Hacking the Q-Link Source

How to get involved in the Q-Link Reloaded Project

Glenn Holmer
Emergency Chicagoland Commodore Convention
September 27, 2008
(revised October 5, 2008)

A Little History

- QuantumLink was a Commodore-only dialup service that operated from 1985 to 1994
- toward the end of that time, more and more resources were devoted to America On-Line, resulting in degraded performance (random teleport menus, corrupted downloads, etc.)
- several projects to revive it over the years
- Jim Brain, working with Keith Henrickson and others, rolls out a reverse-engineered server (written in Java) at SWRAP 2005 expo

What Still Needs To Be Done

- the rest of the communication protocols need to be decoded in order to implement missing features like file upload/download
- client disk needs to be fully disassembled in order to provide enhancements (higher baud rates, support for running it on other devices)
- enhancements to chat (QAdmin, QGuide)
- web-based administrative interface (partly complete)

Prerequisites

- experience coding Java
- a Java IDE for editing and building
- MySQL for the database
- A working client setup for testing
- GlassFish (for admin interface only)

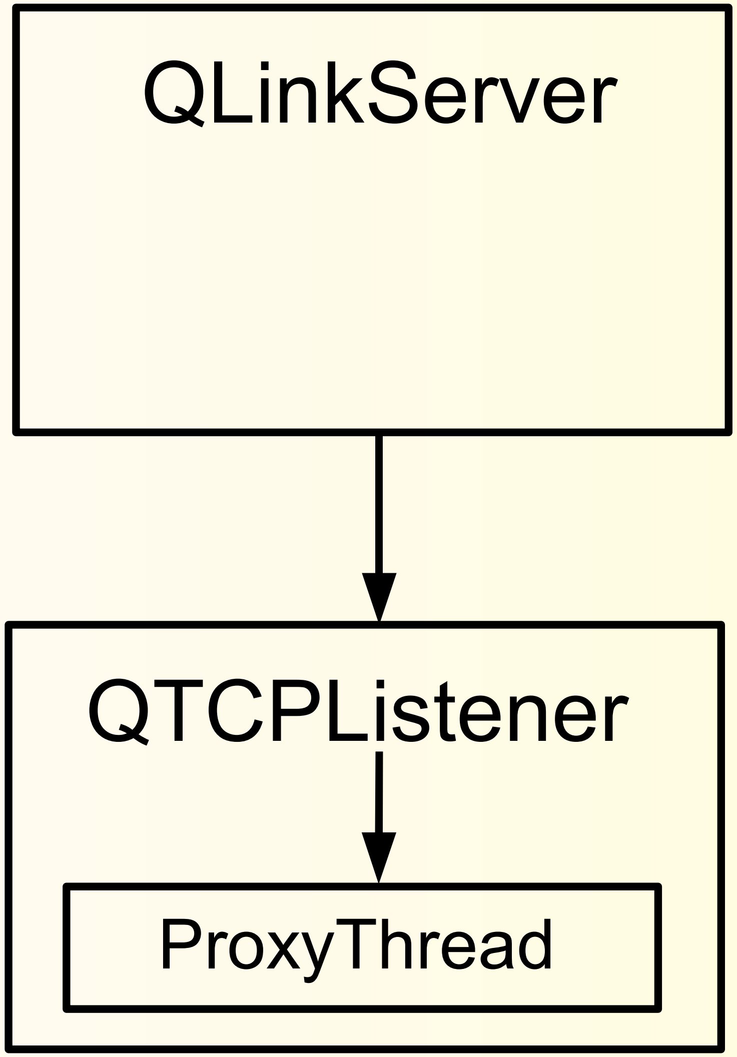
Getting Started

- contact Jim Brain to get access to the source and a test database
- create a project from the source in your IDE and get a test build
- restore the database
- start the server and make sure you can log in
- search the code for the part you want to work on and start hacking!

Server Startup Sequence #1

- the entry point is **`QLinkServer.main()`**, which creates a **`QTCPListener`**
- **`QTCPListener`** starts a thread to wait for incoming connections on a socket
- when a client connects, he starts a **`ProxyThread`**, which creates a **`TelenetProxy`** to emulate a phone connection over the old Telenet dial-up network

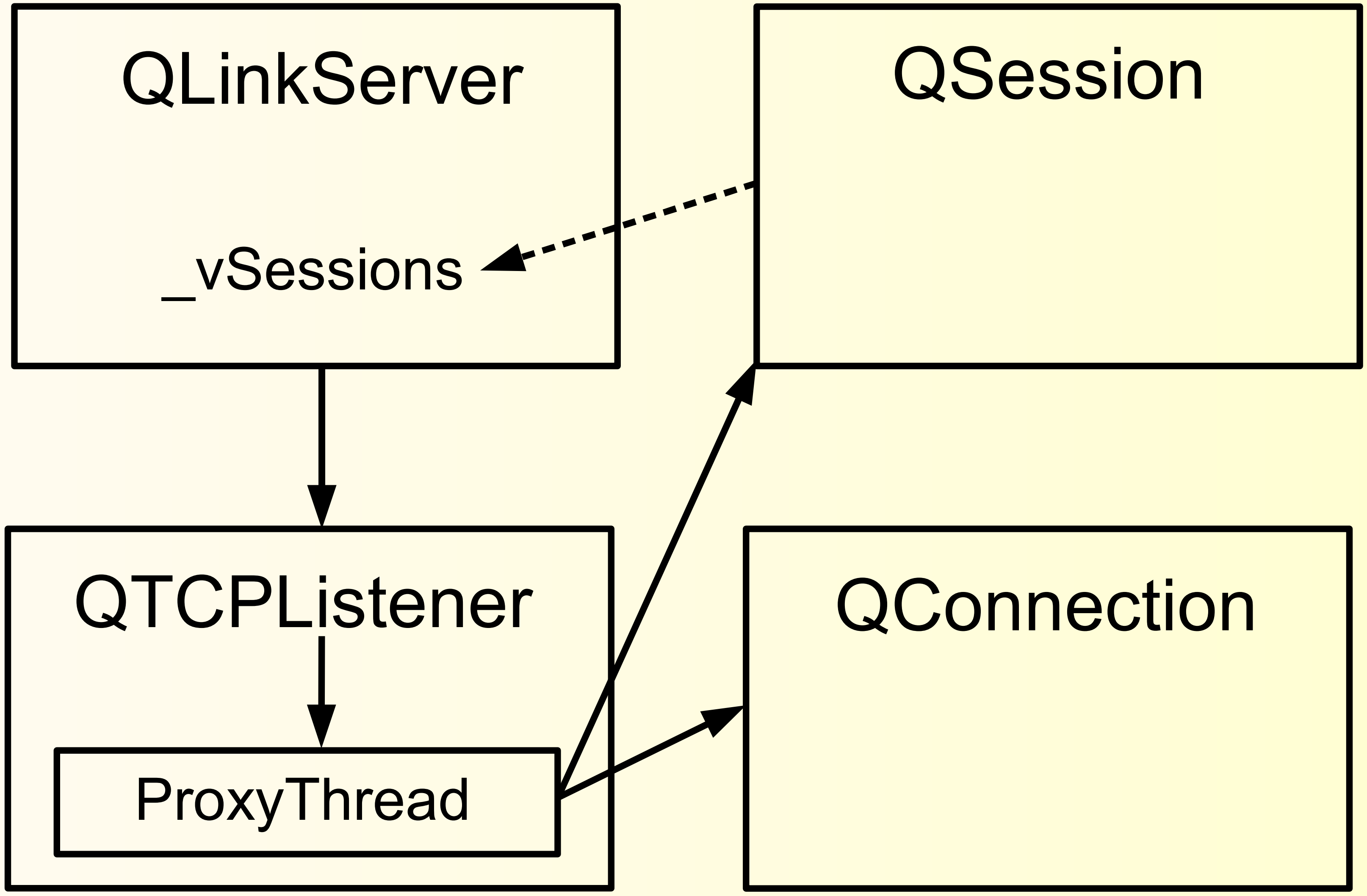
Q-Link Server Diagram



Server Startup Sequence #2

- when connected, **`ProxyThread`** creates a **`QConnection`** (thread) from the socket's input and output streams
- **`ProxyThread`** then creates a **`QSession`**
- **`ProxyThread`** calls the server's **`addSession()`** method, which adds the **`QSession`** to the server's **`_vsessions`** collection

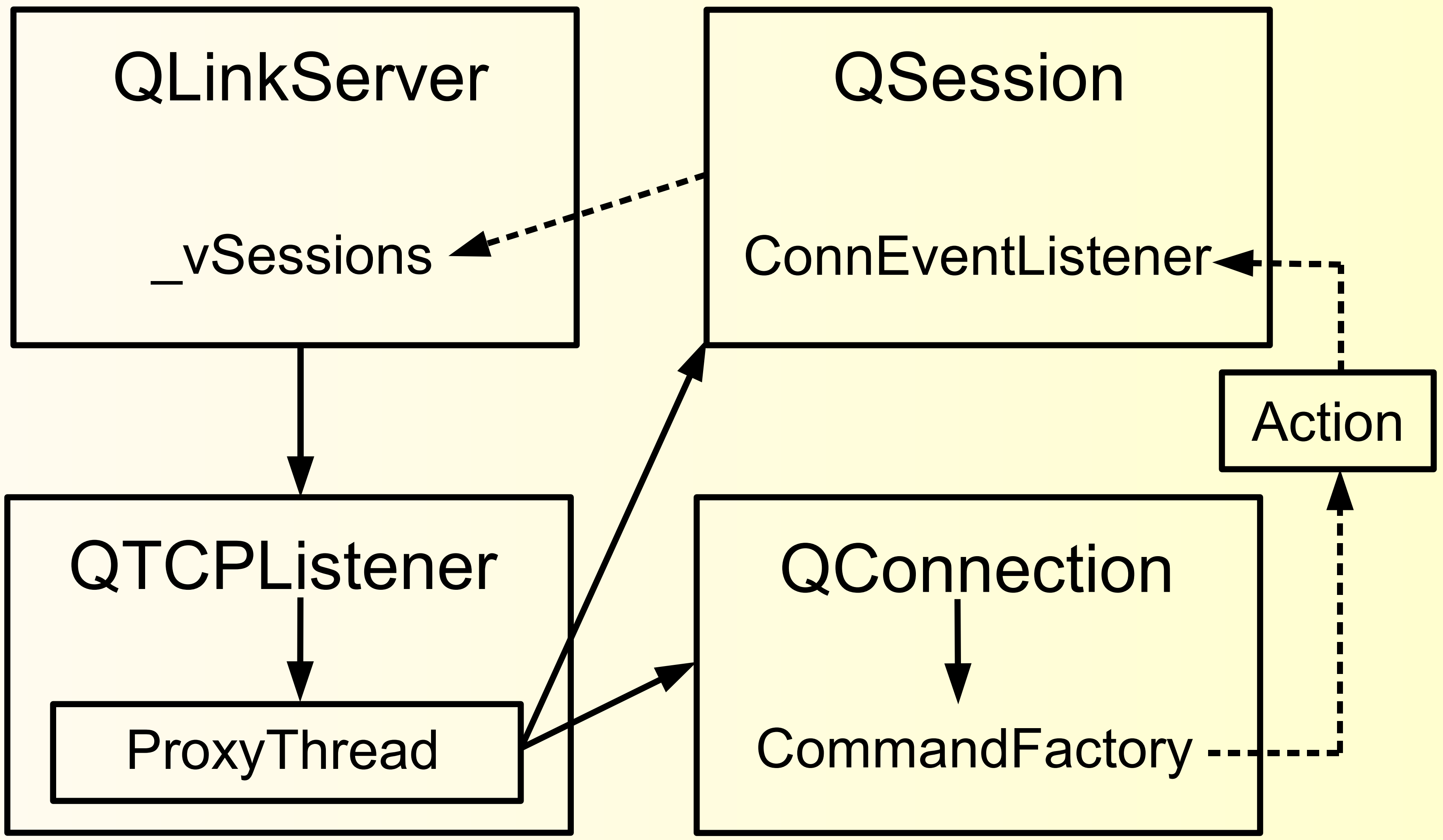
Q-Link Server Diagram



Server Startup Sequence #3

- **`QConnection`**'s constructor creates a **`KeepAliveTask`** (pings the client)
- **`QSession`**'s constructor adds a **`ConnEventListener`** to the **`QConnection`**
- **`QSession`** then starts the **`QConnection`** thread (which creates a **`CommandFactory`**)
- as data are received, **`QConnection`**'s command factory creates **`Command`** and **`Action`** objects; for **`Action`** objects, he fires an **`ActionEvent`**, which **`QSession`** hears

Q-Link Server Diagram



Session Processing

- **`QSession`** is a *state machine*; it has as a member a **`QState`** object
- when the session's listener hears an action event from **`QConnection`**, the state's **`execute()`** method is called
- this can call methods to perform the action, but may also change to a new state (e.g. changing from menu state to chat state)

State Examples

- **`Authentication`**: user is logging in
- **`MainMenu`**: the “sparkle menu”
- **`DepartmentMenu`**: black-on-grey submenus for Commodore Information Network, Software Showcase, message boards, etc.
- **`Chat`**: enter a room, chat, play game, auditorium functions, etc.

State **`execute()`** Example

if the current state is **`DepartmentMenu`** and the action is **`SelectMenuItem`** :

- **`selectItem()`** reads item from database
- if item is a submenu, call **`selectMenu()`** to read **`MenuEntry`** objects from the database
- call **`sendMenu()`** to create **`MenuItem`** objects and send them to the client using the session's **`send()`** method

A Word About the Protocol

- **`QConnection`** assembles packets from the raw data stream
- packets include checksum, sequence number, command byte, instruction, and data (among other things)
- instructions are two-byte ASCII strings
- walk up an action's class hierarchy to see how the entire packet is assembled

Protocol Example

- **`QConnection`** reads packets from the client and sends them to its **`CommandFactory`**
- the factory examines the eighth byte, and either creates a command, or if it's 0x20, sends the packet to its **`ActionFactory`**
- the action factory examines the ninth and tenth bytes to see what kind of action to create (e.g. **`K1`** if a menu item was selected)
- the session creates **`MenuItem`** objects with **`KA`** until the last one, which gets **`KB`**

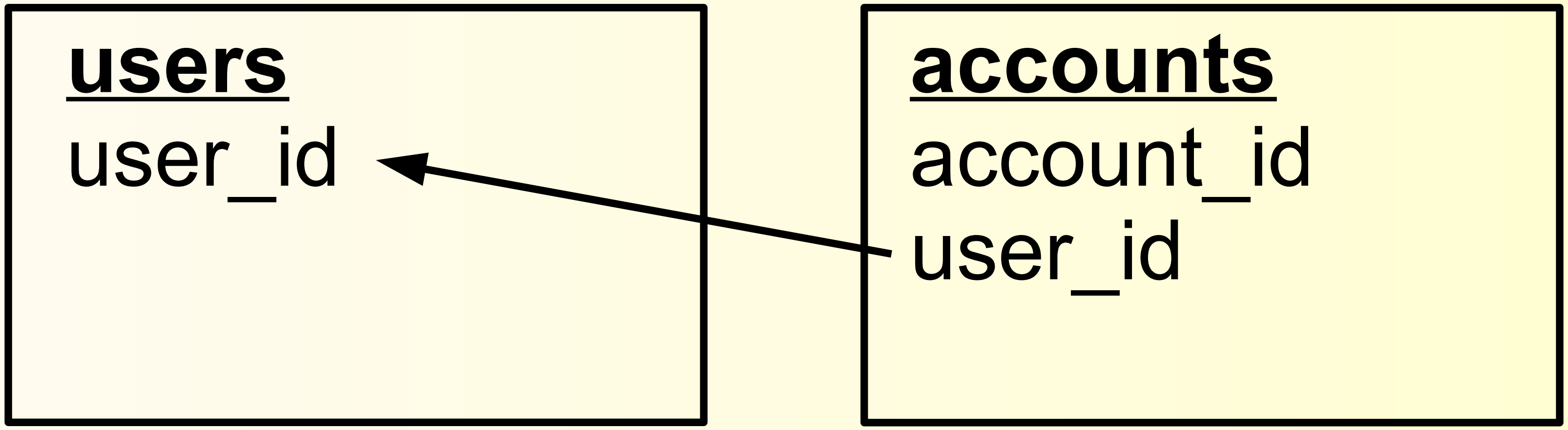
Database Tables

- most objects stored in the database have a master “reference ID”, usually stored as a foreign key to the **`entry_types`** table, which also holds the item's type

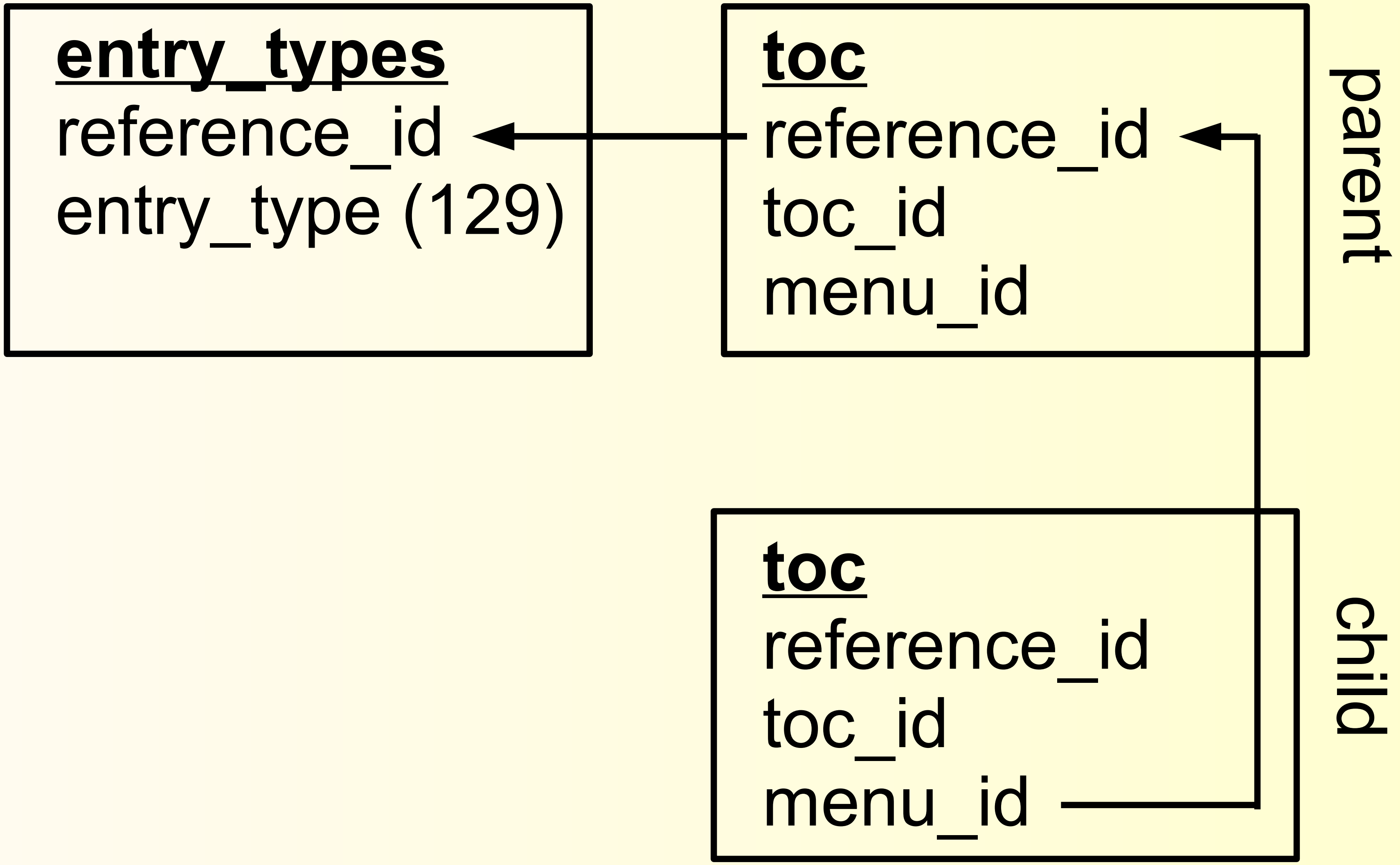
- main groups of tables include:

  `users/accounts` (accounts == handles)
  `toc` (menu items)
  `messages` (message boards)
  `files` (downloads)

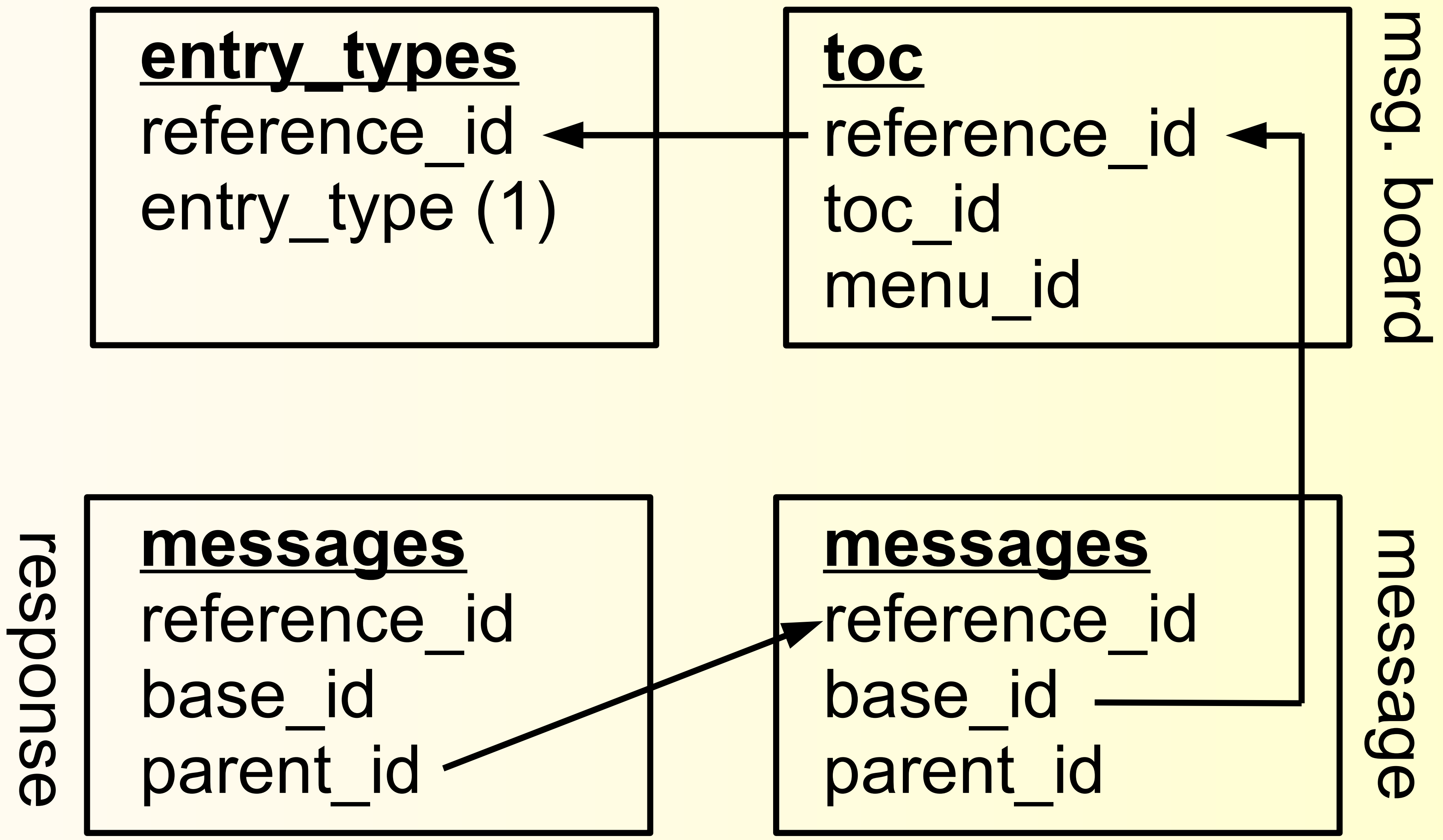
Users/Accounts



Menu Items

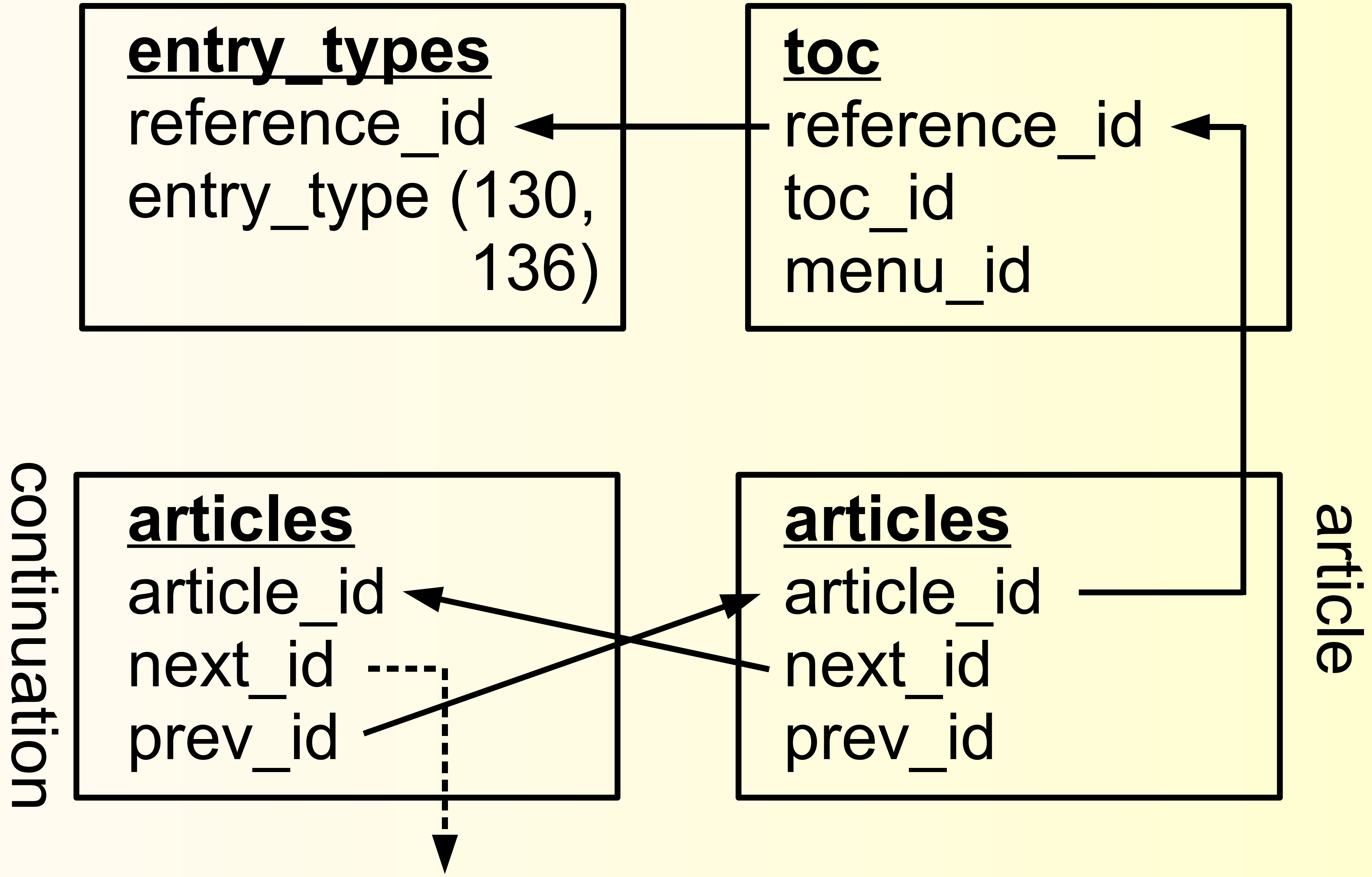

entry_types
reference_id
entry_type (129)
toc
reference_id
toc_id
menu_id
parent
toc
reference_id
toc_id
menu_id
child

Messages

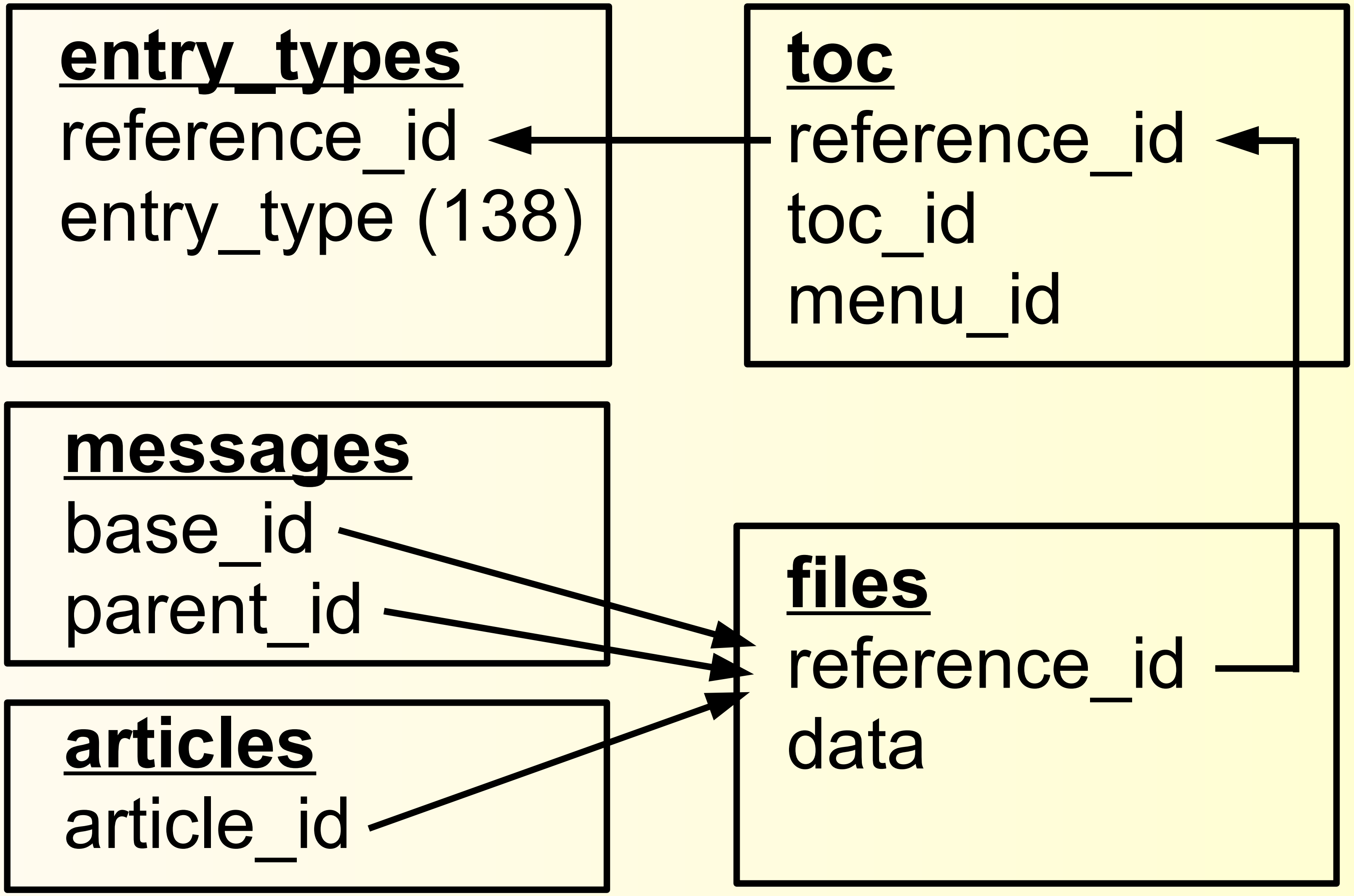


entry_types
reference_id
entry_type (1)
toc
reference_id
toc_id
menu_id
msg. board
messages
reference_id
base_id
parent_id
response
messages
reference_id
base_id
parent_id
message

Articles (single/multiple page)



Files


entry_types
reference_id
entry_type (138)
toc
reference_id
toc_id
menu_id
messages
base_id
parent_id
articles
article_id
files
reference_id
data

Resources / Demo

- Q-Link Reloaded site: http://www.quantumlink.tk
- Q-Link Reloaded message board: http://jledger.proboards19.com

I can demo the following at my table:

- project setup in NetBeans
- a running “closed loop” system
- admin interface

Commodore Software Showcase
+ People + Connection
Commodore Information Network
Q-Link wants
Just For Fun
QUANTUM LINK
YOU!
Customer Service Center
The Mall
RELOADED