ERR_OK`, daß kein Fehler aufgetreten ist. Bei Funktionen, die Werte zurückgeben sollen, wird dies über Adreßzeiger in der Parameter-Liste der jeweiligen Funktion realisiert (dies wird in der Funktionsreferenz an Beispielen bei den entsprechenden Funktionen verdeutlicht).

3.2 System-Funktionen

Die System-Funktionen erledigen Aufgaben, die für das ganze System zentral abgewickelt werden. Dies sind z. B. Funktionen für die Schriftarten (Fonts).

Übersicht System-Funktionen:

SYSTEM_FONT_INIT: Initialisiert die Daten eines Fonts.
SYSTEM_FONT_CLOSE: Schließt den Font und gibt den verwendeten Speicher frei.
SYSTEM_GET_NUMBER_OF_FONTS: Liest die Anzahl der im System momentan verfügbaren Zeichensätze.
SYSTEM_GET_FONT_POINTER: Holt die Objekt–ID für einen mit Namen spezifizierten Font.
SYSTEM_GET_FONT_ASCENT: Holt die Anzahl der Pixel vom oberen Ende des Fonts bis zu seiner "Baseline".
SYSTEM_GET_TEXT_ASCENT: Holt die Anzahl der Pixel vom oberen Ende des Texts bis zu seiner "Baseline". Hierbei wird der benutzte Font und der Textstil berücksichtigt.
SYSTEM_GET_FONT_PIXELHEIGHT: Holt die Gesamthöhe des Fonts (in Pixel).
SYSTEM_GET_TEXT_PIXELHEIGHT: Holt die Gesamthöhe des Fonts (in Pixel). Hierbei wird der benutzte Font und der Textstil berücksichtigt.
SYSTEM_GET_FONT_AVG_WIDTH: Holt die durchschnittliche Breite der Zeichen des Fonts.
SYSTEM_GET_FONT_MAX_WIDTH: Holt die Breite des breitesten Zeichens eines Fonts.
SYSTEM_GET_FONT_NAME: Liefert den Namen eines Fonts als nullterminierten String.
SYSTEM_GET_CHARWIDTH: Holt die Breite eines bestimmten Zeichens eines Fonts (in Pixel).
SYSTEM_GET_TEXTWIDTH: Holt die Breite eines Textstrings in einem bestimmten Font (in Pixel).
SYSTEM_GET_TICKS: Liest den Wert des System-Timers aus.

3.3 Display-Funktionen

Die Koordinaten werden von der linken unteren Ecke an gezählt, d.h. die linke untere Ecke hat die Koordinaten x=0, y=0.
Bitte beachten Sie, daß aus Gründen der Ausführungsgeschwindigkeit auf ein *Clipping* verzichtet wird. Das bedeutet, daß Grafikbefehle, die über den Rand des Displays herausführen würden, abhängig von der Funktion nicht in jedem Fall korrekt

bis zum Rand, sondern eventuell gar nicht oder mit Artefakten ausgeführt werden.

Viele Funktionen werden relativ zum aktuellen *Grafikcursor* ausgeführt. Dieser bezeichnet kein auf dem Display erscheinendes Cursor-Symbol, sondern eine *aktuelle Position* zum Zeichnen auf dem Display. Verschiedene Funktionen verändern den aktuellen Grafikcursor (d.h. also die Position, zu der die folgenden Funktionen relativ arbeiten). Durch diese Arbeitsweise relativ zu einer aktuellen Position kann leicht bezüglich einer vorher eingestellten Startposition gearbeitet werden, so daß, wenn das erwünscht ist, schnell eine Veränderung der Position der gesamten Ausgabe auf dem Display erfolgen kann, indem nur die Startposition geändert wird.
Die Wirkung einer Grafik-Funktion ist außerdem vom *Zeichenmodus* abhängig. Dieser bestimmt, wie das zu zeichnende Element mit dem bereits vorhandenen Hintergrund kombiniert wird. Neben dem Standardmodus `SET`, bei dem der Hintergrund einfach ersetzt wird, sind die logischen Verknüpfungen `OR`, `EXOR` und `AND` verfügbar.

Die Funktionen für das Display können in folgende Kategorien unterteilt werden:

Funktionen für einfache grafische Befehle: Die meisten Funktionen werden relativ zur aktuellen Cursorposition ausgeführt. Das Setzen des Cursors (einer gedachten aktuellen Position auf dem Display) selbst kann absolut oder relativ zu der vorherigen Position geschehen. Die elementaren Funktionen für das Setzen und Lesen von Pixeln sind ebenfalls sowohl mit Absolutkoordinaten als auch relativ zur Cursorposition vorhanden. Durch diesen Ansatz können grafische Strukturen durch einfaches Repositionieren des Cursors an einer anderen Stelle angezeigt werden, ohne nochmals zahlreiche Koordinaten berechnen zu müssen.

Funktionen für das Zeichnen von Linien ermöglichen, unterschiedliche Liniendicken und Linienstile (z.B. durchgezogen oder gestrichelt) zu verwenden. Für die Liniendicke gilt folgendes: Die Linien werden um die ursprünglich verwendete Ausgangskoordinate verbreitert, wenn die Liniendicke größer als 1 gewählt wird. Bei geraden Werten für die Liniendicke wird zunächst in Richtung positiver x- bzw. y-Koordinaten verbreitert. Beispiel: Für eine horizontale Linie mit Liniendicke 2 wird die Linie auf der ursprünglichen y-Koordinate sowie auf der Koordinate y+1 (oder x+1 für Linien mit Steigung größer als 1) ausgeführt. Erhöht man die Liniendicke auf 3, so erfolgt das Zeichnen der Linie nun auf den Koordinaten y-1, y, y+1 (bzw. bei x-1, x, x+1, wenn die Steigung größer 1 ist). Mit Liniendicke = 4 kommt zusätzlich y+2 (oder x+2) hinzu, mit Liniendicke = 5 wird auch bei y-2 (oder x-2) gezeichnet, usw.

Der Linienstil (`linestyle`) wird durch einen 16-Bit-Wert definiert. In die-

sem repräsentiert ein gesetztes Bit einen (in der aktuellen Farbe `current_color`) gesetzten Punkt in der Linie und ein gelöschtes Bit keinen gesetzten Punkt (dort wird die aktuelle Hintergrundfarbe gesetzt). Das Linienmuster beginnt mit dem höchstwertigen Bit und wiederholt sich, sofern das zu zeichnende Linienelement länger als 16 Pixel ist.

Funktionen für die Bearbeitung von rechteckigen Bereichen: Um Flächen auf dem Display zu bearbeiten, stehen Füllstile zur Verfügung. Außerdem können Bilder im verbreiteten BMP-Format dargestellt werden. Füllstile werden verwendet, um rechteckige Flächen auszufüllen.

Es stehen Funktionen zum Zeichenen von Rahmen zur Verfügung, wie sie bei Buttons benutzt werden. Die Rahmen können abgerundete Ecken haben und schattiert sein. Hierbei wird das Aussehen der Rahmen wiederum über einen *Style* definiert.

Weitere Details der Füllstile, Bilder und Rahmen werden bei den jeweiligen Funktionen in der Funktionsreferenz beschrieben.

Funktionen für die Textausgabe: Text läßt sich mit verschiedenen (auch proportionalen) Zeichensätzen darstellen. Er kann außerdem mit üblichen Attributen wie “unterstrichen” , “fett” , “kursiv” und “hell” versehen werden. Für Farbdisplays ab einer Farbtiefe von 8 Bit kann der Text kann auch mit Antialiasing ausgegeben werden, um ein glatteres Aussehen zu erhalten. Dabei wird der Text in der halben Größe des verwendeten Zeichensatzes dargestellt. Bei 24-Bit-Farbtiefe wird Text mit dem Attribut “hell” durch eine geeignete Farbe dargestellt und nicht mehr gerastert. Die vertikale Ausgabe von Text funktioniert nur mit den Attributen “hell” und “invertiert” .

(Zu den Attributen siehe auch die Beschreibung des Befehls `disp_set_textstyle()`).

Die Textfunktionen verwenden für ASCII-Texte Fonts im Windows 2.0 und 3.0 Format (Dateiendung FNT), so daß es leicht möglich ist, weitere Fonts in das System zu übernehmen. Drei Zeichensätze sind in jedes Display-Modul integriert: “Mono6x8”, “Mono8x16” und “Prop14”.

Die Darstellung von UTF8 kodierten Texten (z.B. für asiatische Schriftarten) wird auf die folgende Art und Weise realisiert: Es gibt ein Windows-Tool (*SimplifyCharacterCompiler*) welches einen sogenannten *SSF-Font* erzeugt, der beliebig viele Zeichen enthalten kann und nicht auf 256 beschränkt ist wie das FNT-Format. Hierzu gibt man diesem Tool den gewünschten Truetype-Font vor und einen Text, der alle benötigten Zeichen mindestens einmal in der UTF8-Kodierung enthält. Intern werden daraus mehrere FNT-Fonts mit jeweils 256 Zeichen erstellt und eine Übersetzungstabelle die jedem Unicode-Zeichen eine FNT-Font-Nummer und eine Zeichennummer zuordnet. Die FNT-Fonts und die Tabelle werden zu einem SSF-Font zusammengefügt. Dieser

SSF-Font kann dann als Binär- oder h-File für die GUI-Bibliothek abgespeichert werden. Die Font-Befehle (aus “system.h/c” bzw. “display.h/c”) der GUI-Bibliothek arbeiten mit den SSF-Fonts sowie den herkömmlichen FNT-Fonts zusammen. Mit dem Befehl `disp_text()` kann man dann UTF8 kodierte Strings ausgeben und die entsprechenden Zeichen erscheinen auf dem Display, vorausgesetzt der mit `disp_set_font()` eingestellte SSF-Fonts enthält diese Zeichen dann auch.

Wechselwirkung mit Funktionen des Touch-Channels: Zu beachten ist, daß mit den Befehlen des Display-Channels auch die Darstellung von Buttons und Tastaturen auf dem Display unzulässig verändert werden kann (z.B. werden durch einen Aufruf von `disp_clear()`, solange noch Buttons oder Tastaturen aktiv sind, diese zwar auf dem Display verschwinden, die Touch-Funktionalität bleibt jedoch erhalten). Entfernen Sie daher in solchen Fällen vorher die entsprechenden Touch-Elemente vom Display.

Übersicht Display-Funktionen:

DISP_RESERVE_BMP_COLORS: Reserviert eine bestimmte Anzahl Farben aus der Farbpalette für die Benutzung von BMP–Bildern.
DISP_SET_PALETTE_ENTRY: Erlaubt das Setzen eines Eintrags in der Farbpalette.
DISP_CLEAR: Löscht Display mit der aktuellen Hintergrundfarbe.
DISP_SET_CURSOR_POSITION: Setzt Cursorposition in absoluten Koordinaten.
DISP_GET_CURSOR_POSITION: Liest die aktuelle Cursorposition.
DISP_MOVE_CURSOR: Bewegt Cursorposition relativ zur aktuellen Cursorposition.
DISP_SET_PIXEL: Setzt ein Pixel in der aktuellen Farbe auf der aktuellen Cursor-Position.
DISP_SET_PIXEL_ABS: Setzt ein Pixel in der aktuellen Farbe auf der angegebenen absoluten Position.
DISP_GET_PIXEL: Liest die Farbe des Pixels an der aktuellen Cursorposition.
DISP_GET_PIXEL_ABS: Liest die Farbe des Pixels an einer absoluten Position.
DISP_LINE: Zeichnet eine Linie mit den momentan gültigen Linienattributen (Linienmuster und Dicke) von und relativ zur aktuellen Cursorposition.
DISP_LINE_ABS: Zeichnet eine Linie mit den momentan gültigen Linienattributen (Linienmuster und Dicke) von einer absolut angegebenen Position zu einer anderen absolut angegebenen Position.
DISP_HLINE: Zeichnet eine horizontale Linie mit den momentan gültigen Linienattributen (Linienmuster und Dicke) von und relativ zur aktuellen Cursorposition.
DISP_VLINE: Zeichnet eine vertikale Linie mit den momentan gültigen Linienattributen (Linienmuster und Dicke) von und relativ zur aktuellen Cursorposition.

DISP_RECTANGLE: Zeichnet ein Rechteck relativ zur aktuellen Cursorposition (mit allen Linienattributen).
DISP_COLORED_RECTANGLE: Zeichnet ein Rechteck relativ zur aktuellen Cursorposition mit der aktuellen Farbe.
DISP_FILLED_RECTANGLE: Zeichnet ein mit Muster ausgefülltes Rechteck relativ zur aktuellen Cursorposition (ohne Rahmen).
DISP_COLORED_TRIANGLE: Zeichnet ein mit der aktuellen Farbe gefülltes Dreieck.
DISP_CIRCLE: Zeichnet einen Kreis mit der aktuellen Cursorposition als Kreismittelpunkt.
DISP_COLORED_CIRCLE: Zeichnet einen mit der aktuellen Farbe ausgefüllten Kreis relativ zur aktuellen Cursorposition.
DISP_ARC: Zeichnet ein Kreissegment mit der aktuellen Cursorposition als Kreismittelpunkt zwischen einem Startwinkel und einem Endwinkel.
DISP_GET_COLOR: Liest die Farbe, die aktuell im Display verwendet wird.
DISP_SET_COLOR: Setzt die Farbe, die im Display aktuell verwendet werden soll.
DISP_GET_BACKGROUND_COLOR: Holt die aktuelle Hintergrundfarbe zum Zeichnen auf dem Display.
DISP_SET_BACKGROUND_COLOR: Setzt die aktuelle Hintergrundfarbe zum Zeichnen auf dem Display.
DISP_GET_LINESTYLE: Liest den aktuell verwendeten Linienstil des Displays.
DISP_SET_LINESTYLE: Setzt einen neuen Linienstil zur Benutzung auf dem Display.
DISP_GET_LINEWIDTH: Liest die aktuell verwendete Liniendicke.
DISP_SET_LINEWIDTH: Setzt neue Linien-Dicke für die Verwendung auf dem Display.
DISP_GET_DRAWMODE: Liest den aktuell verwendeten Zeichen-Modus auf dem Display.
DISP_SET_DRAWMODE: Setzt einen neuen Zeichen-Modus für das Display.
DISP_SET_FILLSTYLE: Setzt einen neuen Füllstil (fillstyle) zur Benutzung auf dem Display.
DISP_BMP: Ausgabe eines Bildes im BMP-Format auf dem Display.
DISP_BMP_ALPHA: Gibt ein Bilde im BMP-Format auf dem aktuellen Display aus, wobei eine Farbe als transparent deklariert wird.
DISP_PIXELBLOCK_INIT: Initialisiert eine Pixelblock-Struktur und holt den entsprechenden Speicher.
DISP_PIXELBLOCK_FREE: Gibt den Speicher eines Pixelblocks wieder frei.
DISP_PIXELBLOCK_GET: Kopiert einen Pixelblock aus dem Display an der aktuellen Cursorposition (die dabei nicht geändert wird) in den Speicher.
DISP_PIXELBLOCK_SET: Gibt einen Pixelblock auf dem Display an der aktuellen Cursorposition (die dabei nicht geändert wird) aus.

DISP_INVERT_AREA: Invertiert ein rechteckiges Gebiet auf dem Display (relativ zur aktuellen Cursorposition, die dabei nicht geändert wird).
DISP_COLOR_CHANGE_AREA: Ändert eine Farbe innerhalb eines rechteckigen Bereichs.
DISP_FRAMESTYLE_INIT: Holt Speicher für einen Rahmenstil und initialisiert diesen.
DISP_COPY_AREA: Befehl zum kopieren rechteckiger Bereiche auf dem Display.
DISP_FRAMESTYLE_CLONE: Kopiert einen Rahmenstil und holt Speicher für die Kopie.
DISP_FRAMESTYLE_SET_COLORS: Setzt die Farbe für den Rahmenstil.
DISP_FRAMESTYLE_SET_DIMENSIONS: Setzt die Rahmenbreite und den Radius für abgerundete Ecken.
DISP_FRAMESTYLE_SET_LINESTYLE: Setzt den Linienstil eines Rahmenstils.
DISP_FRAMESTYLE_SET_FLAGS: Setzt die Konfigurations–Flags eines Rahmenstils.
DISP_FRAMESTYLE_FREE: Gibt den Speicher eines Rahmenstils wieder frei.
DISP_FRAME: Zeichnet einen Rahmen z.B. für Buttons oder andere Touch–Objekte.
DISP_DELETE_FRAME: Überschreibt den Rahmen mit der aktuellen Farbe.
DISP_TEXT: Schreibt Text an die aktuelle Cursor-Position.
DISP_FORMATTED_TEXT: Zeigt ein formatiertes Textobjekt an, das bereits in das LCM geladen wurde.
DISP_GET_TEXTSTYLE: Liest die auf dem Display momentan gültigen Text-Attribute.
DISP_SET_TEXTSTYLE: Setzt neue Text-Attribute.
DISP_GET_FONT: Liest Zeichensatz für Textausgabe.
DISP_SET_FONT: Setzt einen neuen Zeichensatz für die Textausgabe.

3.4 LCD-Funktionen

Das Modul *LCD-Device* stellt Funktionen für das in der Zielhardware eingebaute LCD-Display zur Verfügung, wie z.B. eine optionale Einstellung der Beleuchtung und die Einstellung des Kontrasts.

Übersicht LCD-Funktionen:

LCD_GET_STATUS: Liest Status des LCD-Displays.
LCD_GET_SIZE: Liest Größe des virtuellen Bildschirms des LCD-Displays.
LCD_GET_COLOR_DEPTH: Liest die Farbtiefe (in Bit) des LCDs.
LCD_ORIENTATION: Dreht das Display in 90 Grad Schritten.
LCD_LIGHT: Schaltet Beleuchtung des LCD.
LCD_GET_LIGHT: Holt die aktuelle Einstellung der Helligkeit der LCD-Hintergrundbeleucht

LCD_CONTRAST: Stellt Kontrastwert des LCD ein.
LCD_GET_CONTRAST: Liest Kontrastwert des LCD aus.
LCD_SLEEP: Schaltet das LCD aus, um Energie zu sparen.
LCD_WAKEUP: Schaltet LCD-Display nach einem LCD_SLEEP wieder ein.

3.5 Touch

Mit den Touch-Funktionen läßt sich ein berührungsempfindliches Touch-Glas als Eingabegerät benutzen (sofern das verwendete Display-Modul mit einem Touchscreen ausgestattet ist). Das Touch-Glas erlaubt die Abfrage der Positionen und der Systemzeiten dieser Berührungen (sowohl die Werte für den Beginn des Kontakts als auch die für das Ende des Kontakts). Damit lassen sich vielfältige Funktionen wie *Doppelklick* oder *Ziehen von Objekten* softwaremäßig realisieren.

Übersicht Touch-Funktionen:

TOUCH_GET_CONTACT_END: Liest die Endkoordinaten des letzten Kontaktes auf dem Touchglas.
TOUCH_GET_CONTACT_START: Liest die Anfangskoordinaten des letzten Kontaktes auf dem Touch–Glas.
TOUCH_PROCESS_OBJECTS: Testet alle Touch–Objekte mit der aktuellen Touch–Position und dem entsprechenden Zeitstempel.
TOUCH_COUNT_OBJECTS: Zählt verschiedene Touch–Objekte.

3.6 Buttons

Ein oft benötigtes Bedienungselement auf grafischen Benutzeroberflächen sind *Buttons*, also umrandete Schaltflächen, die mit einer Beschriftung oder einer Grafik versehen sind, und die wie Schalter benutzt werden.
Es stehen verschiedene Varianten von Buttons zur Verfügung. Durch entsprechende Wahl der Parameter lassen sich unterschiedliche Größen, Umrandungen und Beschriftungen (Text oder Bilder) auswählen. Das äußere Aussehen der Buttons wird über einen *Style* definiert. Außerdem lassen sich die Buttons mit unterschiedlichem Schaltverhalten ausstatten: Ein Button kann wie ein elektrischer Taster permanent aktiviert sein, solange er gedrückt wird (`FAST_BUTTON`). Er kann nur während des Loslassens aktiviert sein (`STANDARD_BUTTON`). Oder er kann zwei Zustände haben (gedrückt, nicht gedrückt), zwischen denen durch Berühren hin- und hergeschaltet wird (`HOLD_BUTTON`). In den beiden letzten Fällen wird der Schaltvorgang nicht bereits durch die Berührung ausgelöst, sondern durch das Loslassen über der Buttonfläche. Der Bediener des Gerätes kann so nach einer versehentlichen Berührung eines Buttons seine Auslösung verhindern, indem er den Kontakt mit dem Touch-Glas erst außerhalb der aktiven Button-Schaltfläche löst.

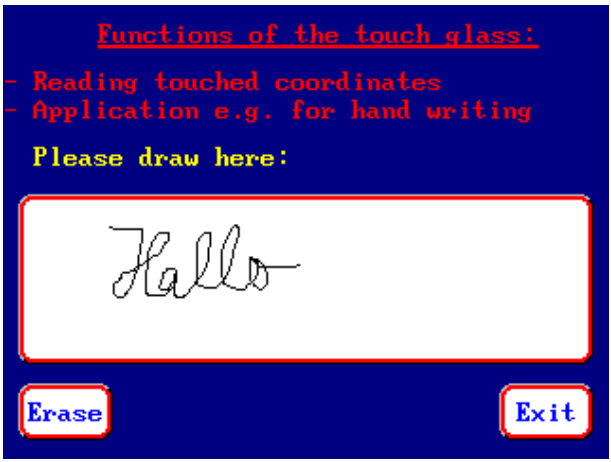


Abbildung 5: Beispiel für einfache Touch-Funktionen

Bitte beachten Sie bei der Verwendung von Buttons folgendes:

1. Bei der Darstellung von Buttons wird zwischen dem gedrückten und dem nicht gedrückten Zustand unterschieden. Beide können ein völlig unterschiedliches Aussehen in Bezug auf die Art des Rahmens oder die Farben haben, so daß hier viele flexible Gestaltungsmöglichkeiten vorhanden sind.

2. Die Größe des Buttons muß ausreichen, um den Rahmen und den Button-Text oder -Bild aufnehmen zu können. Vergrößert man die Rahmendicke so *wächst* dieser nach Innen, so daß u.U. kein Platz mehr für den Button-Inhalt bleibt.

3. Buttons können auch durch Benutzung des Textstils `TXT_VERTICAL` vertikal beschriftet werden.

Übersicht Button-Funktionen:

BUTTON_STYLE_INIT: Initialisiert einen Button-Stil.
BUTTON_STYLE_CLONE: Kopiert einen Button–Stil und holt Speicher für den neuen.
BUTTON_STYLE_FREE: Gibt den Speicher, der von einem Button–Stil belegt wird, wieder frei.
BUTTON_STYLE_SET_FONT: Setzt die Schriftart für einen Button–Stil.
BUTTON_STYLE_SET_LINEFEED: Setzt die Zeilenhöhe für den Button–Stil.
BUTTON_STYLE_SET_ALPHA: Setzt die Farbe des Alphakanals für Button-Bilder mit Transparents.

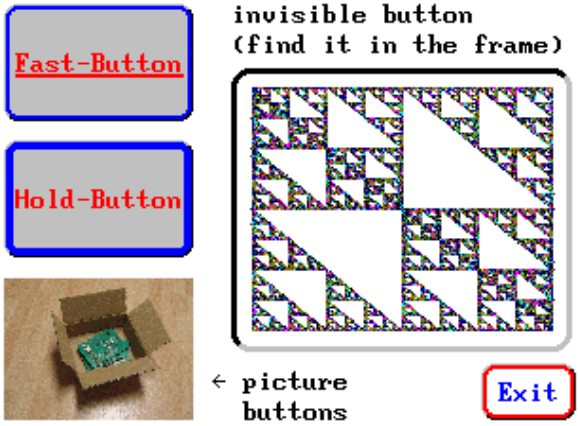


Abbildung 6: Beispiel für verschiedene Buttons

BUTTON_STYLE_SET_FRAMESTYLE: Setzt den Rahmenstil für den Button–Stil.
BUTTON_MAKE_COMBINED_LABEL: Erzeugt eine Button-Beschriftung, die sich aus einem Hintergrundbild und einem Text zusammensetzt.
BUTTON_FREE_COMBINED_LABEL: Gibt den Speicher einer kombinierten Button-Beschriftung wieder frei.
BUTTON_DEFINE: Definiert einen neuen Button.
BUTTON_UNDEFINE: Löscht einen Button und gibt den Speicher frei.
BUTTON_SET_GHOSTSTYLE: Setzt den Stil / das Aussehen eines Buttons im geghosteten Zustand.
BUTTON_SET_GHOSTLABEL: Setzt die Beschriftung (Text oder Grafik) eines Buttons im gehosteten Zustand.
BUTTON_PRESS_HOLD_BUTTON: Versetzt den Hold-Button in den gedrückten Zustand.
BUTTON_LIFT_HOLD_BUTTON: Versetzt einen Hold-Button in den nicht gedrückten Zustand.
BUTTON_GHOST_BUTTON: Deaktiviert einen Button auf dem Touch–Display und stellt die Beschriftung ”geghostet” dar.
BUTTON_UNGHOST_BUTTON: Aktiviert einen Button auf dem Touch–Display wieder, der vorher mit BUTTON_GHOST_BUTTON deaktiviert wurde.
BUTTON_ACTIVATE: Aktiviert einen bereits definierten Button und zeigt ihn auf dem Touch–Display an.
BUTTON_DEACTIVATE: Deaktiviert einen Button und löscht ihn vom Display.

BUTTON_GET_STATUS: Liest den Status eines Buttons.
BUTTON_GET_HOLD_BUTTON_STATUS: Holt den Status eines Hold-Buttons.
BUTTON_GET_POSITION: Liest die Position eines Buttons.
BUTTON_SET_POSITION: Setzt die Position eines Buttons.
BUTTON_GET_LABEL: Setzt Zeiger auf die beiden Beschriftungen eines Buttons.
BUTTON_CHANGE_LABEL: Ändert die Beschriftung (Text oder Grafik) eines Buttons.
BUTTON_SET_STYLE: Setzt den Stil / das Aussehen eines Buttons auf die Eigenschaften eines angegebenen Button–Stils.

3.7 Tglass

Das Touch-Glas ist bereits kalibriert, wenn das LCM ausgeliefert wird. Sollte dennoch eine Kalibration erforderlich sein, stehen die Funktionen des Tglass-Moduls zur Verfügung. Mit ihnen kann die Kalibrierung durchgeführt und die Kalibrierparameter gelesen und gesetzt werden.

Übersicht Tglass-Funktionen:

TGLASS_SET_CALIB_PARAMETERS: Überschreibt die aktuellen Kalibrierungsparameter des Touch-Glases mit neuen Werten.
TGLASS_GET_CALIB_PARAMETERS: Liest die aktuellen Kalibrierungsparameter des Touch-Glases.
TGLASS_CALIBRATION_SEQUENCE: Kalibriert das Tglass–Device und somit das Touch–Glas.

3.8 Keyboard-Channel

In vielen Applikationen sind Texteingaben erforderlich. Um verschiedene Eingabemöglichkeiten für Texte in einen hardwareunabhängigen Rahmen zu stellen, sind hier *Keyboard-Channel* implementiert. Diese werden auf der Zielhardware mit Touch-Tastaturen auf dem Touch-Display verbunden (es ist aber auch denkbar, externe Tastaturen zu verwenden, ohne daß die Programmierung mit Hilfe der Keyboard-Channel-Funktionen geändert werden müßte). Der Keyboard-Channel verfügt über einen Tastatur-Puffer, aus dem die ASCII-Zeichen ausgelesen werden können.

Übersicht Keyboard-Channel-Funktionen:

KEYBOARD_OPEN: Öffnet einen Keyboard–Channel.
KEYBOARD_CONNECT: Verbindet einen Keyboard-Channel mit einem Eingabegerät (z.B. eine Tastatur auf dem Touch-Glas).

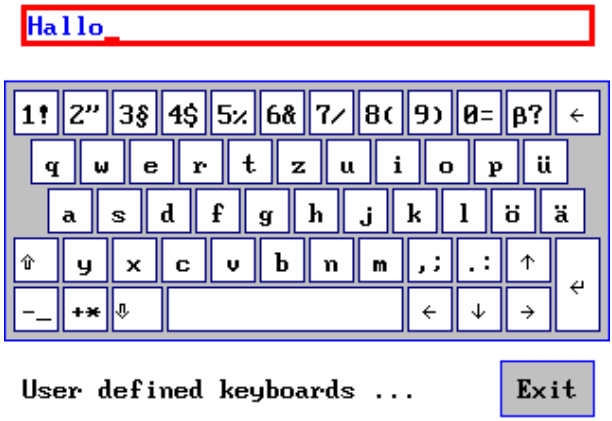


Abbildung 7: Beispiel für Tastatur

KEYBOARD_DISCONNECT: Trennt einen Keyboard-Channel von einem Eingabegerät.
KEYBOARD_CLOSE: Schließt einen Keyboard-Channel.
KEYBOARD_GETCHAR: ASCII–Zeichen aus dem Keyboard–Puffer lesen.
KEYBOARD_GET_STATUS: Lesen des Status des Keyboard-Puffers des Keyboard-Channels.

3.9 t_keyb-Device

Das Touch-Keyboard-Device `tkeyb_device` ist eine Touch-Tastatur, die auf dem Touch-Display (also einem Touch-Channel) dargestellt wird. Obwohl das `tkeyb_device` damit ausschließlich aus Software besteht, die letztlich auf vorhandene Hardware zugreift, wird es hier als Device (also wie eine Hardware-Komponente) behandelt. Im Normalfall wird ein Keyboard-Channel auf der Zielhardware mit Touch-Glas nicht mit einer externen Hardware-Tastatur verbunden, sondern eben mit einem `tkeyb_device` (als wäre es ein Stück Hardware).
Durch die flexiblen Möglichkeiten der Funktionen des `tkeyb_device` können leicht verschiedene Tastaturen für die unterschiedlichen Anwendungen konfiguriert werden, z.B. alphanumerische Tastaturen, Ziffernblöcke und andere anwendungsspezifische Tastatur-Anordnungen. Das äußere Aussehen einer Touch-Tastatur wird über einen *Style* definiert.
Wie bei *großen* Tastaturen steht auch hier eine Repeat-Funktion für längere Tastendrücke zur Verfügung, ebenso die Möglichkeit, verschiedene Zeichen auszuwählen

durch sogenannte *Modifiers*: SHIFT, ALT, CTRL, CAPS_LOCK. Bei Bedarf läßt sich solch eine Touch-Tastatur auch auf dem Display verschieben.

Übersicht t_keyb-Device-Funktionen:

T_KEYB_STYLE_INIT: Initialisiert einen Touch–Keyboard–Stil und holt Speicher dafür.
T_KEYB_STYLE_FREE: Gibt den Speicher eines Tastaturstils wieder frei.
T_KEYB_STYLE_SET_FONT: Setzt den Font für einen Touch-Keyboard-Stil.
T_KEYB_STYLE_SET_CHANGE_FLAG: Setzt das Change–Flag, wodurch das Drücken auf eine Umschalttaste (z.B. Shift oder Alt) zu einer Änderung der Tastenbeschriftungen führt.
T_KEYB_STYLE_SET_COLORS: Setzt die Farben für einen Touch-Keyboard-Stil.
T_KEYB_STYLE_SET_SELECT_COLORS: Setzt die zusätzlichen Farben eines Touch–Keyboard–Stils für den ausgewählten Zustand der Tasten.
T_KEYB_STYLE_SET_BORDER_WIDTH: Setzt Rahmenbreite des Tastaturstils.
T_KEYB_MAKE: Initialisiert ein Touch-Keyboard.
T_KEYB_FREE: Gibt den Speicher einer Tastatur–Struktur frei.
T_KEYB_OPEN: Öffnet ein Touch-Keyboard und stellt dieses dar.
T_KEYB_CLOSE: Schließt ein Touch-Keyboard und löscht es von der Anzeige.
T_KEYB_PROCESS: Erledigt Abfrage eines Touch-Keyboards.
T_KEYB_GET_POSITION: Liest die Position des Touch-Keyboards.
T_KEYB_SET_POSITION: Bewegt Touch–Keyboard auf eine neue Position.

3.10 Sound-Funktionen

Für den alltäglichen Einsatz wird oft eine akustische Signalisierung benötigt, sei es, um eine Rückmeldung für eine Berührung auf dem Touch-Display zu bekommen (*Klick*) oder z.B. um die Aufmerksamkeit des Benutzers zu erlangen. Hier sind einfache hardwareunabhängige Sound-Funktionen verfügbar, die die Ausgabe von Tönen erlauben. (Voraussetzung für die Sound-Ausgabe ist, daß das verwendete LCM über einen Piezo-Lautsprecher verfügt).

Übersicht Sound-Funktionen:

SOUND_SET_VOLUME: Stellt neuen Lautstärkewert für den Lautsprecher ein.
SOUND_GET_VOLUME: Liest die aktuelle Lautstärkeeinstellung für den Lautsprecher.
SOUND_MAKE: Erzeugt einen Ton auf dem Piezolautsprecher.
SOUND_MAKE_ASYNC: Erzeugt einen Ton auf dem Piezolautsprecher.
SOUND_ON: Schaltet einen Ton ein.

SOUND_OFF: Schaltet einen Ton auf aus.
SOUND_CLICK: Erzeugt Tastatur–Klick.

3.11 Event-Verarbeitung

Ziel dieses Moduls ist es, die Abfrage von *touchable objects*, also GUI-Elemente, die durch eine Touch-Eingabe bedient werden, einheitlich abzufragen und zu bearbeiten (siehe Funktionsreferenz). Hierzu wird beim Starten der Event-Verarbeitung eine Liste angelegt (*event queue*), in der die einzelnen Ereignisse, die durch die Touch-Objekte ausgelöst wurden, in der Reihenfolge ihres Auftretens abgespeichert werden. Dies geschieht beim GUI-Interpreter intern ohne Zutun des ansteuernden Masters. Die Auswerteroutine holt dann die einzelnen Ereignisse zur Weiterverarbeitung dort ab. Bei der Gelegenheit wird ein Ereignis auch gerade wieder aus der Liste gelöscht. Werden die Events schneller erzeugt als abgearbeitet, so ist irgendwann die Kapazität der Liste erschöpft und weitere Ereignisse werden verworfen. Man sollte also die Größe der Event-Queue bei der Initialisierung entsprechend wählen und regelmäßig Ereignisse abholen. Im Beispiel-Master wird diese Möglichkeit der Ergeinisverarbeitung demonstriert.

Übersicht Event-Verarbeitungs-Funktionen:

EVENTS_ACTIVATE: Aktiviert das Event-Handling für Buttons und für andere Eingabeelemente.
EVENTS_IS_ACTIVE: Überprüft, ob die Ereignisverarbeitung aktiviert ist.
EVENTS_DEACTIVATE: Schaltet das Event-Handling aus.
EVENTS_GET_NEXT_EVENT: Liest ein Ereignis aus der Event-Liste und löscht es dort.

3.12 Fortschrittsbalken

Die Dateien “bar_indicator.h/.c” stellen Funktionen zur Darstellung und Kontrolle von Fortschrittsbalken zur Verfügung. Die Balken werden von den Standardrahmen aus “display.h/.c” umgeben und können sowohl horizontal als auch vertikal ausgerichtet sein. Der Start- und Endwert sind frei wählbar. Hierbei besteht die Möglichkeit, den aktuellen Wert zusammen mit einer physikalischen Einheit (z.B. Temperatur) innerhalb des Balkens anzuzeigen. Außerdem kann ab einem bestimmten Schwellwert der Balken die Farbe ändern, um z.B. eine Gefahrensituation zu signalisieren.

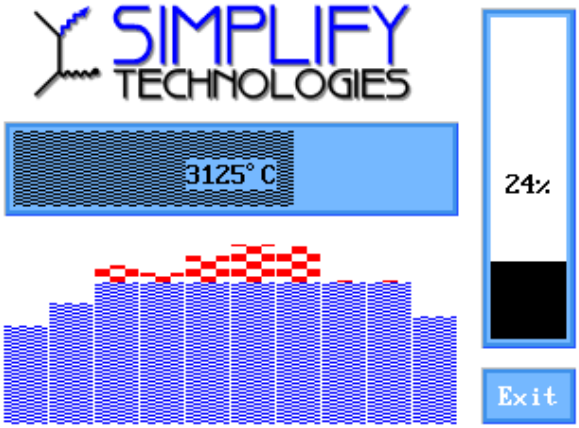


Abbildung 8: Beispiel für Fortschrittsbalken

Übersicht Bar_Indicator-Funktionen:

BARIND_STYLE_INIT: Holt Speicher und initialisiert einen Stil für die Fortschrittsanzeige.

BARIND_STYLE_CLONE: Holt Speicher für einen neuen Bar–Indicator–Stil und füllt diesen mit den Werten aus einem schon vorhandenen Stil.

BARIND_STYLE_FREE: Gibt den Speicher eines Bar–Indicator–Stils wieder frei.

BARIND_STYLE_SET_FRAME: Setzt den Rahmenstil für den Stil des Fortschrittbalkens.

BARIND_STYLE_SET_FONT: Setzt Schriftart, Textstil und Farbe für die Textmarkierung des Fortschrittbalkens.

BARIND_STYLE_SET_BAR_APPEARANCE: Setzt das Aussehen des Fortschrittbalkens (Farben und Zeichenmuster).

BARIND_DEFINE: Erzeugt einen Forschrittsbalken.

BARIND_UNDEFINE: Deaktiviert einen Fortschrittsbalken und gibt den belegten Speicher wieder frei.

BARIND_ACTIVATE: Zeigt den Fortschrittsbalken auf dem Display an.

BARIND_DEACTIVATE: Löscht einen Fortschrittsbalken vom Display.

BARIND_SET: Setzt den Fortschrittsbalken auf einen neuen Wert.

3.13 Clipping

Die Funktionen aus “clip.h/.c” testen, ob die Linien ein vorgegebenes Rechteck scheiden und passen die Anfangs- und Endpunkte so an, daß diese auf dem Rand dieses Rechtecks liegen.

Übersicht Clipping-Funktionen:

CLIP_POINT: Testet, ob einen Punkt in einem rechteckigen Bereich liegt oder nicht.
CLIP_LINE: Eine Linie wird auf eine Schnittmenge mit einem rechteckigen Bereich getestet. Die Punkte der Linie werden u.U. so angepaßt, daß sie auf dem Rand des rechteckigen Bereichs liegen.
CLIP_TEXT: Zeichnet einen Text innerhalb eines rechteckigen Bereichs, der vom Text nicht überschritten wird.

3.14 Eingabezeile

Dieses Modul “inputline.h/.c” unterstützt den Benutzer bei der Programmierung von Eingabezeilen, wie sie z.B. mit Touch-Tastaturen häufig zum Einsatz kommen. Die zur Verfügung gestellten Funktionen (siehe Funktionsreferenz) übernehmen die grafische Repräsentation der Eingabezeile, wie z.B. bewegen des Textcursors, scrollen des eingegebenen Strings, löschen/einfügen von Zeichen usw.. Der String in der Zeile darf größer sein als die Zeile selber und wird bei Bedarf gescrollt. Die Eingabezeile benutzt die Standardrahmen aus “display.h/.c”.

Übersicht Inputline-Funktionen:

INPUTLINE_STYLE_INIT: Richtet die Struktur des Eingabezeilenstils ein und holt entsprechend Speicher.
INPUTLINE_STYLE_FREE: Gibt den Speicher eines Eingabezeilenstils wieder frei.
INPUTLINE_STYLE_SET_FRAME: Setzt die Eigenschaften des Rahmens eines Eingabezeilenstils.
INPUTLINE_STYLE_SET_COLORS: Setzt die Farben für den Eingabezeilenstil.
INPUTLINE_STYLE_SET_FONT: Setzt den Zeichensatz und den Textstil für einen Eingabezeilenstil.
INPUTLINE_DEFINE: Definiert eine neue Eingabezeile, indem zunächst Speicher geholt und die Datenstruktur initialisiert wird. Anschließend wird die Eingabezeile auf dem Display dargestellt.
INPUTLINE_UNDEFINE: Löscht eine Eingabezeile vom Display und gibt den belegten Speicher frei.

INPUTLINE_DRAW: Zeichnet die Eingabezeile.
INPUTLINE_BLINK: Lässt die Eingabezeile blinken.
INPUTLINE_SET_CURSOR: Setzt den Cursor auf ein neues Zeichen in der Eingabezeile.
INPUTLINE_MOVE_CURSOR_RIGHT: Verschiebt den Cursor um ein Zeichen nach rechts.
INPUTLINE_MOVE_CURSOR_LEFT: Verschiebt den Cursor um ein Zeichen nach links.
INPUTLINE_MOVE_CURSOR_LEFT_BORDER: Bewegt den Cursor zum linken Rand der Eingabezeile.
INPUTLINE_MOVE_CURSOR_RIGHT_BORDER: Bewegt den Cursor zum rechten Rand der Eingabezeile.
INPUTLINE_DELETE_CHAR: Löscht ein Zeichen in der Zeichenkette an der aktuellen Cursorposition.
INPUTLINE_DELETE_STRING: Löscht den kompletten Inhalt der Eingabezeile.
INPUTLINE_INSERT_CHAR: Fügt ein Zeichen in die Eingabezeile an der aktuellen Cursorposition ein.
INPUTLINE_INSERT_STRING: Fügt einen String in die Eingabezeile an der aktuellen Cursorposition ein. Der String muß mit Null abgeschlossen sein und darf maximal 98 Zeichen lang sein.
INPUTLINE_SET_STRING: Setzt einen neuen Text in der Eingabezeile.
INPUTLINE_GET_CURSOR_POSITION: Holt die aktuelle Cursorposition.
INPUTLINE_GET_CHAR: Holt das Zeichen im String an der angegebenen Cursorposition.
INPUTLINE_READ_STRING: Nur-Lesezugriff auf den Textspeicher der Eingabezeile. Hierzu wird der Textspeicher kopiert.

3.15 Menüs

Es gibt drei Unterarten von Menüs:

- *Popup*-Menüs: Ein Menü, welches programmgesteuert auf dem Display erscheint und nach Auswahl eines Eintrags wieder verschwindet (siehe “popup_menu.h/.c”). Das Menü wird ebenfalls wieder ausgeblendet, wenn ein Touch-Kontakt außerhalb des Menüs erfolgt.

- *Single*-Menüs: Eine Kombination von einem Popup-Menü mit einem Button. Der Button dient dazu, das Popup-Menü erscheinen zu lassen. Dieses verschwindet, sobald ein Menüeintrag ausgewählt wurde bzw. der Aktivierungs-Button wieder gedrückt wurde (siehe “menu.h/.c”). Auch hier wird das Menü durch einen Touch-Kontakt außerhalb ebenfalls ausgeblendet.

- Menü-*Bars*: Eine Kombination von einigen Single-Menüs, wobei die Aktivierungs-Buttons alle nebeneinander in einer Zeile liegen und die einzelnen Menüs direkt unter oder über dem Aktivierungs-Button aufklappen (siehe "menu.h/.c").

Die Menüs benutzen die Standardrahmen aus "display.h/.c". Bei den Menüs handelt es sich um *touchable objects*, welche durch die selben Funktionen wie die Buttons abgefragt werden können (siehe `touch_process_objects()` in der Funktionsreferenz.

Übersicht Popup-Menü-Funktionen:

POPUP_STYLE_INIT: Holt Speicher für eine Menüstildatenstruktur und initialisiert diese.
POPUP_STYLE_FREE: Gibt den Speicher eines Menüstils wieder frei.
POPUP_STYLE_SET_FRAME: Setzt den Rahmenstil des Menüstils.
POPUP_STYLE_SET_SEPARATOR_HEIGHT: Setzt die Höhe der Linie (*Separator*), die zwischen zwei Menüeinträgen gesetzt werden kann.
POPUP_STYLE_SET_FONT: Setzt den Zeichensatz und das Aussehen des Textes für einen Menüstil.
POPUP_DEFINE: Legt ein Popup–Menü im Speicher an und initialisiert es.
POPUP_ADD_ENTRY: Fügt einen neuen Eintrag zu dem Popup–Menü hinzu. Das Menü muß dazu deaktiviert werden.
POPUP_REMOVE_ENTRY: Entfernt einen Eintrag aus dem Popup–Menü. Das Menü muß dazu deaktiviert werden.
POPUP_ACTIVATE: Aktiviert das Popup–Menü und zeigt es an.
POPUP_DEACTIVATE: Löscht das Menü von der Anzeige.
POPUP_GET_DIMENSIONS: Holt die Größe des Popup–Menüs.
POPUP_ENTRY_WAS_PRESSED: Überprüft, ob ein bestimmter Eintrag eines Popup–Menüs ausgewählt wurde.
POPUP_SET_FLAGS: Setzt die Flags des Popup–Menüs.
POPUP_GET_FLAGS: Holt die Flags des Popup–Menüs.
POPUP_SET_ENTRY_FLAGS: Setzt die Flags für einen Menüeintrag.
POPUP_GET_ENTRY_FLAGS: Holt die Flags eines Menüeintrag.
POPUP_UNDEFINE: Löscht ein Popup–Menü vom Display und gibt den belegten Speicher wieder frei.

Übersicht Menü-Funktionen:

MENU_DEFINE_SINGLE: Initialisiert ein Single–Menü und holt den benötigten Speicher.

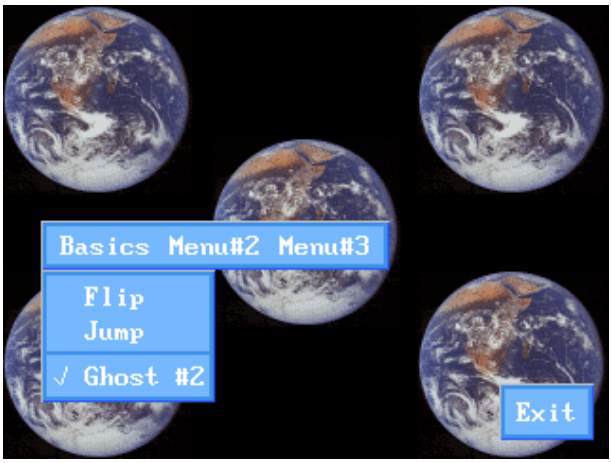


Abbildung 9: Beispiel für Menü

MENU_UNDEFINE_SINGLE: Löscht das Single–Menü vom Display und gibt den belegten Speicher wieder frei.
MENU_ACTIVATE_SINGLE: Zeigt den Aktivierungs–Button an und fügt das Single–Menü in die Liste der Touch–Objekte ein.
MENU_DEACTIVATE_SINGLE: Löscht ein Single–Menü von der Anzeige.
MENU_SINGLE_ENTRY_WAS_PRESSED: Überprüft, ob ein bestimmter Eintrag eines Single–Menüs ausgewählt wurde.
MENU_DEFINE_BAR: Initialisiert eine Menüzeile und holt Speicher für die Datenstrukturen.
MENU_ADD_BAR_POPUP: Fügt ein Popup–Menü zu einer Menüzeile hinzu.
MENU_ACTIVATE_BAR: Zeigt die Menüzeile an.
MENU_DEACTIVATE_BAR: Entfernt die Menüzeile von der Anzeige.
MENU_UNDEFINE_BAR: Die Menüzeile wird vom Display entfernt und der belegte Speicher wird freigegeben.
MENU_BAR_ENTRY_WAS_PRESSED: Überprüft, ob ein bestimmter Eintrag der Menüzeile gedrückt wurde.
MENU_BAR_SET_FLAGS: Setzt die Flags der Menüzeile.
MENU_BAR_GET_FLAGS: Holt die Flags einer Menüzeil.

3.16 Sieben-Segment-Anzeige

Das Modul “seven_segments.h/.c” stellt Funktionen zur Darstellung von Sieben-Segment-Zahlen (mit Dezimalpunkt und “E”) zur Verfügung (siehe hierzu auch die

Funktionsreferenz. Hierbei sind die Anzeigen frei skalierbar und auf eine schnelle Ausgabe hin optimiert. Die darzustellenden Zahlen sind als String zu übergeben; alle nicht zulässigen Zeichen in diesem String werden ignoriert.

Übersicht Funktionen der Sieben-Segment-Anzeige:

SEG7_INIT: Initialisiert eine Info–Datenstruktur für die Sieben–Segment–Anzeige und holt Speicher dafür.
SEG7_FREE: Gibt den Speicher frei, der durch die Info-Struktur der 7-Segment-Anzeige verbraucht wurde.
SEG7_SET_COLORS: Setzt die Farben der Zahlen.
SEG7_GET_WIDTH: Holt die Breite der gesamten Sieben–Segment–Anzeige in Pixel.
SEG7_DEFINE: Stellt eine Sieben–Segement–Anzeige auf dem Display dar.
SEG7_UPDATE: Erneuert den Inhalt der Sieben–Segment–Anzeige.

3.17 Schieberegler

Das Modul “slider.h/.c” enthält Funktionen zur Realisierung von Schiebereglern (siehe auch die Funktionsreferenz). Durch Berühren des Touch-Glases kann der Schieberegler ähnlich wie bei anderen Benutzeroberflächen verschoben werden. Der dabei zurückgelieferte Wert liegt zwischen den vorher definierten Grenzen, die wiederum unabhängig von der physikalischen Größe des Reglers sind. Der Schieberegler benutzt die Standardrahmen aus “display.h/.c”. Bei dem Schieberegler handelt es sich um ein *touchable object*, welches durch die selben Funktionen wie die Buttons abgefragt werden kann (siehe `touch_process_objects()` in der Funktionsreferenz). Eine einfache und einheitliche Methode zum Abfragen von *touchable objects* wird in Abschnitt 3.11 vorgestellt.

Übersicht Slider-Funktionen:

SLIDER_STYLE_INIT: Initialisiert eine Datenstruktur für einen Schieberegler und belegt den Speicher.
SLIDER_STYLE_CLONE: Kopiert einen Schiebereglerstil und legt Speicher für die Kopie an.
SLIDER_STYLE_FREE: Gibt den Speicher eines Schiebereglerstils wieder frei.
SLIDER_STYLE_SET_FRAMESTYLE: Setzt den Rahmenstil für den Stil des Schiebereglers.
SLIDER_STYLE_SET_COLOR: Setzt die Farben eines Schiebereglerstils.
SLIDER_DEFINE: Definiert einen neuen Schieberegler, indem Speicher für die Datenstruktur belegt wird und diese dann initialisiert wird.

Abbildung 10: Beispiel für Sieben-Segmen-Anzeige

SLIDER_SET_POSITION: Setzt den Schieber des Schiebereglers auf eine neue Position.

SLIDER_GET_POSITION: Holt die Position des Schiebers des Schiebereglers.

SLIDER_REDEFINE: Setzt einen neuen Bereich für den Schieberegler. Setzt die Größe und die Position des Schiebers neu.

SLIDER_UNDEFINE: Löscht einen Schieberegler und den von ihm belegten Speicher.

SLIDER_ACTICVATE: Aktiviert einen Schieberegler und stellt ihn auf dem Display dar.

SLIDER_DEACTIVATE: Deaktiviert einen Schieberegler und löscht ihn vom Display.

3.18 Auswahllisten

Das Modul “selection_list.h/.c” enthält Funktionen zur Realisierung von Auswahlisten (siehe auch die Funktionsreferenz). Durch Touch-Bedienung kann die Liste nach oben und unten gescrollt werden, dabei wählt der Benutzer einzelne oder mehrere Einräge aus. Jeder Eintrag kann aus einem Piktogramm, einem links- und einem rechtsbündigen Text bestehen. Es gibt auch eine Variante der Auswahlliste, die nur zur Darstellung von Informationen dient.

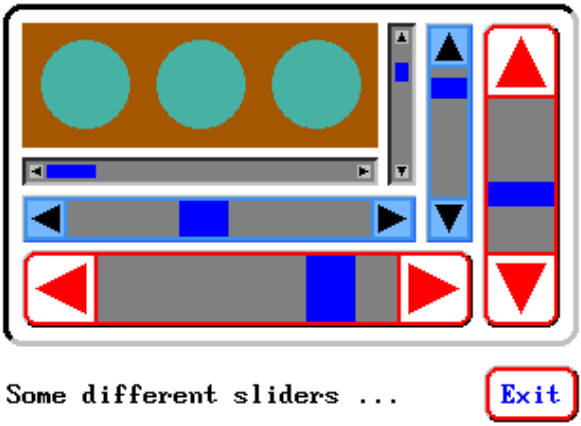


Abbildung 11: Beispiel für Schieberegler

Übersicht Selection-List-Funktionen:

SL_ITEM_STYLE_INIT: Initialisiert eine Stil für die Einträge einer Auswahlliste.
SL_ITEM_STYLE_CLONE: Holt Speicher für einen neuen Stil für die Einträge der Auswahlliste, wobei der neue Stil die Kopie eines alten ist.
SL_ITEM_STYLE_FREE: Gibt den Speicher eines Stils für die Einträge der Auswahlliste wieder frei.
SL_ITEM_STYLE_SET_FONT: Setzt den Zeichensatz, den Textstil und die Farbe für einen Texteintragsstil der Auswahlliste.
SL_ITEM_STYLE_SET_COLORS: Setzt die verbleibende(n) Farbe(n) des Stils der Einträge der Auswahlliste.
SL_STYLE_INIT: Initialisiert den Stil der Auswahlliste.
SL_STYLE_CLONE: Kopiert einen Stil der Auswahlliste und legt Speicher für die Kopie an.
SL_STYLE_FREE: Gibt den Speicher eines Auswahllistenstils wieder frei.
SL_STYLE_SET_SLIDER: Setzt die Parameter des Schiebereglers, der in der Auswahlliste benutzt wird.
SL_STYLE_SET_GAP: Setzt den Abstand zwischen den Icons und dem Text eines Eintrags und zwischen den Listeneinträgen und dem Schieberegler.
SELECTION_LIST_DEFINE: Definiert eine neue Auswahlliste.
SELECTION_LIST_UNDEFINE: Löscht eine Auswahlliste und gibt den benutzten Speicher frei.

Abbildung 12: Beispiel für Auswahlliste

SELECTION_LIST_ACTIVATE: Aktiviert eine Auswahlliste und stellt sie auf dem Bildschirm dar.
SELECTION_LIST_DEACTIVATE: Deaktiviert eine Auswahlliste und löscht diese vom Bildschirm.
SELECTION_LIST_ADD_ITEM: Fügt einen Eintrag zu einer Auswahlliste hinzu.
SELECTION_LIST_REMOVE_ITEM: Entfernt einen Eintrag aus der Auswahlliste.
SELECTION_LIST_CHANGE_ITEM_LABELS: Ändert die Texte eines Listeneintrags.
SELECTION_LIST_COUNT_ITEMS: Zählt die Einträge in der Auswahlliste.
SELECTION_LIST_SET_POSITION: Setzt die Auswahlliste auf einen bestimmten Eintrag.
SELECTION_LIST_SCROLL_UP: Scrollt die Auswahlliste um eine Zeile nach oben.
SELECTION_LIST_SCROLL_DOWN: Scrollt die Auswahlliste um eine Zeile nach unten.
SELECTION_LIST_ITEM_SET_STATUS: Setzt den Status eines Eintrags der Auswahlliste.
SELECTION_LIST_GET_LAST_EVENT_ITEM_STATUS: Holt den Status des Eintrags, der den letzten Event ausgelöst hat.
SELECTION_LIST_GET_SELECT_ITEM: Holt den Index und den Status des zuletzt selektierten/deselektierten Eintrags.

SELECTION_LIST_GET_FOCUS_ITEM: Holt den Index und den Status des zuletzt fokusierten Eintrags.

3.19 Logger-Framework

Die Module “logger_data.h/.c” und “logger_disp.h/.c” enthalten Funktionen zum Verwalten von Datensätzen und zur Darstellung von Diagrammen. Eine detailierte Beschreibung und eine Befehlsreferenz befindet sich in der separaten Anleitung zum Logger-Framework.

Übersicht Logger-Framework-Funktionen:

DL_DATASET_ATTACH: Ordnet einen Datensatz einem angeschlossenen Gerät zu (in der Regel ein LCM-Modul mit GUI-Interpreter Software), um diesen dort anzuzeigen.
DL_DATASET_SYNC_CURRENT_INDEX: Gleicht den aktuellen Index zum Zugriff auf den Datensatz zwischen Master und GUI-Interpreter ab.
DL_SET_RAW_DATA: Zugriffsfunktion: Schreibt Rohdaten in den Datensatz.
DL_SET_NEXT_RAW_DATA: Zugriffsfunktion: Fügt Rohdaten an die nächste Position im Datensatz ein.
DL_AXIS_STYLE_INIT: Holt Speicher für einen Achsenstil und initialisiert diesen.
DL_AXIS_STYLE_FREE: Entfernt einen Achsenstil aus dem System und gibt den Speicher wieder frei.
DL_AXIS_INIT: Holt Speicher für eine Achse und initialisiert die Achsenstruktur.
DL_AXIS_SET_PARAMETERS: Setzt die Parameter einer Achse.
DL_AXIS_FREE: Gibt den Speicher einer Achse wieder frei.
DL_AXIS_CONVERT: Konvertiert eine Pixel–Position in einen physikalischen Wert in Einheiten der Achse.
DL_AXIS_CONVERT_REAL: Konvertiert einen Wert in physikalischen Einheiten auf einer Achse zu der Position in Pixel–Koordinaten.
DL_AXIS_ITEM_ADD: Fügt ein Axis–Item zur Menge der Axis-Items einer Achse hinzu.
DL_AXIS_ITEMS_REMOVE: Entfernt alle Axis–Items von einer Achse.
DL_AXIS_ITEMS_AUTO: Automatische Generierung der Achsenbeschriftung auf einem Bereich der Achse.
DL_DIAGRAM_STYLE_INIT: Holt Speicher für einen Diagrammstil und initialisiert diesen.
DL_DIAGRAM_STYLE_FREE: Entfernt einen Diagrammstil.
DL_DIAGRAM_INIT: Holt Speicher für ein Diagramm und initialisiert diesen.
DL_DIAGRAM_SET_PARAMETERS: Setzt die Parameter des Diagramms.
DL_DIAGRAM_FREE: Entfernt ein Diagramm aus dem Speicher.

DL_DIAGRAM_CONTROL_CURSOR: Kontrolliert die Cursor-Linien Position und deren Aktivierungszustand.
DL_DIAGRAM_DRAW: Zeichnet das Diagramm.
DL_DIAGRAM_DRAW_LINE: Zeichnet eine Linie im Diagramm.
DL_PLOT_STYLE_INIT: Holt Speicher für einen Plotstil und initialisiert diesen.
DL_PLOT_STYLE_FREE: Entfernt einen Plotstil aus dem Speicher.
DL_PLOT_INIT: Holt Speicher für einen Plot und initialisiert diesen.
DL_PLOT_FREE: Entfernt einen Plot aus dem Speicher.
DL_PLOT_SYNC_RAW_RANGE: Synchronisiert einen Bereich von Rohdaten aller benutzer Datensätze eines Plots mit einem externen Gerät, welches dann die eigentliche Visualisierung der Daten übernimmt.
DL_PLOT_DRAW: Zeichnet einen Plot.
DL_PLOT_EXTEND_RIGHT: Erweitert einen existierenden Plot zur rechten Seite hin, indem neue Datenpunkte zusätzlich gezeichnet werden.
DL_DIAGRAM_ROLL_LEFT: Verschiebt den Diagramminhalt nach links.
DL_ADJUST_PLOT_AFTER_ROLLING: Nach der Verschiebung eines Plots müssen für diesen interne Berechnungen neu durchgeführt werden, um die Darstellungsgeschwindigkeit zu verbessern.
DL_ADD_PLOT_POINT: Zeichnet den aktuellen Datenpunkt (= letzter hinzugefügter Punkt).

3.20 Encoder

Das Modul “encoder.h/.c” enthält Funktionen zur Konfiguration und Abfrage eines optionalen optischen Encoders (siehe auch Funktionsreferenz).

Übersicht Encoder-Funktionen:

ENCODER_SET_CHARACTERISTICS: Setzt das Verhalten des Encoders und dessen Schalters.
ENCODER_GET_DIFFERENCE: Holt die relative Differenz (Distanz) des Encoder–Werts im Vergleich zum letzten Aufruf der Funktion.

3.21 Animationen

Das Modul “guii_animation.h/.c” enthält Funktionen zur Erstellung von Animationen, die im Hintergrund ohne weiteres zutun des Benutzers ablaufen können (siehe auch Funktionsreferenz).

Übersicht Animationsfunktionen:

ANIM_INIT: Initialisiert ein Animationsobjekt und holt Speicher dafür.

ANIM_ADD_FRAME: Fügt ein neues Bild zum Animationobjekt hinzu.
ANIM_FREE: Gibt den Speicher des Animationobjekts wieder frei.
ANIM_ACTIVATE: Aktiviert ein Animationsobjekt und stellt dieses dar.
ANIM_DEACTIVATE: Deaktiviert ein Animationsobjekt und löscht es vom Bildschirm.
ANIM_STOP: Stoppt eine Animation.
ANIM_START: Startet eine Animation.
ANIM_CONT: Lässt eine gestoppte Animation weiterlaufen.

3.22 Gerätespezifische Funktionen

Es gibt einige Funktionen, die für Einstellungen und Verwaltung des LCM-Modules zuständig sind (z. B. Schalten in den Service-Mode für Software-Updates).

Übersicht Interpreter-Funktionen für die modulspezifische Funktionen (z. B. Objektverwaltung):

LCM_UPLOAD_TO_RAM: Startet das Laden eines binären Objekts in das LCM.
LCM_INIT_FLASH: Weist den GUI-Interpreter an, den optionalen externen Flash-Speicher des LCM zu löschen und Vorbereitungen zur Speicherung von Daten im Flash zu treffen.
LCM_INIT_FLASH_NO_OUTPUT: Weist den GUI-Interpreter an, den optionalen externen Flash-Speicher des LCM zu löschen und Vorbereitungen zur Speicherung von Daten im Flash zu treffen.
LCM_UPLOAD_TO_FLASH: Startet die Übertragung von binäre Daten zum GUI-Interpreter, der diese dann im Flash-Speicher der Hardware ablegt.
LCM_FINALIZE_FLASH: Schließt den Flash-Vorgang ab und setzt die Hardware auf der der GUI-Interpreter läuft zurück.
LCM_FINALIZE_FLASH_NO_RESET: Schließt den Flash-Vorgang ab, setzt aber die Hardware nicht zurück.
LCM_FLASH_DUMP: Bereitet die Übertragung des Flash–Speicherinhalts auf den Master vor.
LCM_SWITCH_FLASH_PAGE: Schaltet zu einer anderen Flash–Page, auf die dann schreibend zugegriffen werden kann.
LCM_FLASH_PAGE_INFO: Holt Informationen über eine Flash–Page ein.
LCM_FLASH_PAGES_CHECK_IDS: Überprüft alle Flash–Pages nach mehrfach vergebenen Objekt–IDs.
LCM_KILL_BINARY_OBJECT: Löscht ein binäres Objekt (Font, Füllmuster, BMP-Bild o.ä.) aus dem RAM-Speicher.
LCM_ACCESS_BINARY_OBJECT: Liest Daten aus einem binären Objekt.

LCM_SLEEP: Versetzt das LCM in den Standby-Modus (falls die Hardware das zulässt), so daß der Stromverbrauch erheblich reduziert wird.
LCM_DOWNLOAD_DISPLAY: Bereitet die Übertragung des Display–Inhalts auf den Master vor.
LCM_DOWNLOAD_DATA: Überträgt Daten aus dem angegebenen Transfer–Puffer zum Master.
LCM_TERMINATE_DOWNLOAD: Mit diesem Kommando kann ein Transfer-Puffer gelöscht und so die Datenübertragung beended werden.
LCM_TEMPERATURE: Bestimmt die Temperatur auf der Platine des LCM.
LCM_CONTRAST_OFFSET: Modifiziert die Kontrastspannung um einen zusätzlichen Offset zum temperaturgeregelten Standardwert der Kontrastspannung.
LCM_SERVICE_MODE: Schaltet das LCM in den Service–Mode, um z.B. eine neue Version des GUI-Interpreters in das Gerät zu übertragen.
LCM_MACRO_REC: Startet die Aufzeichnung eines Makros.
LCM_MACRO_STOP: Schließt eine Makroaufzeichnung ab.
LCM_MACRO_PLAY: Spielt ein vorher mit LCM_MACRO_REC und LCM_MACRO_STOP aufgenommenes Makro wieder ab.
LCM_MACRO_KILL: Löscht ein zuvor aufgezeichnetes Makro wieder aus dem Speicher.
LCM_MACRO_TO_FLASH: Überträgt ein ins RAM aufgezeichnetes Makro in einen (optionalen) Flashspeicher.
LCM_EEPROM_READ_BYTE: Liest ein Byte aus dem EEPROM–Speicher (optional).
LCM_EEPROM_WRITE_BYTE: Schreibt ein Byte in den EEPROM–Speicher (optional).

Eine Übersicht über die Interpreter-Funktionen für die Kommunikation finden Sie bei der Beschreibung der Kommunikationsprotokolle in Abschnitt 2.5 auf Seite 17.

4 Programmierung

In den folgenden Abschnitten werden einige häufige Anwendungsfälle besprochen, wobei auf die Funktionen der bereitgestellten Master–Bibliothek (in C) bezug genommen wird.

4.1 Einschaltverhalten des GUI–Interpreters

Mit einem speziellen binären Objekt, welches im Flashspeicher eines LCM–Modules abgelegt wird kann das Einschaltverhalten des GUI–Interpreters konfiguriert werden. Dieses Konfigurationsobjekt wird mit dem Befehl `LCMUploadToFlash()` (siehe Abschnitt 4.3.1 auf Seite 50) übertragen. Hierzu gibt es im beiliegenden Beispiel–Master eine Funktion, die diese Aufgabe übernimmt (`LCMConfigToFlash()` in den Dateien "lcm.h/c "). Das in der Datei "user_configuration.h" definierte Konfigurationsobjekt (siehe Tabelle 4) besteht nur aus einer Folge von 32 Bit–Zahlen. Am Anfang kommt eine Kennung und die Anzahl der Parameter, danach folgen die eigentlichen Einstellungsparameter.
Bei der Diashow kann man auch manuell die Bilder weiterschalten (nur bei Geräten mit Touch), indem man auf das rechte drittel bzw. linke drittel des Displays drückt, um ein Bild vor bzw. zurück zu schalten. Drückt man auf den mittleren Bereich des Displays, so pausiert die Diashow. Im Diashow–Modus reagiert das LCM–Modul nicht auf Interpreterbefehle. In den Interpretermodus geht das Gerät zurück, nachdem man dreimal den Buchstaben "Q" mit 9600 Baud an die serielle Schnittstelle gesendet hat. Anschließend kann nach dem Autobauding z.B. der Flash–Speicher gelöscht werden und das Gerät befindet sich wieder im Auslieferungszustand.

4.2 Farbmodelle und deren Handhabung

Mit der Verwendung mehrerer Farben bekommt nun im Gegensatz zu monochromen Displays die Farbe des Hintergrundes eine besondere Bedeutung. Diese kann nun unter verschiedenen Farben gewählt werden und erweitert so die Gestaltungsmöglichkeiten der Benutzeroberfläche.
Achtung: Bei älteren Version der GUI–Interpreter Software ($< 4.0.0$) wurden Farbwerte durch 1 Byte übertragen. Dies hat sich nun auf 4 Bytes erhöht, um auch Geräte mit einer größeren Farbtiefe zu unterstützten. Die Master–Implementation muß dies berücksichtigen, wenn auch ältere LCM–Module eingesetzt werden sollen.

4.2.1 Farbtiefe von 8 Bit

Die farbfähigen LCD–Module mit einer Farbtiefe von 8 Bit können 256 Farben darstellen. Diese begrenzte Palette führt zwar zu gewissen Einschränkungen in der Be-

32 Bit–Wert (Default)	Beschreibung
`0xC7779DD9`	Kennung des Konfigurationsobjekts (*magic*)
`0x00000003`	Anzahl der Parameter
`0x00000001`	Flags Bit 1: Einschaltmeldung des GUI–Interpreters aktiv. Bit 2: Aktivieren der Diashow von Bildern im Flash–Speicher. Die ID des ersten Bildes der Diashow wird durch die ID des Bildes, welches beim Systemstart gezeigt werden kann definiert (siehe unten). Die weitern Bilder der Diashow müssen aufsteigende Nummern haben. Bit 3 - 32: reserviert (diese Bits nicht setzen)
`0x00000064`	Starthelligkeit in Prozent
`0x00000000`	ID eines BMP–Bildes im Flash–Speicher, das beim Systemstart angezeigt wird (Defaultwert 0: es wird nur ein leerer Bildschirm angezeigt). Hierzu muß ausserdem die Einschaltmeldung (s.o.) ausgeschaltet werden.
`0x00000003`	Verzögerung zwischen den Bildern der Diashow in Sekunden.
`0x00000001`	Hintergrundfarbe der Diashow.

Tabelle 4: Beispiel eines Konfigurationsobjekts mit Default–Einstellungen.

nutzung der Farben, diese Einschränkungen stören jedoch in der Praxis kaum (ein Farbfoto läßt sich auch so ordentlich darstellen).
Um eine ansprechende Darstellung einer Fotografie zu erzielen, ist es nötig, den begrenzten Farbraum auf dieses Bild hin anzupassen. Schwierig wird dies, wenn zusätzlich noch andere Fotos oder andersfarbige Elemente gleichzeitig auf dem Display dargestellt werden sollen. Es wäre insbesondere störend oder inakzeptabel, wenn die vorher sorgfältig entworfenen Bedienelemente durch eine Änderung einer Farbpalette ihre Farben ändern würden. Daher wird (wie auch in der GUI–Bibliothek) wie folgt verfahren:

- Es werden zunächst feste *Systemfarben* definiert, die unabhängig von darge-

stellten Bildern zur Verfügung stehen und sich nicht ändern. Dies sind die 16 Standardfarben der VGA–Palette: `BLACK`, `MAROON`, `GREEN`, `OLIVE`, `NAVY`, `PURPLE`, `TEAL`, `GRAY`, `SILVER`, `RED`, `LIME`, `YELLOW`, `BLUE`, `FUCHSIA`, `AQUA`, `WHITE`. Diese können bei Bedarf leicht an abweichende Anforderungen angepaßt werden (`disp_set_palette_entry()`). Diese Farben können in den Funktionen der Bibliothek auch so mit ihrem Namen angesprochen werden.

- Abhängig von der Anzahl der auf dem Gerät darstellbaren Farben können die über die 16 Standardfarben hinausgehenden Farben zur Darstellung von Bildern genutzt werden. Bei verfügbaren 256 Farben sind so 240 für die Darstellung von Bildern verfügbar. Bei Bedarf ist es auch möglich, die Anzahl der Farben, die für BMP–Bilder reserviert werden mit dem Befehl `disp_reserve_bmp_colors()` einzustellen, so daß z.B. für Bilder 224 Farben benutzt werden und die restlichen 32 unbeeinflußt bleiben und als ”Standardfarben” genutzt werden können.

4.2.2 BMP-Bilder bei einer Farbtiefe von 8 Bit

Auf Geräten mit einer Farbtiefe von 8 Bit werden zur Zeit unkomprimierte Monochrom–BMP–Bilder sowie 16– und 256–Farben–BMP–Bilder (unkomprimiert bzw. RLE–komprimiert) unterstützt. Beim Anzeigen eines farbigen BMP–Bildes werden die Farben des Bildes so in die Farbpalette abgelegt, daß die Paletteneinträge, die nicht für die BMP-Bilder reserviert sind auch nicht geändert werden. Um eine korrekte Bilddarstellung zu erzielen ist auf folgendes zu achten

- Das Bild muß auf die zur Verfügung stehende Anzahl Farben reduziert werden. Dies kann mit einem beliebigen Grafikprogramm durchgeführt werden.
- Wenn mehrere Bilder gleichzeitig angezeigt werden sollen, ist es vorteilhaft, wenn diese alle die selbe Palette haben, so daß nicht soviele Einträge in der Farbtabelle verbraucht werden. Auch dies kann in einem Grafikprogramm[2] sichergestellt werden. Vorgeschlagene Arbeitsschritte sind:

  1. Kopieren Sie alle Bilder in ein großes neues Echtfarbenbild.
  2. Reduzieren Sie die Farben dieses neuen Bildes auf die zulässige Anzahl.
  3. Separieren Sie die ursprünglichen Bilder nun aus dem Gesamtbild. Die einzelnen Bilder verwenden nun denselben Farbraum (Farbpalette) und

[2] Prinzipiell wäre es auch möglich gewesen, auf die externen Schritte zur Bearbeitung der Bilder zu verzichten und die Umrechnung der Bilder innerhalb des LCM–Moduls vorzunehmen. Da dies jedoch einen erheblichen Mehraufwand in Rechenzeit und Speicherbedarf bedeuten würde, wurde darauf verzichtet und die externe Bildbearbeitung vorgesehen.

können gleichzeitig ohne Farbstörungen auf dem LCD angezeigt werden.

Wird ein neues Bild geladen, so wird zunächst versucht, die Palette des Bildes in den schon belegten Einträgen der Farbtabelle des LCMs zu finden und diese dann mitzubenutzen. Gibt es keine Überdeckung, so wird die Palette des Bildes in den noch nicht benutzten Farbtabelleneinträgen abgelegt. Sind dort keine Plätze mehr frei, so werden aus dem Bereich der für BMP–Bilder reservierten Farben die ersten Farbtabelleneinträge überschrieben.

4.2.3 Farbtiefe von 24 Bit *(Truecolor)*

Der Typ `t_color` enthählt in einem 32 bit Wort die Farbinformationen der Rot-, Grün- und Blaukomponente, hierbei steht der Rotwert in den Bits 0–7, der Grünwert in den Bits 7–15 und der Blauwert in den Bits 16–23. Die Bits 24–31 sind unbenutzt. Es wird empfohlen das vordefinierte Makro `DISP_RGB( r, g, b)` aus "display_colors.h" zu benutzen, um einen Farbwert zu erzeugen. Mit den Makros `DISP_R_RGB( rgb)`, `DISP_G_RGB( rgb)` und `DISP_B_RGB( rgb)` können die einzelnen Grundfarben aus einem Farbwert extrahiert werden.
BMP–Bilder mit einer Farbtiefe von 24 Bit werden nun unterstützt. Meist reicht es jedoch Bilder mit 256 Farben zu verwenden, die nun ohne Einschränkungen, gleichzeitig dargestellt werden können und darüberhinaus Speicherplatz sparen.
Der Textstil `TXT_LIGHT` wird im 24 Bit–Modus durch eine passende Farbe realisiert und nicht mehr durch Rasterung der Schrift erreicht.

4.3 Ablegen von Daten ins RAM oder den Flash–Speicher

4.3.1 Handhabung "binärer Objekte" (Bilder, Fonts, Texte)

Da einige Befehle des GUI-Interpreters mit großen Datenmengen hantieren (z.B. `disp_bmp()` bei der Darstellung eines BMP-Bildes) ist es unpraktisch, diese jedesmal beim seriellen Übertragen des Befehls mitzusenden. Der GUI-Interpreter unterstützt deshalb binäre Objekte (die Funktionen befinden sich in den Dateien *lcm.h/c*), die zunächst in das RAM oder den optionalen externen Flash-Speicher des LCM hochgeladen werden. Die binären Daten können dann bei Bedarf über eine Objekt-ID angesprochen werden. Die Objekt-ID ist ein `uint32`-Wert, der dem Benutzer beim Hochladen eines Objekts ins RAM zugeteilt wird. Objekte im RAM können mit dem `LCMKillBinaryObject()` Kommando wieder aus dem Speicher gelöscht werden.
Bevor Objekte in den Flash-Speicher übertragen werden können, müssen die internen Datenstrukturen des LCM darauf vorbereitet und der Flash-Speicher gelöscht

Kommando	ID	Beschreibung
LCM_UPLOAD_TO_RAM	20	lädt binäre Daten ins RAM
LCM_KILL_BINARY_OBJECT	23	löscht ein binäres Datenobjekt aus dem RAM

Tabelle 5: Funktionen für Objekte, die in das interne RAM geladen werden können.

werden (`LCMInitFlash()`). Erst danach kann der Anwender Daten im Flash ablegen. Bei der Übertragung von binären Daten in den Flashspeicher (`LCMUploadToFlash()`) kann der Benutzer die Objekt-IDs selber wählen, wobei jedoch sichergestellt sein muß, daß eine ID nur einmal (!) vorkommt. Lediglich die `NULL_ID` (definiert in der Datei "portab_guii.h") darf nicht verwendet werden.

Entsprechendes gilt für die Übertragung von Makros aus dem RAM in den Flash-Speicher (`LCMMacroToFlash()`) (Makros sind Befehlssequenzen, die im Modul abgespeichert werden können, und die sich später durch einen einzigen Befehl "abspielen" lassen). Der Datenübertragungsvorgang wird mit dem `LCMFinalizeFlash()`–Kommando abgeschlossen. Hierbei legt das LCM die erzeugten Verwaltungsinformationen zu den Objekten, die dann bereits in den Flash geschrieben wurden, ebenfalls im Flash-Speicher ab. `LCMFinalizeFlash()` überprüft außerdem, ob IDs für Objekte im Flash-Speicher mehrfach vergeben wurden. Nach dem Übertragen der Verwaltungsinformationen können keine zusätzlichen Flash-Objekte an das LCM gesendet werden. Dies ist nur dann möglich, wenn man den gesamten Flash-Speicher mit `LCMInitFlash()` löscht und mit der Übertragung von vorne beginnt.

Handhabung der im Flash befindlichen Objekte nach einem Reset des LCM: Nach einem Reset durchsucht der GUI–Interpreter das externe Flash (sofern vorhanden) nach Verwaltungsinformationen über die dort enthaltenen Objekte. Anschließend werden die im GUI–Interpreter direkt enthaltenen Fonts in der Liste der RAM–Objekte abgelegt. Die Fonts bekommen hierbei, genau wie jedes andere binäre Objekt, eine ID vom GUI–Interpreter zugeteilt. Da zu diesem Zeitpunkt alle IDs der Objekte im Flash bekannt sind, erhalten die Fonts noch unbenutzte Kennungen. Das ist auch der Grund, warum bei Objekten im RAM die ID nicht gewählt werden kann, sondern diese vom GUI–Interpreter vergeben wird: Es könnte anderenfalls zu Kollisionen zwischen den IDs im Flash und denen im RAM kommen.

Kommando	ID	Beschreibung
LCM_INIT_FLASH	33	initialisiert die internen Datenstrukturen für die Datenverwaltung und löscht den Flash
LCM_INIT_FLASH_NO_OUTPUT	60	wie LCM_INIT_FLASH, es wird keine Ausgabe auf dem Display gemacht
LCM_UPLOAD_TO_FLASH	34	lädt binäre Daten ins Flash-Memory
LCM_MACRO_TO_FLASH	43	überträgt Makros aus dem RAM ins Flash-Memory
LCM_FINALIZE_FLASH	35	überträgt die Verwaltungsinformationen für die vorher geflashten Daten in den Flash-Speicher
LCM_FINALIZE_FLASH_NO_RESET	59	wie LCM_FINALIZE_FLASH, das Modul wird aber im Anschluß nicht zurückgesetzt
LCM_FLASH_DUMP	51	überträgt den Inhalt des Flashspeichers zum Master

Tabelle 6: Funktionen für Objekte, die in den optionalen externen Flashspeicher geladen werden können.

Beschränkungen: Die Anzahl der Objekte im RAM ist nur durch die Größe des RAM–Speichers begrenzt. Die Anzahl der Objekte im Flash ist ebenfalls durch dessen Größe und generell auf z.Z. 3000 Objekte beschränkt.

4.3.2 Beispiele

Häufig bietet es sich an verschiedene Daten (Texte, Bilder, Fonts) in den RAM– oder Flash–Speicher des LCM–Modules zu übertragen (siehe auch Abschnitt 4.3.1 auf Seite 50). Hierdurch kann z.B. ein länglicher serieller Transfer an kritischen Stellen vermieden werden. Im folgenden wird exemplarisch der Transfer eines Fonts in den RAM–Speicher beschrieben, wobei die Funktionen aus den Dateien *lcm.h/c* benutzt werden:

- Der Font muß im Speicher des Mastersystems liegen, z.B. in einem Array.
- Der Font wird durch den Befehl `LCMUploadToRam()` hochgeladen (mit

Kommando	ID	Beschreibung
LCM_ACCESS_BINARY_OBJECT	53	erlaubt den byteweisen Zugriff auf die Daten

Tabelle 7: Funktionen für Objekte im RAM oder im Flashspeicher.

dem Befehl `LCMUploadToRam()` können beliebige Speicherinhalte zum LCM–Modul übertragen werden).

- Wenn das Hochladen des Fonts erfolgreich war, dann liefert `LCMUpload-ToRam()` eine sogenannte Objekt–ID zurück, über die der Font dann angesprochen werden kann, diese könnte z.B. in die Variable `new_font` abgelegt werden.
- Jetzt muss der Font mit dem Befehl `system_font_init( new_font)` aus *system.h/c* noch aktiviert werden. Anschließend können Texte mit `disp_-text()` oder `disp_formatted_text()` ausgegeben werden (*display.h/c*).

Zum Übetragen von Bildern wird in der selben Art und Weise vorgegangen. Die Bilder müssen im BMP–Format vorliegen, von welchem verschiedene Varianten unterstützt werden: bis 256 Farben, mit und ohne RLE–Komprimierung und Truecolor–Bilder (vgl. Abschnitt 4.2.2 auf Seite 49).
Wie bei den Fonts werden Bilder ebenfalls mit `LCMUploadToRam()` ins RAM übertragen. Man bekommt eine Objekt–ID zurück mit der das Bild dann mit `disp_-bmp()` angezeigt werden kann.
Der Transfer von Fonts oder Bildern in den Flash–Speicher funktioniert wie folgt:

- Zunächst muß die Flashprozedur mit dem Befehl `LCMInitFlash()` vorbereitet werden, wodurch der gesamte Flash–Speicher auch gelöscht wird.
- Mit dem Befehl `LCMUploadToFlash()` können danach “binäre Objekt” (Fonts, Bilder, Texte usw.) in den Flash übertragen werden. Man kann hierbei die ID unter der die Objekte angesprochen werden sollen frei wählen (außer 0). Natürlich müssen die IDs unterschiedlich und eindeutig sein.
- Sind alle Objekte im Speicher abgelegt worden, wird das Flashen mit dem Befehl `LCMFinalizeFlash()` abgeschlossen.
- Anschließend startet das LCM–Modul neu und man kann die Objekte die nun im Flash liegen anhand ihrer Objekt–ID benutzen.

Die mit den Befehlen `LCMMacroRec()` und `LCMMacroStop()` aufgezeichneten Makros (siehe Abschnitt 4.4 auf Seite 55) überträgt man mit dem dafür vorgesehenen Kommando `LCMMacroToFlash()` in den Flashspeicher. Vorher abgelegte

Befehlssequenzen können so ausgeführt werden ohne die serielle Leitung zu belasten. Dies ist besonders für Systeme interessant, die nur mit einer geringen Baudrate arbeiten müssen.

4.3.3 Benutzung von "Flash–Pages"

Der Flashspeicher kann in mehrere "*Flash–Pages*" unterteilt werden, die unabhängig von einander beschrieben und gelöscht werden können. Die Größe der einzelnen Pages und deren Anzahl kann mit dem Befehl `LCMGetSystemInfo()` ermittelt werden. Beim lesenden Zugriff macht es keinen Unterschied in welcher Flash–Page die Objekte liegen. Hier wird lediglich die Objekt–ID verwendet.

Kommando	ID	Beschreibung
LCM_GET_SYSTEM_INFO	52	gibt u.a. die Anzahl und die Größe der Flash–Pages aus
LCM_SWITCH_FLASH_PAGE	56	Schaltet zwischen den einzelnene Flash–Pages um
LCM_FLASH_PAGE_INFO	57	gibt Informationen zu einer Flash–Page aus
LCM_FLASH_PAGES_CHECK_IDS	58	überprüft die Eindeutigkeit der IDs

Tabelle 8: Funktionen zur Kontrolle der Flash–Pages.

Mit dem Befehl `LCMSwitchFlashPage()` wird auf die entsprechende Flash–Page umgeschaltet. Die Nummerierung der Flash–Pages beginnt bei Null; wird ein Wert kleiner als Null als Parameter angegeben, wird der gesamte Flash–Speicher für Schreiboperationen benutzt anstatt eine einzelne Page. Wie oben beschrieben (Abschnitt 4.3.1 auf Seite 50), kann nun mit den Befehlen `LCMInitFlash()`, `LCMUploadToFlash()` usw. auf die aktuelle Flash–Page zugeriffen werden. Mit einem abschliessenden `LCMFinalizeFlash()` wird die Verwaltungsinformation zu den Objekten der aktuellen Flash–Page ebenfalls in dieser abgelegt und man kann auf die binären Objekte zugreifen.

Der Befehl `LCMFlashPageInfo()` liefert die Größe einer Flash–Page, die Anzahl der benutzten Bytes und die Anzahl der dort vorhandenen Objekte zurück. Mit `LCMFlashPagesCheckIDs()` läßt sich überprüfen, ob eine Objekt–ID versehentlich doppelt vergeben wurde.

4.4 Arbeiten mit Makros

Komplette Befehlssequenzen können als Makros aufgezeichnet werden und im RAM oder im Flashspeicher abgelegt werden (die seriellen Kommandos finden sich in den Dateien *lcm.h/c*). Hierbei können nur Befehle in einem Makro aufgenommen werden, die keine Daten zurückliefern (siehe Anhang B auf Seite 75), da ein Makro wie ein einziges Kommando aufgerufen wird und dessen Unterbefehle nicht mehr individuell mit dem Master kommunizieren.
Die Aufzeichnung eines Makros wird mit dem Befehl `LCMMacroRec()` gestartet, der eine Objekt–ID zurückliefert unter der das Makro später aufgerufen werden kann. Alle darauf folgenden Befehle werden dann nicht mehr ausgeführt, sondern im RAM des LCM–Modules abgelegt. Die Befehle können im Moment der Aufzeichnung auch ungültige Parameter enthalten, da die Abarbeitung erst beim Aufruf des Makros erfolgt. Die Makroaufzeichnung wird mit `LCMMacroStop()` abgeschlossen.
Aufgezeichnete Sequenzen werden mit `LCMMacroPlay()` abgespielt, wobei die Objekt–ID übergeben wird, die von `LCMMacroRec()` zurückgeliefert wurde.
Wie in Abschnitt 4.3 beschrieben, können Makros auch in den Flashspeicher verlegt werden, so daß sie nur einmal aufgezeichnet werden müssen.

4.5 Erstellung von Buttons

Zu Erstellung von Buttons sind zunächst Vorarbeiten zu erledigen. Zunächst müssen die Text– oder Grafik–Labels in den Speicher des Geräts übertragen und der Rahmen– bzw. der Button–Stil erzeugt werden. Hier besteht die Möglichkeit sowohl für den normalen als auch für den Ausgewählten Zustand des Buttons ein unterschiedliches Aussehen zu definieren.
Für ein einfaches Beispiel (siehe auch die Dateien “guii_button_demo.h/c”) werden folgenden Konstanten und Variablen definiert:

```
enum{NORMAL = 0, SELECTED = 1};

const char button_label_normal[] = "One Button";
const char button_label_selected[] = "SELECTED";

TObjectID my_button_label[2];
TObjectID my_framestyle[2];
TObjectID font_id;
TObjectID my_buttonstyle[2];
TObjectID my_button;
```

Der Button zeigt normalerweise den Text “One Button”, der sich im ausgewählten Zustand zu “SELECTED” ändert. Die beiden Texte werden im Beispiel ins RAM abgelegt.

```
LCMUploadToRAM( sizeof(button_label_normal),
   (const uint8*)(&button_label_normal), &my_button_label[NORMAL]);
LCMUploadToRAM( sizeof(button_label_selected),
   (const uint8*)(&button_label_selected), &my_button_label[SELECTED]);
```

Anschließend wird der Rahmen definiert, wobei der ausgewählte Button einen leicht verschiedene Umrandung haben soll.

```
disp_framestyle_init( &my_framestyle[NORMAL]);
disp_framestyle_set_colors(
   my_framestyle[NORMAL], RED, WHITE, BLACK, SILVER);
disp_framestyle_set_dimensions( my_framestyle[NORMAL], 2, 1, 7);
disp_framestyle_set_flags( my_framestyle[NORMAL], FS_SHADOW);
disp_framestyle_set_linestyle( my_framestyle[NORMAL], 0xFFFF);
disp_framestyle_clone( my_framestyle[NORMAL], &my_framestyle[SELECTED]);
disp_framestyle_set_colors(
   my_framestyle[SELECTED], RED, BLUE, SILVER, BLACK);
```

Mit der fertigen Rahmenbeschreibung kann man den eigentlichen Button–Stil erzeugen. Da der Button Text–Labels bekommen soll, muß der gewünschte Font ebenfalls eingestellt werden.

```
system_get_font_pointer( "Mono8x16", &font_id);

button_style_init( &my_buttonstyle[NORMAL]);
button_style_set_font( my_buttonstyle[NORMAL], font_id, TXT_NORMAL, BLUE);
button_style_set_linefeed( my_buttonstyle[NORMAL], 0);
button_style_set_framestyle( my_buttonstyle[NORMAL], my_framestyle[NORMAL]);
button_style_clone( my_buttonstyle[NORMAL], &my_buttonstyle[SELECTED]);
button_style_set_font( my_buttonstyle[SELECTED], font_id, TXT_NORMAL, BLACK);
button_style_set_framestyle( my_buttonstyle[SELECTED], my_framestyle[SELECTED]);
```

Schließlich wird der Button angelegt und aktiviert.

```
button_define( 110, 95, 100, 50, STANDARD_BUTTON | BUTTON_KLICK,
   my_buttonstyle[NORMAL], my_buttonstyle[SELECTED],
   my_button_label[NORMAL], my_button_label[SELECTED], &my_button);
button_activate( my_button);
```

Die Abfrage des Buttons erfolgt mit den Befehlen aus Abschnitt 3.11 auf Seite 33. Die Freigabe des belegten Speichers erfolgt mit der folgenden Sequenz.

```
button_deactivate( my_button);

button_style_free( my_buttonstyle[NORMAL]);
button_style_free( my_buttonstyle[SELECTED]);

disp_framestyle_free( my_framestyle[NORMAL]);
disp_framestyle_free( my_framestyle[SELECTED]);

LCMKillBinaryObject( my_button_label[NORMAL]);
LCMKillBinaryObject( my_button_label[SELECTED]);
```

4.6 Abfrage eines optischen Encoders

Ist das Display–Modul mit einem (optionalen) optischen Encoder ausgestattet, so kann dessen Zustand über das Event–Handling (siehe 3.11 auf Seite 33) abgefragt werden. Ereignisse der folgenden Typen können dabei auftreten (Tabelle 9):

Event	Beschreibung
ENCODER_PRESSED_EVENT	Wird beim Drücken des Encoder–Knopfes ausgelöst.
ENCODER_RELEASED_EVENT	Wird beim Loslassen des Encoder–Knopfes ausgelöst.
FAST_ENCODER_EVENT	Wird ausgelöst, wenn der Encoder–Knopf gehalten wird.
ENCODER_CHANGE_EVENT	Wird beim Drehen des Encoders ausgelöst.

Tabelle 9: Mögliche Ereignisse, die durch einen Encoder ausgelöst werden können.

Je nachdem, wie das Schaltverhalten des Encoder–Knopfes mit dem Befehl `EncoderSetCharacteristics()` eingestellt wird, werden die entsprechenden Ereignisse ausgelöst (siehe Funktionsreferenz). Ein weiterer Befehl (`EncoderGetDifference()`) wird benötigt, um nach einem `ENCODER_CHANGE_EVENT` die Einstellung des Encoders zu ermitteln. Die beiden Encoder–Befehle sollten immer mit der Kennung `FIRST_ENCODER_ID` oder mit dem Rückgabewert des Befehls `events_get_next_event()` angesteuert werden.
Ein einfaches Quellcodefragment demonstiert die Nutzung:

```
static void EncoderTest( void)
{
   err_code err;

   EncoderSetCharacteristics( FIRST_ENCODER_ID, 1, ENCODER_SWITCH_BOTH);
   events_activate( 16);
   while (1)
   {
      e_event_type event;
      TObjectID encoder_id;

      (void)events_collect();

      err = events_get_next_event( &event, &encoder_id);
      if (event != NO_EVENT)
      {
         printf("An event occured.");
      }/* if */
      if (ERR_OK == err)
      {
         if (ENCODER_PRESSED_EVENT == event)
         {
            printf("Encoder pressed. \n");
         }/* if */
         if (ENCODER_RELEASED_EVENT == event)
         {
            printf("Encoder released. \n");
         }/* if */
         if (ENCODER_CHANGE_EVENT == event)
         {
```

```
            int16 diff;

            (void)EncoderGetDifference( encoder_id, &diff);
            printf("Encoder changed %d. \n", diff);
         }/* if */
      }/* if */
   }/* while */
}/* EncoderTest */
```

4.7 Übertragung des kompletten Display–Inhaltes an den Master

Um den Inhalt der Anzeige zum Master zu übertragen, muß dieser Vorgang zunächst vorbereitet werden. Hierzu dient der Befehl `LCMDownloadDisplay()`, der die Abmessungen des zu übertragenden Display-Inhaltes, die Anzahl der Datenbytes und eine Objekt-ID liefert.
Die Daten selber werden durch wiederholtes Aufrufen der Funktion `LCMDownloadData()` nach Bedarf abgeholt. Diese Funktion bekommt die Objekt-ID übergeben, die durch den Aufruf von `LCMDownloadDisplay()` erhalten wurde, und füllt einen Puffer mit einem Teil der Anzeigedaten. Das LCM überträgt in diesem Fall die Bytes der Anzeige zeilenweise von oben nach unten. Der Anwender kann so relativ einfach z.B. einen Drucker mit diesen Daten ansteuern. Der Download der Display-Daten ist beendet, wenn `LCMDownloadData()` einen `ERR_BEMP`-Fehler (“Puffer leer”) liefert (so wird auch sichergestellt, daß die LCM-internen Resourcen wieder freigegeben werden). Das Paket, das mit `ERR_BEMP` empfangen wurde, ist das letzte, welches gültige Daten enthält.

4.8 Erstellung von Animationen

Mit den Animationsfunktionen ist es möglich mehrere BMP–Bilder zu kleinen Filmsequenzen zusammenzufassen. Die einzelnen Bilder (*Frames*) müssen vorher im Flash– oder im RAM–Speicher abgelegt werden. Die Funktion `AnimAddFrame()` fügt einen Frame zur Animation und definiert für jeden Frame, wie lange er zur Anzeige kommt. Ferner kann man festlegen, daß die Animation immer wieder starten soll oder ständig vorwärts und rückwärts gezeigt wird. Der Master kann die Animation jederzeit anhalten und wieder starten.
Das folgende Quellcodefragment demonstiert die Nutzung:

```
#include "guii_animation.h"

static void AnimTest( void)
{
   TAnimObject *clock_anim;
   uint8 i;
   uint32 bmp_ptr;

   AnimInit( &clock_anim);
   for (i = 0; i < 12; i++)
   {
```

```
      GetBObject( 1000 + i, &bmp_ptr);
      AnimAddFrame( clock_anim, (t_bmp_ptr)bmp_ptr, 20);
   }/* for */

   ...

   AnimActivate( clock_anim, 50, 50, 100, 100, 0, BOOL_FALSE, BOOL_FALSE);

   ...

   AnimDeactivate( clock_anim);
   AnimFree( clock_anim);
}/* AnimTest */
```

Am Anfang wird die Animation mit dem Befehl `AnimInit()` eingerichtet. Danach hängt der Befehl `AnimAddFrame()` die einzelnen Frames aneinander (die Bilder liegen im Beispiel ab Objekt–ID 1000 im Speicher). Hierbei soll beim Abspielen der Sequenz jedes Bild für 200 ms dargestellt werden. `AnimActivate()` startet die Animation vom ersten Bild an und gibt vor, daß diese nur einmal abläuft. `AnimDeactivate()` löscht die Animation vom Bildschirm, sie könnte allerdings mit `AnimActivate()` an einer anderen Stelle wieder zum Vorschein gebracht werden. `AnimFree()` zerstört die Animation und gibt den belegten Speicher wieder frei. Daneben gibt es noch die Befehle `AnimStart()`, `AnimCont()` und `AnimStop()` mit denen der Master den Ablauf neu starten, fortsetzen oder anhalten kann.

Beschränkungen: Wenn der Master sehr viele und/oder große Animationen startet, dann kann der vorgegebene zeitliche Ablauf nicht mehr eingehalten werden, da die Darstellung der Einzelbilder zuviel Resourcen verbraucht. Außerdem nimmt der Durchsatz der verarbeiteten Befehle an den Master ab, wenn dieser gleichzeitig Animationen abspielt.

5 Beispiel–Implementationen

Neben dem Windows–Master, der es erlaubt auf einem PC Software für die LCM–Module zu entwickeln, steht auf der CD eine weitere Beispielimplementationen des Masters für den GUI–Interpreter zur Verfügung (“Complete_Master”). Die Quelltexte sind für eine UNIX/Linux Umgebung erstellt und lassen sich leicht an andere, kundenspezifische Hardware anpassen.
Die Master–Implementationen geht zunächst davon aus, daß das LCM mit dem GUI–Interpreter eingeschaltet wurde, oder aus einem Reset kommt. Es wird deshalb nach dem Autobauding (vgl. Abschnitt 2.1) der Init–String `zBIN\r` gesendet, um in den binären Paketmodus zu wechseln.

5.1 Der Windows–Master für den GUI-Interpreter

Der Windows–Master enthält eine komplette Implementation der seriellen Kommunikation zu den LCM–Modulen. Mit ihm ist es sehr leicht möglich Anwendungssoftware in C für diese zu schreiben, die später ohne Änderungen auf der Zielhardware, die die LCM–Module ansteuern soll, lauffähig ist.
Der Windows–Master ist Teil des Starterkits für die LCM–Module und wird in der dazugehörigen Dokumentation beschrieben.

5.2 Master–Implementation für eine UNIX/Linux Umgebung

Der “Complete_Master” unterstützt alle Möglichkeiten der Kommunikation (z.B. kurze/lange Antwortpakete und CRC16–Checksummen) und demonstriert viele Funktionen des GUI–Interpreters. Die einzelnen Quelltexte und die dort enthaltenen Beschreibung der Quelltextdateien:

gui_interpreter_master.c: Öffnet die serielle Schnittstelle und sendet die Befehle an den Interpreter.

serial_transfer.h/c: Funktionen zum Senden und Empfangen von Paketen über die serielle Schnittstelle; Funktionen zum Aktivieren/Deaktivieren der verschiedenen Übertragungsprotokolleigenschaften.

serial_local.h/c: Lowlevel–Befehle für die Benutzung der seriellen Schnittstelle und das Senden und Empfangen der Daten (hardware–abhängige Funktionen, die an Sie an Ihr System angepassen müssen).

display.h/c: Funktionen der GUI–Bibliothek zum Zeichnen auf dem Display; hier lösen GUI–Interpreter Kommandos diese Funktionen aus.

lcd.h/c: Funktionen zur Steuerung des LCD–Panels.

sound.h/c: Funktionen zum Ansteuern des optionalen Piezo–Lautsprechers.

system.h/c: Funktionen der GUI–Bibliothek u.a. zur Zeichensatzverwaltung; hier senden diese Funktionen ein entsprechendes Kommando über die serielle Schnittstelle an den GUII–Interpreter.

lcm.h/c: Protokollsteuerbefehle und Funktionen zum Übetragen von binären Objekten (Fillstyles, Spritepatterns, Bitmaps usw.) an das LCM.

touch.h/c: Funktionen für die Ansteuerung eines optional vorhandenen Touch–Glases.

button.h/c: Funktionen für die Erzeugung und Kontrolle von Touch–Buttons.

tglass.h/c: Funktionen zum Kalibrieren des Touchglases.

keyboard.h/c: Befehle zur Verwendung eines Keyboard-Channels, der für Touch–Tastaturen benötigt wird.

t_keyb.h/c: Funktionen für die Verwendung der Touch–Keyboards für Module, die mit einem Touch–Display ausgestattet sind.

event_handling.h/c: Funktionen zur Abfrage von Touch–Objekten (Buttons, Tastaturen).

serial_command_buffer.h/c: Funktionen, um einzelne Kommandos zusammenzusetzen und wieder in die einzelnen Parameter zu zerlegen.

checksum.h/c: Funktionen zum Berechnen von Checksummen.

global_settings_guii.h, config_master.h: Definition von globalen Konstanten und Compile–Switches. **Beachten Sie, daß die Compile–Switches passend zu Ihrem Gerät eingestellt werden müssen (z.B. kommt es zu fehlerhaften Darstellungen, wenn ein Schwarz/Weiss-Gerät Befehle vom Master bekommt, die für ein Farbgerät gedacht waren).**

serial_commands.h: Definition der einzelnen Kommando–Kennungen.

portab.h: Definition von grundlegenden Datentypen, um Kompatibilität zwischen verschiedenen Compilern herzustellen.

portab_guii.h: Datentypen, die speziell für den GUI–Interpreter benötigt werden.

errors.h/errsys.h/errors_guii.h: Definition der Fehlernummern.

5.3 Vorgehensweise bei der Implementierung eines Masters für den GUI–Interpreter

Wenn Sie den als Beispiel beigefügten Code für den Master nicht verwenden möchten oder z.B. eine andere Programmiersprache als C auf Ihrem ansteuernden Gerät verwenden, orientieren Sie sich an der "Complete–Master"–Implementation. Wir empfehlen folgende Vorgehensweise:

- Schreiben Sie Funktionen zum Ansteuern der seriellen Schnittstelle mit einer Standardbaudrate von 9600 bis 115200 Baud, 8 Datenbits, 1 Stopbit, keine Parität (anpassen von "serial_local.h/c" der GUII–Master–Library). Im wesentlichen müssen Funktion zum Lesen und Schreiben von Bytes implementiert werden. Beachten Sie beim Einlesen von Zeichen aus der seriellen Schnittstelle, daß es unterschiedlich lange dauert, bis ein Befehl vom LCM ausgeführt und eine Antwort gesendet wird. Man kann demnach nicht davon ausgehen, dass nach dem Absenden eines Befehls sofort die Antwort geliefert wird. Eine angemessene Timeout–Behandlung sollte vorgesehen werden. Es ist wichtig, daß der Timeout Ihrer Funktion zum Einlesen eines Bytes bei 0.1s liegt.
- Durch Aufruf der Funktion `InitSerialConnection()` (in "serial_transfer.h/c") werden die in "serial_local.h/c" erstellten Basisfunktionen zugeordnet.
- Implementieren Sie das Autobauding–Verfahren, so daß sich der GUI–Interpreter auf die von Ihnen gewünschte Baudrate anpassen kann (siehe die Funktion `TriggerAutobauding()` in "serial_transfer.h/c").
- Als ersten Test senden Sie den String `zBIN\r` (Hex: `0x7a 0x42 0x49 0x4e 0x0d`), die Einschaltmeldung des LCMs müßte dann verschwinden. (Relevante Dateien aus den Beispielen: "serial_transfer.h/c", siehe hier die Funktion `InitDevice()`).
- Schreiben Sie nun Funktionen, die Pakete entsprechend der Dokumentation zusammensetzen; hierbei sollte zunächst auf alle Zusatz–Features des Protokolls verzichtet werden (Sequenzierung, Adressmodus, CRC16–Checksumme, lange Antworten vom LCM). Als Checksumme werden einfach alle anderen Bytes des Pakets aufsummiert. (Relevante Dateien: "serial_command_buffer.h/c": `SCBAdd...()`, "serial_transfer.h/c ": `SendPackage()`, "serial_commands.h", "display.h/c": z.B. `disp_line()`, "checksum.h/c": `AddSimpleSum()`, `SimpleSum()`).
- Schicken Sie ein einfaches Paket (z.B. `DISP_LINE`) an das LCM und versuchen Sie das Antwort–Byte einzulesen. Hat alles funktioniert, so sollte 0x00 empfangen werden. (Relevante Dateien: "serial_transfer.h/c": `SendPackage()`, `ReceivePackage()`, "display.h/c": z.B. `disp_line()`).

- Erweitern Sie dann die Funktionalität des Masters nach Ihren Bedürfnissen. Es ist ausreichend, wenn Sie nur die Protokolleigenschaften und Funktionsaufrufe realisieren, die Sie verwenden möchten.

6 Portabilität zwischen LCM-Modulen und GUI–Bibliotheks–Anwendungen

Abhängig von der Art des Projektes und anderen Randbedingungen möchte man entweder ein fertiges LCM–Modul für die Benutzerschnittstelle der Anwendung einsetzen, oder aber eine eigene Hardware dafür bauen. Für die Verwendung einer eigenen Hardware bieten wir eine Sourcecode-Bibliothek an, deren Befehle weitgehend kompatibel mit denen des GUI–Interpreters auf den LCM-Modulen sind.

Dadurch wird eine gute Portierbarkeit von Anwendungs-Programmen ermöglicht, so daß ein Hardwarewechsel ohne große Änderungen in der Programmierung der Benutzerschnittstelle erfolgen kann !

6.1 Eigenschaften des Grafiksystems – die GUI–Bibliothek als Basis für den GUI–Interpreter

Die Simplify Technologies GUI–Bibliothek liegt als Grafiksystem dem GUI–Interpreter zugrunde. Die Funktionen zur Erstellung der Datenpakete können daher analog zu den in der GUI–Bibliothek verwendeten angelegt werden. Damit kann eine Anwendung verhältnismäßig leicht von einer GUI–Interpreter–Anwendung in eine GUI–Bibliothek–Anwendung portiert werden. Der mitgelieferte Beispielcode (siehe Kapitel 5 auf Seite 60) zur Ansteuerung des GUI–Interpreters versucht, eine möglichst weitgehende Entsprechung zu realisieren. Er ist, wie auch die GUI–Bibliothek, in ANSI–C geschrieben.

Die Funktionen der GUI–Bibliothek können in zwei Klassen eingeteilt werden:

Funktionen, die hardwareunabhängig angelegt sind: Diese steuern die sogenannten *Channels* in der GUI–Bibliothek. Dies sind abstrakte Repräsentanten der Ein- bzw. Ausgabegeräte wie z.B. Displays, Tastaturen etc. . Beispiele für solche Funktionen, bei deren Anwendung kein Bezug auf die real vorhandene Hardware genommen wird, sind z.B.: `disp_line()` (zeichnet eine Linie auf dem Display–Channel) und `disp_text()` (Ausgabe von Text). Entsprechende Funktionen werden vom Master bei der Ansteuerung des GUI-Interpreters verwendet.

Funktionen, die einen Bezug zur real vorhandenen Hardware haben: Diese Funktionen steuern die sogenannten *Devices* an. Dies sind die real vorhandenen Peripheriegeräte, wie z.B. das LCD–Display oder der Piezo–Lautsprecher. Die Funktionen sind durch die Eigenschaften der jeweiligen Peripheriegeräte geprägt; so gibt es z.B. für LCD–Displays, bei denen der Kontrast eingestellt werden kann, die Funktion `lcd_contrast()`.

In der GUI–Bibliothek werden die *Channels* mit den *Devices* verbunden, so daß die Verwendung einer Funktion für einen Channel letztlich ihre Auswirkung auf dem damit verbundenen Device (z.B. ein LCD–Display) findet. Die für den GUI–Interpreter relevanten Channels und Devices sind in Tabelle 10 aufgeführt.

Channel	dazugehörendes Device
Display–Channel	LCD–Device
Sound–Channel	Piezo–Device
Touch–Channel	Touchglas–Device

Tabelle 10: Abstrakte Channels und dazugehörige Devices.

Für die Verwendung des GUI–Interpreters auf den LCM–Modulen ist es nicht nötig, zwischen Channel und Device zu unterscheiden, da die Module ja bereits fest mit einer bestimmten Hardware / einem bestimmten Display ausgerüstet sind, so daß Befehle wie `disp_line()` direkt auf diesem Display arbeiten.

6.2 Unterschiede zwischen Funktionen für GUI–Interpreter und GUI–Bibliothek

Bei der Umsetzung der Befehle der GUI–Bibliothek für den GUI–Interpreter wurden einige Vereinfachungen vorgenommen, da der Bibliothek in diesem Fall genau bekannt ist, welche Hardware eingesetzt wird (eben die Module der LCM–Serie), so daß eine allgemeinere Behandlung des Displays nicht notwendig ist. Daher werden gegenüber der GUI–Bibliothek die Initialisierung und die Handhabung der *Channels* und *Devices* vereinfacht. Beim Starten des Moduls werden automatisch die vorhandenen Devices initialisiert, passende Channels dazu erzeugt und mit den Devices verbunden, so daß man direkt Displaybefehle senden kann (siehe auch Anhang C auf Seite 78.

Manche Funktionen der GUI–Bibliothek verwenden Pointer als Argument, z.B. die Adresse eines BMP–Bildes, das angezeigt werden soll. Die Verwendung von Pointern ist so mit dem GUI–Interpreter nicht möglich, da die Ansteuerung von einem anderen System aus erfolgt, so daß auf dort vorhandene *Objekte* nicht vom LCM–Modul direkt zugegriffen werden kann. Daher wird in für den GUI-Interpreter folgendermaßen vorgegangen:

1. Das betreffende Objekt wird entweder im LCM–Modul bereitgestellt und über einen Befehl dort erzeugt oder vom ansteuernden Gerät auf das LCM–Modul übertragen.

2. Das LCM–Modul nimmt das Objekt in eine interne Liste auf. Für Objekte, die zur Laufzeit im RAM gespeichert werden, erzeugt das LCM eine ID–Nummer (*Objekt ID*), die an das ansteuernde Gerät übertragen wird. Objekte, die einmalig in den optionalen externen Flash–Speicher übertragen werden, können dabei mit einer von außen vorgegebenen ID versehen werden. Lediglich die `NULL_ID` (definiert in der Datei “portab_guii.h”) darf nicht verwendet werden.
3. Wenn nun das ansteuernde Gerät in einer Funktion auf dieses Objekt Bezug nehmen möchte, überträgt es die entsprechende Object–ID anstelle eines Pointers.

Für die Anwendungsprogrammierung besteht der Unterschied letztlich nur darin, daß für die GUI-Bibliothek ein Pointer an die entsprechenden Funktionen übergeben wird, im Fall des GUI-Interpreters eine ID. Weil man einen Pointer als eine besondere Form einer ID ansehen kann, sind im Fall der Portierung so kaum Änderungen im Anwendungs-Code notwendig.

Eine Umsetzung der Befehle in entsprechende Datenpakete für die Programmierung des ansteuernden Gerätes findet sich in Kapitel 5 auf Seite 60. Dort wird beispielhaft auch eine Implementation des seriellen Protokolls, das im Detail in Kapitel 2 auf Seite 8 beschrieben ist, vorgestellt.

Weitere Einzelheiten zur Simplify Technologies GUI–Bibliothek finden Sie im entsprechenden Handbuch, daß Sie bei Bedarf gerne von uns erhalten können.

7 Sicherheitsfragen

7.1 Fakten

Software und Hardware sind komplexe Systeme, und es ist nur möglich, eine begrenzte Anzahl der möglichen Zustände zu testen. Bereits bei fertigen Programmen, die mit unterschiedlichen Parametern und Randbedingungen laufen, ergeben sich normalerweise sehr viele mögliche Abläufe. Daher kann nicht davon ausgegangen werden, daß eine Software fehlerfrei ist. Auch Hardwarekomponenten können Fehler aufweisen, z. B. durch Alterung.
Bei der Verwendung des GUI-Interpreters können u. U. zusätzlich Übertragungsfehler bei der Ansteuerung durch den Master auftreten. Ein stabil laufender Master ist Voraussetzung für einen sicheren Betrieb.
Simplify Technologies ist bemüht, hochqualitative Software und Hardware bereitzustellen. Dennoch ist auch beim LCM und dem GUI-Interpreter mit dem Auftreten von Fehlern zu rechnen. Sollte Ihnen ein Fehler auffallen, teilen Sie uns dies bitte mit, damit er behoben werden kann.

7.2 Vorsichtsmaßnahmen

Die Erstellung zuverlässiger Software und Systeme kann hier nicht in Kürze adäquat abgehandelt werden. Bitte informieren Sie sich in der zu diesem Thema zahlreich verfügbaren Literatur [3], der Dokumentation Ihrer Entwicklungswerkzeuge sowie in Fachliteratur für Ihre Anwendung und in eventuell dafür geltenden Vorschriften.
Ein defensiver Programmierstil kann wesentlich zur Zuverlässigkeit Ihrer Anwendung beitragen. Dazu gehört u. a. das Überprüfen von Parametern von Funktionen auf Überschreitung eines erlaubten Wertebereiches und das Abtesten der Rückgabewerte aufgerufener Funktionen, um aufgetretene Fehler abfangen zu können und ebenso die Überprüfung der korrekten Datenübertragung zum LCM durch die Möglichkeiten des Kommunikationsprotokolls.
Die fertiggestellte Anwendung sollte in jedem Fall sorgfältig getestet werden. Unter Umständen kann es angezeigt sein, einen von der Software unabhängigen ”Watchdog” einzusetzen, der die Integrität des Gesamtsystems und der Anwendung überprüft, und der das System in einen sicheren Zustand überführt, wenn diese Überprüfung fehlschlägt.

[3] z. B. Hoang Pham: Software Reliability, Springer 2000, ISBN 981-3083-84-0, oder John D. Musa: Software Reliability, McGraw-Hill 1999, ISBN 0-07-913271-5

8 Funktionsreferenz

Eine Übersicht und die Beschreibung der Funktionen finden Sie in diesem Dokument in der Beschreibung der Funktionsmodule im Kapitel 3.

Die detaillierte Funktionsreferenz zur den Funktionen des GUI-Interpreters befindet sich in der separaten Funktionsreferenz, die Sie auf der CD des Starterkits finden.
Für jede der über 300 Funktionen wird dort neben dem Kommando, den Parametern und dem Kommunikationsprotokoll für den Binär-Modus auch der Funktionsprototyp der Master–Beispielimplementation mit seinen Parametern beschreiben.
Die Pakete in den Beispielsequenzen gehen von einer Übertragung mit langen Antwortpaketen, ohne Sequenzierung und mit einer einfachen Checksumme aus (vgl. Abschnitt 2.3). Dem entsprechend ist das Paketstatusbyte immer `0x80`. Sind in den Beispielsequenzen Bytes unbestimmt, da sie z.B. von den Daten abhängen, so wird "xx" als Platzhalter verwendet. Hinter dem Beschreibungstext für jedes Byte in den Beispielpaketen steht der Dezimalwert des entsprechenden Parameters in runden Klammern.
Wie im folgenden Beispiel ist bei manchen Befehlen auch die Ansteuerung im ASCII–Modus möglich:

DISP_LINE

Zeichnet eine Linie mit den momentan gültigen Linienattributen (Linienmuster und Dicke) von und relativ zur aktuellen Cursorposition. Anschließend wird der Endpunkt der Linie zur neuen Cursorposition.

Kommando–ID: 217

Befehlsparameter:

- Relative x–Koordinate, zu der die Linie führen soll (int16),
- relative y–Koordinate, zu der die Linie führen soll (int16).

Rückgabewerte:

- Enthält Error-Code (uint8) (ERR_OK: Erfolg, ERR_PAR: bei Endpunkten, die auf negativen Koordinaten lägen oder bei Linienlänge 0, ERR_DDE: bei Fehlern des mit dem Display verbundenen Device (insbesondere bei Endpunkten mit zu großen Koordinaten)).

Beispielsequenz:

Befehlspaket

0A	Paketlänge (10)
80	Statusbyte
00	Befehls ID (217) MSB
D9	Befehls ID (217) LSB
00	x–Koordinate (50) MSB
32	x–Koordinate (50) LSB
00	y–Koordinate (60) MSB
3C	y–Koordinate (60) LSB
01	Checksumme MSB
D1	Checksumme LSB

Antwortpaket

07	Paketlänge (7)
80	Statusbyte
00	Übertragungsfehler (ERR_RESPONSE_OK) MSB
01	Übertragungsfehler (ERR_RESPONSE_OK) LSB
00	Befehlsfehler (ERR_OK)
00	Checksumme MSB
88	Checksumme LSB

Funktionsprototyp der GUI–Interpreterfunktion für den Master:

```
err_code disp_line(
    int16 delta_x,
    int16 delta_y);
```

Parameter der GUI-Interpreterfunktion für den Master:

delta_x, delta_y: Relative Distanz des Endpunktes von der aktuellen Cursorposition.

ASCII-Modus:

Befehl: zLINE

Parameter: relative Koordinaten des Endpunkts der Linie

Beispiel: zLINE 50 60 <cr>

Antwort: Error-Code

A Fehlercodes

Zusammenstellungen der Fehlernummern, die den Fehlercodes entsprechen, die in der Funktionsreferenz verwendet werden. Diese Fehlernummern werden bei der seriellen Übertragung verwendet.

Fehlercode	Fehler-Nr.	Bemerkung
ERR_OK	0	Erfolg
ERR_NYI	1	Funktion ist nicht implementiert
ERR_FNA	2	Funktions steht jetzt nicht zur Verfügung
ERR_PAR	3	Parameter außerhalb des Bereichs
ERR_DNA	4	Gerät kann nicht angesprochen werden
ERR_DIU	5	Gerät wird illegal benutzt
ERR_MEM	6	Speicherplatz reicht nicht aus
ERR_CNA	7	Channel nicht verfügbar
ERR_DDE	8	Fehler des Gerätetreibers
ERR_DFE	9	Fehler der Anzeigefunktion
ERR_FNS	10	Format wird nicht unterstützt
ERR_MAI	11	Speicherbereich nicht ausreichend
ERR_ONF	12	Objekt konnte nicht gefunden werden
ERR_PAS	13	Eigenschaft wurde bereits gesetzt
ERR_LST	20	Fehler bei der Listenverarbeitung
ERR_FIF	21	Fifo-Fehler: Fifo ist voll
ERR_FIL	22	Fifo-Fehler: Fifo ist gesperrt
ERR_FIE	23	Fifo-Fehler: Fifo ist leer
ERR_BUT	30	Fehler beim Bearbeiten eines Buttons
ERR_TKB	32	Fehler beim Bearbeiten einer Tastatur
ERR_NOM	34	Menü nicht verfügbar
ERR_MEU	35	Fehler bei der Menübearbeitung
ERR_MHI	36	Menü muß deaktiviert werden
ERR_BRI	37	Fehler bei einem Fortschrittsbalken
ERR_FLG	50	Flag ist bereits gesperrt
ERR_NCA	60	Kein Zeichen verfügbar, Tastaturpuffer ist leer
ERR_KBF	61	Tastaturpuffer ist voll
ERR_PRE	70	Allgemeiner Druckerfehler
ERR_PRP	71	Drucker hat kein Papier mehr
ERR_PRT	78	Zeitüberschreitung beim Drucker

Fortsetzung auf der nächsten Seite

Fortsetzung von der vorherigen Seite		
Fehlercode	Fehler-Nr.	Bemerkung
ERR_PRN	79	Drucker ist nicht bereit
ERR_OK_TRUE	98	Positives Ergebnis einer Funktion (kein Fehler)
ERR_OK_FALSE	99	Negatives Ergebnis einer Funktion (kein Fehler)
ERR_BSY	100	Etwas ist noch nicht fertig
ERR_TMP	101	Fehler beim Auslesen des Temperatursensors
ERR_STI	130	Zeitüberschreitung bei der sereillen Schnittstelle
ERR_SRE	132	Empfangsfehler bei der seriellen Schnittstelle
ERR_STE	134	Übertragungsfehler bei der seriellen Schnittstelle
ERR_FTE	139	Fehler beim Testen des Flashspeichers
ERR_MMA	140	Speichertestfehler: Moving-Inversions-Error
ERR_MA1	141	Speichertestfehler: 1. Byte beim Adresstest
ERR_MA2	142	Speichertestfehler: 2. Byte beim Adresstest
ERR_MA3	143	Speichertestfehler: 3. Byte beim Adresstest
ERR_MA4	144	Speichertestfehler: 4. Byte beim Adresstest
ERR_FWE	145	Schreibaktivierungsfehler beim Flash-Speicher
ERR_FLE	146	Fehler beim Flash-Speicher
ERR_FWA	147	Adressschreibfehler beim Flash-Speicher
ERR_FBE	148	Fehler beim Überschreiten von 128Byte Grenzen beim Flash-Speicher
ERR_FER	149	Fehler beim Löschen des Flash-Speichers
ERR_LCC	150	LCD-Controller-Fehler
ERR_SNR	201	Falsche Anzahl von Bytes an der seriellen Schnittstelle empfangen
ERR_SCK	202	Checksummen-Fehler bei der Paketübertragung
ERR_SQC	203	Sequenz-Fehler bei der Paketübertragung
ERR_CNS	204	Befehl wird nicht unterstützt
ERR_BON	205	Binäres Objekt kann nicht gefunden werden
ERR_MRE	206	Maximale Anzahl der Wiederholungen bei der Paketübertragung erreicht
ERR_REP	207	Antwortpaket wird wiederholt
ERR_MDE	208	Mehr Datenbytes als erwartet wurden empfangen
ERR_MREC	209	Fehler bei der Makroaufzeichnung
ERR_BEMP	210	Puffer geleert
ERR_MAC	211	Fehler bei der Makroverarbeitung
ERR_MBSY	212	Beschäftigt mit der Aufzeichnung eines Makros
ERR_BNS	213	Baudrate wird nicht unterstützt
		Fortsetzung auf der nächsten Seite

Fortsetzung von der vorherigen Seite		
Fehlercode	Fehler-Nr.	Bemerkung
ERR_INU	214	Objekt–IDs sind nicht einzigartig
ERR_BUG	255	Interner Fehler

Tabelle 11: Fehlercodes

Fehlercode	Fehler-Nr.	Bemerkung
DL_ERR_OK	0	Erfolg
DL_ERR_DATAINDEX	1	Datenzugriff mit ungültigem Index
DL_ERR_WORLD_-DATA_INVALID	2	Konversion nach Weltkoordinaten ist fehlgeschlagen
DL_ERR_MEM	3	Speicher konnte nicht alloziert werden
DL_ERR_NOSPACE	4	Es können keine weiteren Daten abgelegt werden
DL_ERR_NODATA	5	Es konnten keine gültigen Daten geholt werden
DL_ERR_PARAMETER	6	Falsche Parameter
DL_ERR_BUG	255	Interner Fehler

Tabelle 12: Fehlercodes, die für das Logger–Framework spezifisch sind.

Antwort-String	Bemerkung
WPA	Falscher Parameter
STS	Befehlsstring zu kurz
STL	Befehlsstring zu lang
ICO	Ungültiges Kommando
IER	Interner Fehler

Tabelle 13: Fehlercodes, die neben den Fehlernummern im ASCII–Mode auftreten können

B Makrofähige Kommandos

Die folgenden Befehle können in Makros verwendet werden:

ANIM_ACTIVATE = 2103,
ANIM_CONT = 2107,
ANIM_DEACTIVATE = 2104,
ANIM_START = 2106,
ANIM_STOP = 2105,
BARIND_ACTIVATE = 908,
BARIND_DEACTIVATE = 909,
BARIND_SET = 910,
BUTTON_ACTIVATE = 567,
BUTTON_CHANGE_LABEL = 582,
BUTTON_DEACTIVATE = 568,
BUTTON_GHOST_BUTTON = 565,
BUTTON_LIFT_HOLD_BUTTON = 585,
BUTTON_PRESS_HOLD_BUTTON = 584,
BUTTON_SET_GHOSTLABEL = 597,
BUTTON_SET_GHOSTSTYLE = 596,
BUTTON_SET_POSITION = 580,
BUTTON_SET_STYLE = 583,
BUTTON_UNGHOST_BUTTON = 566,
CLIP_LINE = 1001,
CLIP_POINT = 1000,
CLIP_TEXT = 1002,
DISP_ARC = 226,
DISP_BMP_ALPHA = 281,
DISP_BMP = 240,
DISP_CIRCLE = 224,
DISP_CLEAR = 209,
DISP_COLORED_CIRCLE = 225,
DISP_COLORED_RECTANGLE = 222,
DISP_COPY_AREA = 280,
DISP_DELETE_FRAME = 278,
DISP_FILLED_RECTANGLE = 223,
DISP_FORMATTED_TEXT = 266,
DISP_FRAME = 277,
DISP_HLINE = 219,
DISP_INVERT_AREA = 245,
DISP_LINE_ABS = 218,

DISP_LINE = 217,
DISP_MOVE_CURSOR = 212,
DISP_PIXELBLOCK_GET = 243,
DISP_PIXELBLOCK_SET = 244,
DISP_RECTANGLE = 221,
DISP_RESERVE_BMP_COLORS = 269,
DISP_SET_BACKGROUND_COLOR = 231,
DISP_SET_COLOR = 229,
DISP_SET_CURSOR_POSITION = 210,
DISP_SET_DRAWMODE = 237,
DISP_SET_FILLSTYLE =239,
DISP_SET_FONT = 264,
DISP_SET_LINESTYLE = 233,
DISP_SET_LINEWIDTH = 235,
DISP_SET_PALETTE_ENTRY = 268,
DISP_SET_PIXEL_ABS = 214,
DISP_SET_PIXEL = 213,
DISP_SET_TEXTSTYLE = 262,
DISP_TEXT = 260,
DISP_VLINE = 220,
DL_ADD_PLOT_POINT = 1632,
DL_ADJUST_PLOT_AFTER_ROLLING = 1631,
DL_AXIS_ITEM_ADD = 1609,
DL_AXIS_ITEMS_AUTO = 1629,
DL_AXIS_ITEMS_REMOVE = 1610,
DL_AXIS_SET_PARAMETERS = 1605,
DL_DIAGRAM_CONTROL_CURSOR = 1616,
DL_DIAGRAM_DRAW_LINE = 1618,
DL_DIAGRAM_DRAW = 1617,
DL_DIAGRAM_ROLL_LEFT = 1630,
DL_DIAGRAM_SET_PARAMETERS = 1614,
DL_PLOT_DRAW = 1624,
DL_PLOT_EXTEND_RIGHT = 1625,
INPUTLINE_DELETE_CHAR = 1112,
INPUTLINE_DELETE_STRING = 1113,
INPUTLINE_INSERT_CHAR = 1114,
INPUTLINE_INSERT_STRING = 1115,
INPUTLINE_MOVE_CURSOR_LEFT_BORDER = 1110,
INPUTLINE_MOVE_CURSOR_LEFT = 1109,
INPUTLINE_MOVE_CURSOR_RIGHT_BORDER = 1111,
INPUTLINE_MOVE_CURSOR_RIGHT = 1108,

INPUTLINE_SET_CURSOR = 11107,
LCD_CONTRAST = 156,
LCD_LIGHT = 155,
LCD_SLEEP = 157,
LCD_WAKEUP = 158,
LCM_AUTOBAUDING = 15,
LCM_CONTRAST_OFFSET = 40,
LCM_MACRO_KILL = 32,
LCM_MACRO_REC = 29,
LCM_PING = 18,
LCM_SILENT = 42,
LCM_SLEEP = 28,
MENU_ACTIVATE_BAR = 1252,
MENU_ACTIVATE_SINGLE = 1201,
MENU_BAR_SET_FLAGS = 1256,
MENU_DEACTIVATE_BAR = 1253,
MENU_DEACTIVATE_SINGLE = 1202,
POPUP_ACTIVATE = 1308,
POPUP_DEACTIVATE = 1310,
POPUP_SET_ENTRY_FLAGS = 1314,
POPUP_SET_FLAGS = 1312,
SEG7_SET_COLORS = 1402,
SEG7_UPDATE = 1405,
SLIDER_ACTIVATE = 1510,
SLIDER_DEACTIVATE = 1511,
SLIDER_REDEFINE = 1508,
SLIDER_SET_POSITION = 1506,
SOUND_CLICK = 608,
SOUND_MAKE_ASYNC = 611,
SOUND_MAKE = 607,
SOUND_OFF = 610,
SOUND_ON = 609,
SOUND_SET_VOLUME = 605,
TGLASS_CALIBRATION_SEQUENCE = 505,
TGLASS_SET_CALIB_PARAMETERS = 503,
T_KEYB_CLOSE = 705,
T_KEYB_OPEN = 704,
T_KEYB_SET_POSITION = 708.

C Nicht unterstützte GUI–Bibiotheksfunktionen

Die folgenden Funktionen der GUI–Bibliothek haben keine Entsprechung im GUI–Interpreter:

system_init,
system_wait_ticks,
system_wait_until,
system_interval_elapsed,
lcd_init,
lcd_close,
lcd_set_viewport,
disp_open,
disp_connect,
disp_disconnect,
disp_make_current,
disp_get_current,
disp_close,
disp_update,
disp_save_properties,
disp_restore_properties,
disp_printf,
tglass_init,
tglass_close,
touch_open,
touch_connect,
touch_disconnect,
touch_close,
touch_update,
sound_open,
sound_connect,
sound_disconnect,
sound_close,
sound_update,
piezo_init,
piezo_close,
keyboard_update,
events_append_event,
events_collect,
events_encoder_collect

D Häufige Fragen und Troubleshooting

Hier finden Sie Antworten auf häufig gestellte Fragen und Hilfe bei häufig auftretenden Problemen.

1. **Das Autobauding funktioniert direkt nach dem Einschalten nicht :**

   Die LCM–Geräte sind erst nach 500 ms bereit zum Autobauding.

2. **Alles funktioniert, aber ein bestimmtes Grafikelement oder Text wird nicht dargestellt:**

   Überprüfen Sie, ob die Größe und Positionierung so ist, daß das Element vollständig auf dem Display dargestellt werden kann.

3. **Könnte man nicht noch das Feature xy im GUI–Interpreter haben ?**

   Antwort: Eventuell. Wenn es sich um eine Eigenschaft von großem allgemeinem Interesse handelt, können wir so eine Erweiterung vornehmen.

4. **Wie kann ich andere Zeichensätze bekommen ?**

   Der GUI-Interpreter verwendet Zeichensätze im FNT–Format. Mit einem geeigneten Fonteditor können Sie eigene Fonts erstellen oder bereits vorhandene Fonts in dieses Format umwandeln.

5. **Nach manchen Befehlen verhält sich der GUI-Interpreter seltsam ?**

   Wurde lange genug auf die Abarbeitung des Befehls gewartet? Ist der Timeout des Masters groß genug?

6. **Ich erhalte nicht die erwarteten Resultate, wenn ich bestimmte Befehle sende. Z.B. das Kommando `disp_get_color()` scheint keine vernünftige Farbe zu liefern.**

   Stellen Sie sicher, daß ausführliche Antwortpakete angefordert werden (vgl. Abschnitt 2.3 auf Seite 10).

   Im Kurzantwortmodus, bei dem nur 0x00 bzw. 0xFF als Antwort gesendet wird, machen alle Befehle, die ansonsten einen Status oder andere Ergebnisse im Antwortpaket zurück senden, keinen Sinn.

Sollten wichtige Fragen offen geblieben sein, können Sie diese auch direkt an uns stellen.

E Zeichensätze

Für die im GUI-Interpreter bzw. dem LCM integrierten Zeichensätze wurden die europäischen Zeichen nach ISO-8859-1 zugrundegelegt. Für manche Zwecke ist es günstig, noch zusätzliche Pfeile zur Verfügung zu haben, z. B. für Tastaturen. Solche Pfeile wurden zusätzlich unter den Hex-Codes 0x1C - 0x1F abgelegt.

Die Zeichensätze stehen in folgenden drei Größen zur Verfügung:

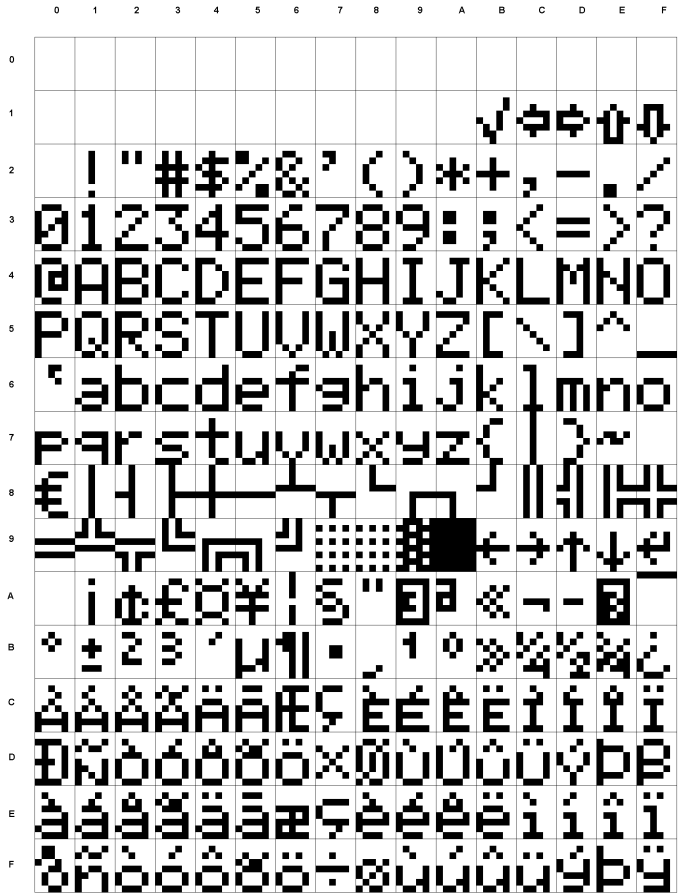


Abbildung 13: Font Mono6x8

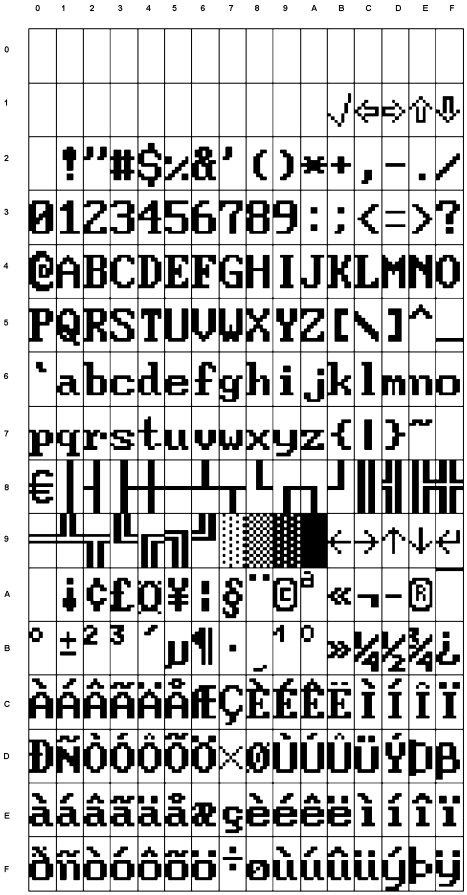


Abbildung 14: Font Mono8x16

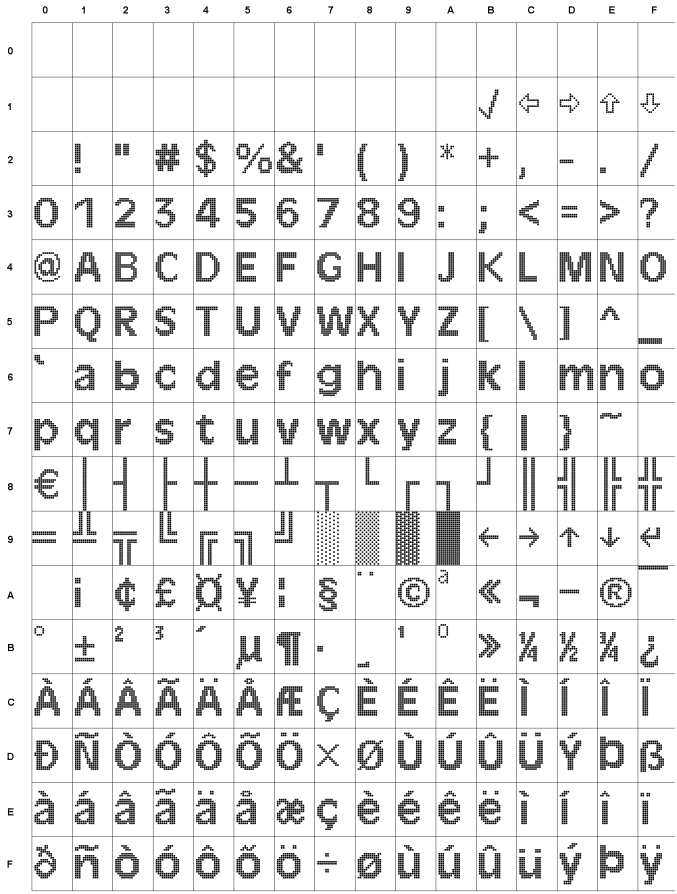


Abbildung 15: Proportionalfont Prop14

F Lizenz

Die spezifischen Eigenschaften und die Art, wie die Software von Simplify Technologies, die auf den und für die LCM2-Module geliefert wird, in der Anwendung eingesetzt werden kann, werden in unserer ”Allgemeinen Softwarelizenz” berücksichtigt, die als ausschließliche Basis für die Benutzung der Software vereinbart wird. Vor der Registrierung und schriftlichen Annahme der Lizenz ist die mitgelieferte Software eine ”Testversion” und darf nur mit den entsprechenden Einschränkungen gemäß der Lizenzbedingungen genutzt werden. Wenn Sie Fragen zur Lizenz haben, helfen wir gerne weiter.

Bitte akzeptieren Sie unsere ”Allgemeinen Softwarelizenz” für die Lizensierung unserer Software durch Unterzeichnung des Registrier-Formulars und senden Sie das Formular per Brief oder Fax an:

Simplify Technologies GmbH
Steinbuehlstraße 15
35578 Wetzlar
Tel.: (+49) (0)6441-210390
FAX.: (+49) (0)6441-210399

```
Allgemeine Softwarelizenz der Simplify Technologies GmbH

Die Firma Simplify Technologies GmbH, Steinbühlstr. 15, D-35578 Wetzlar (im
folgenden Lizenzgeber genannt) gewährt dem Lizenznehmer Rechte zur Benutzung der
vorliegenden Software, im folgenden "Software" genannt, gemäß den folgenden
Bedingungen. Durch Benutzung, Beschaffung oder Verarbeitung der Software werden
diese Bedingungen anerkannt.

Definitionen: Die Software kann in folgender Form vorliegen:

"Ausführbare Programme" bezeichnet Software, die direkt auf einer dafür
vorgesehenen Zielhardware ausgeführt werden kann, eventuell unter Zuhilfenahme
einer Laufzeitumgebung, oder die direkt als Teil eines ausführbaren Programmes
(Laufzeit-Bibliothek) genutzt wird. Hierzu zählt auch Software, die direkt auf
vom Lizenzgeber gelieferten Systemen implementiert ist ("embedded software").

"Sourcecode" bezeichnet Software, in der Form, in der sie für den Menschen
lesbar sind und bearbeitet werden können. Typischerweise ist Sourcecode nicht
direkt ausführbar, sondern wird erst durch geeignete Übersetzung oder
Interpretation zu ausführbarer Software, d.h. zu ausführbaren Programmen, oder
Teilen davon.

"Testversion" bezeichnet Software, die dem Lizenznehmer zu Testzwecken
zur Verfügung gestellt wird.

"Beispiel-Software" bezeichnet Software, die als Beispiel für weitere
Entwicklungen des Lizenznehmers im Zusammenhang mit Produkten des Lizenzgebers
und zur Anleitung dienen soll.

"Käuflich erworbene Software" bezeichnet Software, für die die Lizenz separat
von anderen Produkten und Leistungen des Lizenznehmers gegen Zahlung eines
vereinbarten Preises zu erwerben ist, oder die zusammen mit und als Bestandteil
```

eines vom Lizenzgeber käuflich erworbenen Produktes geliefert wird.

1. Urheberrecht und Leistungsumfang
a) Der Lizenzgeber hält als Inhaber und Verfügungsberechtigter das nach §§ 69 a ff. UrhG und nach internationalen Vereinbarungen geschützte Softwarerecht. Der Lizenznehmer erkennt den vorstehend genannten Schutz ausdrücklich an.
b) Für kostenfrei bereitgestellte Software und Testversionen gilt: Die Software wird zur Verfügung gestellt, wie sie ist ("as is"). Eigenschaften werden nicht zugesichert.
c) Für käuflich erworbene Software gilt: Der Lizenznehmer erhält alle Unterlagen und Sachen zur Durchführung dieses Vertrages sowie eine angemessene Dokumentation der Software. Die Lieferung erfolgt auf einem Datenträger oder per Datenfernübertragung, die Dokumentation je nach Lieferumfang auch in gedruckter Form.
d) Das Eigentum sowie die Inhaberschaft an geistigen Eigentumsrechten jeder Art sowie an Know-how behält sich der Lizenzgeber vor. Jede Weitergabe an Dritte, soweit nicht nachfolgend gestattet, ist nicht erlaubt.

2. Lizensierte Rechte
Dem Lizenznehmer werden folgende Rechte eingeräumt:
a) Der Lizenznehmer erwirbt ein nicht ausschließliches Nutzungsrecht an der Software. Dieses ist zeitlich unbegrenzt, außer im Fall von "Testversionen", wo es zeitlich auf 30 Tage nach der Beschaffung der Software beschränkt ist.
b) Die Software wird nicht lizensiert für folgende sicherheitskritische Anwendungen: Anwendungen in der Luft-, Raumfahrt-, der Militärtechnik und der Nukleartechnik, Anwendungen mit ionisierender Strahlung, Laser- oder Maserstrahlen, Anwendungen zur Beeinflussung der Bewegung von Fahrzeugen, Anwendungen in der Verkehrssicherheitstechnik (z. B. Airbags, Anti-Blockiersysteme), Anwendungen in Lebenserhaltungssystemen, insbesondere in der Medizintechnik, Anwendungen, bei denen im Versagensfall gefährliche Substanzen freigesetzt werden.
c) Für käuflich erworbenen Sourcecode gilt:
  c1) Der Lizenznehmer darf die Software auf Einzelplatzrechnern oder in Netzwerken innerhalb und am Ort seiner Organisation an einem Standort installieren und mit bis zu 5 Nutzern benutzen.
  c2) Die Software oder Teile davon dürfen vom Lizenznehmer mit beliebigen Entwicklungswerkzeugen in eine ausführbare Form übersetzt und so in dessen Produkte übernommen werden und als fester Bestandteil dieser Produkte zusammen mit diesen veräußert werden, sofern es sich nicht um Entwicklungswerkzeuge für Display-Anwendungen Dritter handelt und die Software nicht Bestandteil eines Entwicklungswerkzeuges wird.
  c3) Der Lizenznehmer ist berechtigt, die Software im Sourcecode zu modifizieren. Die Rechte auch der modifizierten Versionen liegen beim Lizenzgeber, auch ansonsten unterliegen die modifizierten Versionen diesem Lizenzvertrag. Über beabsichtigte Veränderungen und Verbesserungen hat der Lizenznehmer den Lizenzgeber zu informieren und diesem rechtzeitig vor der Produktfertigstellung ein kostenloses Probeexemplar zukommen zu lassen.
  c4) Eine Weitergabe der Software im Quelltext und in linkfähigen Objektformaten an Dritte ist nicht gestattet, auch nicht in Teilen.
d) Für "Beispiel-Software" gilt:
  d1) Der Lizenznehmer darf die Software auf Einzelplatzrechnern oder in Netzwerken innerhalb und am Ort seiner Organisation an einem Standort installieren und mit bis zu 5 Nutzern der Software an einem Standort ausschließlich im Zusammenhang mit Produkten des Lizenzgebers benutzen.
  d2) Soweit die Beispiel-Software aus Sourcecode besteht, darf sie oder Teile davon vom Lizenznehmer mit beliebigen Entwicklungswerkzeugen in eine ausführbare Form übersetzt und so in dessen Produkte übernommen werden und als fester Bestandteil dieser Produkte zusammen mit diesen veräußert werden, sofern es sich nicht um Entwicklungswerkzeuge für Display-Anwendungen Dritter handelt, die Software nicht Bestandteil eines Entwicklungswerkzeuges wird und die Software in Zusammenhang mit Produkten des Lizenzgebers eingesetzt wird. Wenn die Software eine Testversion ist, besteht das Recht, diese zu verkaufen, auch im Zusammenhang mit Produkten des Lizenzgebers, nicht.
  d3) Soweit die Beispiel-Software aus Sourcecode besteht, ist der Lizenznehmer berechtigt, die Software im Quellcode zu modifizieren. Die Rechte auch der modifizierten Versionen liegen bei dem Lizenzgeber, auch ansonsten unterliegen die modifizierten Versionen diesem Lizenzvertrag. Über beabsichtigte Veränderungen und Verbesserungen hat der Lizenznehmer den Lizenzgeber zu informieren und diesem rechtzeitig vor der Produktfertigstellung ein kostenloses Probeexemplar zukommen zu lassen.

e) Für "Ausführbare Programme" gilt:
e1) Sofern es sich bei der Software nicht um Beispielprogramme handelt gilt: Der Lizenznehmer darf die Software auf einem Einzelplatzrechner und am Ort seiner Organisation an einem Standort installieren und nutzen. Außerdem darf der Lizenznehmer bis zu zwei Sicherheitskopien, ausschließlich zur Archivierung, anlegen.
e2) Bezüglich der Rückentwicklung und Decompilierung (Reverse Engineering) gilt das Gesetz über Urheberrecht und verwandte Schutzrechte.
e3) Wenn die Software mit dem .NET-Framework der Firma Microsoft erstellt wurde, gilt bezüglich der von Microsoft bereitgestellten Komponenten der entsprechende, in diesem Fall der Software beigelegte "Endbenutzer-Lizenzvertrag für Microsoft- Software". Die von Microsoft bereitgestellten Komponenten dürfen vom Lizenznehmer, der die Software mit den vom Lizenzgeber hergestellten Produkten weitervertreibt, ausschließlich in Verbindung mit und als Teil der vertragsgegenständlichen Software weitervertrieben werden, wobei die Bestimmungen des "Endbenutzer-Lizenzvertrag für Microsoft-Software" einzuhalten sind.

3. Vergütung
Für "Käuflich erworbene Software" gilt: Für die lizensierten Rechte ist ein Entgelt gemäß Rechnungsstellung zu entrichten. Es gelten die jeweils aktuellen Preislisten bzw. Angebote. Die Lizenz gilt erst zu dem Zeitpunkt als erteilt, in dem die vollständige Bezahlung erfolgt ist.

4. Einschränkungen
a) Sämtliche Schutz-, Urheber- und geistigen Eigentumsrechte liegen beim Lizenzgeber oder dessen Lieferanten, insbesondere die für Programme, Software, Texte, Bilder, Animationen, Audiodaten. Alle nicht ausdrücklich in dieser Lizenz gewährten Rechte bleiben dem Lizenzgeber bzw. dessen Lieferanten vorbehalten.
b) Der Lizenzgeber behält sich zukünftige Änderungen an der Software vor.

5. Beendigung der Lizenz und Geheimhaltung
a) Der Lizenznehmer kann diese Softwarelizenz jederzeit durch vollständiges Löschen der Software von seinem Computer bzw. Netzwerk beenden.
b) Die Lizenz endet automatisch, wenn der Lizenznehmer gegen die Bestimmungen der Lizenz verstößt. In einem solchen Fall ist der Lizenznehmer verpflichtet, sämtliche Kopien der Software mit allen ihren Komponenten zu vernichten und vom Lizenzgeber erhaltene Produktunterlagen, Dokumentationen und schriftlich fixiertes Know-how an den Lizenzgeber zurückzugeben. Ein Zurückbehaltungsrecht steht dem Lizenznehmer nicht zu.
c) Für "käuflich erworbene Software" und "Beispiel-Software" gilt: Mit Beendigung der Lizenz erlischt auch das Recht, Produkte, die diese Software oder Teile davon in ausführbarer Form enthalten, zu veräußern.
d) Die Vertragsparteien verpflichten sich, alle in diesem Vertragsverhältnis erlangten Informationen über den Vertragspartner unbefristet geheim zu halten. Das gilt neben den Kenntnissen über die Produkt- und Geschäftspolitik sowie Vertriebswege besonders für alle Informationen, die als vertraulich bezeichnet werden oder als Betriebs- und Geschäftsgeheimnisse erkennbar sind.
e) Die Vertragspartner verpflichten eingeschaltete Dritte, ebenfalls diese Pflichten zu erfüllen.

6. Beschränkte Gewährleistung
a) Der Lizenzgeber haftet nicht dafür, daß die lizensierte Software bestimmte Leistungsergebnisse herbeiführt. Das gilt auch für die Gebrauchsfähigkeit des Softwarepakets zu dem vereinbarten oder einem anderen Zweck. Das Risiko der wirtschaftlichen Verwertbarkeit liegt beim Lizenznehmer.
b) Für "käuflich erworbene Software" gilt:
b1) Der Lizenzgeber leistet Gewähr dafür, daß die Software im wesentlichen den in der Dokumentation beschriebenen Funktionen entspricht und daß die Software frei von Mängeln ist, die den Wert oder die Gebrauchsfähigkeit der Software zu dem vertraglich vorausgesetzten Zweck aufheben oder mindern. Unerhebliche Abweichungen oder Minderungen bleiben außer Betracht. Die vorstehende Gewährleistung bezieht sich nicht auf Mängel, die auf Veränderungen der Software durch den Lizenznehmer zurückzuführen sind.
b2) Rügt der Lizenznehmer einen oder mehrere Mängel, so ist der Lizenzgeber dazu berechtigt und verpflichtet, den/die Mängel auf seine Kosten zu

beseitigen oder gleichwertigen Ersatz zu liefern.
b3) Scheitern Nachbesserungsversuche, so hat der Lizenznehmer das Recht, nach seiner Wahl entweder das vereinbarte Entgelt herabzusetzen oder vom Vertrag zurückzutreten.
c) Für nicht käuflich erworbene Software gilt:
Es werden ausdrücklich keine Eigenschaften der Software zugesichert. Auch die Beschreibung in der Dokumentation ist keine zugesicherte Eigenschaft. Der Lizenzgeber übernimmt keine Gewähr dafür, daß die Software fehlerfrei ist, oder den Anforderungen des Lizenznehmer, auch in Zusammenhang mit anderen vom Lizenznehmer verwendeten Programmen, genügt.
d) Für Beispiel-Software gilt:
Es werden ausdrücklich keine Eigenschaften der Software zugesichert. Auch die Beschreibung in der Dokumentation ist keine zugesicherte Eigenschaft. Der Lizenzgeber übernimmt keine Gewähr dafür, daß die Software fehlerfrei ist, oder den Anforderungen des Lizenznehmers, auch in Zusammenhang mit anderen vom Lizenznehmer verwendeten Programmen, genügt.
e) Die Gewährleistungsfrist beträgt 12 Monate bei gewerblichen Kunden und juristischen Personen und 24 Monate bei Verbrauchern. Die Verjährung beginnt mit der Ablieferung der vertragsgegenständlichen Software.

7. Beschränkte Haftung und Freistellung
a) Bei Software als einem naturgemäß komplexen Produkt kann nicht davon ausgegangen werden, daß sie fehlerfrei ist. Die Software ist nicht für die Verwendung in Produkten oder auf eine Art und Weise geeignet, die bei fehlerhafter Funktion Schäden verursachen könnte. Dies gilt insbesondere für die Verwendung der Software im Bereich sicherheitskritischer Anwendungen wie z. B. der Medizintechnik, der Luft-, Raumfahrt- und Verkehrstechnik, der Nukleartechnik und Militärtechnik. Der Lizenznehmer erkennt dies durch die Benutzung der Software an. Für die Folgen aus der Benutzung der Software ist der Lizenznehmer selbst verantwortlich. Der Lizenznehmer ist verpflichtet, die Eignung der Software und die korrekte Funktion für die jeweiligen Anwendungszwecke selbst zu verifizieren und sicherzustellen.
b) Eine Haftung ist, soweit nicht der Lizenzgeber oder seine Erfüllungsgehilfen vorsätzlich oder grob fahrlässig gehandelt hat, dem Grunde nach ausgeschlossen. Schadenersatzansprüche des Kunden, gleich aus welchem Rechtsgrund, insbesondere aus Vertragsverletzung, aus der Verletzung von Pflichten bei Vertragsverhandlungen und aus unerlaubter Handlung sind ausgeschlossen. Dies gilt nicht, soweit z. B. nach dem Produkthaftungsgesetz oder in den Fällen des Vorsatzes, der groben Fahrlässigkeit, des Fehlens zugesicherter Eigenschaften oder der Verletzung wesentlicher Vertragspflichten, sowie bei dem Lizengeber zurechenbaren Körper- und Gesundheitsschäden oder bei Verlust des Lebens zwingend gehaftet wird.
c) Insbesondere wird die Haftung auch für folgende Schäden ausgeschlossen: Der Lizenzgeber haftet nicht für Datenverluste. Der Lizenznehmer weiß, dass er zu regelmäßiger Datensicherung verpflichtet ist. Ausgeschlossen wird auch die Haftung für entgangenen Gewinn, Betriebsunterbrechung, Verlust geschäftlicher Informationen oder irgendeinem anderen Vermögensschaden aus der Benutzung der Software oder aus der Tatsache, daß sie nicht benutzt werden kann. Ebenso ist die Haftung für unmittelbare oder mittelbare Schäden aus der Benutzung der Software bzw. der Unmöglichkeit, die Software zu benutzen, ausgeschlossen. Die Haftung für unvorhersehbare, untypische Schäden, sowie für Folgeschäden ist ebenfalls ausgeschlossen. Dies gilt auch dann, wenn der Lizenzgeber auf die Möglichkeit solcher Schäden hingewiesen worden ist. Eine Änderung der Beweislast zum Nachteil des Kunden ist mit dieser Regelung nicht verbunden. Der Lizenzgeber empfiehlt dem Lizenznehmer, seine Datenbestände regelmäßig zu sichern, und die Ergebnisse seiner Arbeit zu überprüfen.
d) Unter keinen Umständen übersteigt der Haftungsbetrag die für die Software bezahlte Lizenzgebühr.
e) Die Haftungsbeschränkung gilt auch für Mitarbeiter, Vertreter, Erfüllungsgehilfen und Lieferanten des Lizenzgebers.
f) Der Lizenznehmer stellt den Lizenzgeber von Ansprüchen Dritter aus Produkthaftung frei.
g) Der Lizenznehmer haftet für alle Angaben und Behauptungen, die er bei Vertrieb und Werbung aufstellt.

8. Schutzrechte Dritter
a) Der Lizenzgeber geht ausschließlich für den Bereich der Bundesrepublik Deutschland davon aus, daß der vertragsgemäße Gebrauch der Software keine Schutzrechte Dritter beeinträchtigt. Beeinträchtigt er dennoch die Schutzrechte Dritter, haftet der Lizenzgeber gegenüber diesen Dritten für den Bereich der

Bundesrepublik Deutschland. Für eine Verletzung von Schutzrechten Dritter außerhalb der Bundesrepublik Deutschland übernimmt der Lizenzgeber keine Gewährleistung. Die Überprüfung, ob die Software außerhalb der Bundesrepublik Deutschland eingesetzt werden darf, obliegt dem Lizenznehmer.
b) Der Lizenznehmer benachrichtigt den Lizenzgeber unverzüglich, wenn Dritte Schutzrechtsverletzungen geltend machen. Der Lizenzgeber trägt die Kosten für rechtliche Auseinandersetzungen um Schutzrechte für den Bereich der Bundesrepublik Deutschland. Der Lizenzgeber entscheidet über die rechtlichen Abwehrmaßnahmen sowie bei Vergleichsverhandlungen. Kosten für rechtliche Auseinandersetzungen für den Bereich außerhalb der Bundesrepublik Deutschland trägt der Lizenznehmer.
c) Beeinträchtigt eine vertragsgemäße Nutzung die Schutzrechte Dritter, im Bereich der Bundesrepublik Deutschland, hat der Lizenzgeber, unter Berücksichtigung der besonderen Umstände des Lizenznehmers, die Wahl, ob er die Lizenz erwirbt, die Software ändert, sie - eventuell teilweise - austauscht, oder die Lizenz beendet.
d) Räumt der Lizenzgeber nicht die Rechte Dritter für den Bereich der Bundesrepublik Deutschland aus, berechtigt das den Lizenznehmer zum Rücktritt. Bei Zahlungen jeder Art zum Beispiel als Schadenersatz erhält der Lizenznehmer dann einen Anteil von der Lizenzgebühr nach § 3 des Vertrages, wenn der Rechtsinhaber ihn in der Ausübung seiner Lizenz verletzte.
e) Für nicht käuflich erworbene Software entsteht die Haftung des Lizenzgebers aufgrund von Schutzrechten Dritter entsprechend den Voraussetzungen des Vertrages nur im Fall grober Fahrlässigkeit oder bei Vorsatz.

9. Schlussbestimmungen
a) Zu einer Abtretung seiner Rechte aus diesem Vertrag bedarf der Lizenznehmer der schriftlichen Einwilligung des Lizenzgebers.
b) Eine Aufrechnung gegen die Forderung nach Lizenzgebühr kann der Lizenznehmer nur mit anerkannten oder rechtskräftig festgestellten Forderungen erklären.
c) Wenn in dieser Lizenz Regelungen fehlen, gelten diesbezüglich ergänzend zu dieser Lizenz die Allgemeinen Geschäftsbedingungen (AGB) des Lizenzgebers. Ansonsten enthält der Vertrag enthält alle getroffenen Vereinbarungen. Weitere schriftliche oder mündliche Nebenabreden bestehen nicht. Änderungen und Ergänzungen bedürfen der Schriftform.
d) Die Rechtsunwirksamkeit einer Bestimmung berührt die Rechtswirksamkeit der anderen Vertragsteile nicht. Die Vertragsparteien verpflichten sich, eine unwirksame Bestimmung durch eine wirksame Regelung zu ersetzen, die ihr im wirtschaftlichen Ergebnis am nächsten kommt und dem Vertragszweck am besten entspricht.
e) Erfüllungsort ist der Sitz des Lizenzgebers.
f) Gerichtsstand für alle Streitigkeiten, die sich aus oder im Zusammenhang mit dieser Lizenz ergeben, ist, soweit gemäß § 38 ZPO vereinbar, der Sitz des Lizenzgebers. Es gilt ausschließlich deutsches Recht, UN-Kaufrecht ist ausgeschlossen.

Stand: 16.05.2007

G Registrierung

Registrierformular für die Software, die auf den und für die Simplify Technologies LCM2-Module geliefert wird

Wir haben die ”Allgemeinen Softwarelizenz” von Simplify Technologies GmbH erhalten und möchten hiermit die Software der LCM2-Module registrieren und akzeptieren die Regelungen der ”Allgemeinen Softwarelizenz” der Simplify Technologies GmbH.

Firma: _____

Name: _____

Email: _____

Abteilung: _____

Straße: _____

PLZ, Stadt: _____

Land: _____

Tel: _____

Fax: _____

Datum und Unterschrift des hierfür autorisierten Vertreters:

Datum: _____ Firma, Unterschrift: _____