# Event Based Communication

* Module Management (Loaded, Unloaded event)
  + Modules Loaded List (Type Classes)
  + Request Load Module or Directory
  + Request Unload Module
  + Is Loaded?
* Settings Management (ChangedModule, ChangedConnection event)
  + XML
  + Database
  + Query Database
  + Version
* Connection Management (NewInstance)
  + Module Instances List
  + Mapped settings to connections
  + Map connections to instances
* Client Interface Manager
  + List of connected Interfaces (Clients)
    - Execution of Interface commands (Using Connection Manager)
    - Local storage of interface registered status tracking (Instance event listener)
* Redundancy Provider  
  -- How to negotiate primary controller
  + Interfaces know the secondary controller
  + Secondary controller should forward requests and responses to and from Interfaces  
    (Where interface may not be able to connect to primary controller)
  + Secondary Controller takes control when either connection to primary is lost
    - Maintains a connection on 2 Ethernet ports
    - Secondary port is for all redundant communication
    - First controller online maintains master status
  + How do we track module state?
  + Emulate device received packets? (Provided by primary controller)

## Settings Change

* Settings Management to Direct settings changes
  + Compare DB to current settings
  + If a change has been made
    - Inform Module manager to load required modules
      * Download new required modules if required
    - Send connection manager the list of connections required
      * Connection manager to inform changing modules of disconnect and then delete instances
      * Connection manage to load new modules and setup connections
      * If connections do not change, only module, then connection maintained (module not informed of disconnect)
    - Inform Module manager to forget / unload unneeded types
    - Delete unneeded modules

## Mainlines

* Mainline Interface + Manager
  + Wrappers for accessing modules (Without local error detection)
    - Status + Command Sending (Through wrapper)
    - Can sign up to events directly
  + Read only access to settings (Modules, server, db, versions)
  + Can register command functions
    - Some commands may require security checks to process
* Security provider (Wrapper for application, directory based and windows authentication)

SendCommand(‘ModuleName’, ‘Command’, params);

Modules(‘Projector’, 1).Send(‘Command’, Params);

Modules(‘Projector.1’).Send

Modules(‘Projector’, ‘1’).Send

Modules(‘Projector’).Send(‘Command’, Params); //Sends to all projectors (Case insensitive)

If Module does not exist it is logged however no error is thrown (handled outside of mainline).

# Coding Standards:

Classes: FirstLetterCapitalised

Instances: firstLetterLowerCase

Function Names: descriptiveWithFirstLetterLower

Constants: ALL\_CAPITALS

Namespaces:

* Control.Module (Device module interfaces)
* Control.Mainline (Mainline module interfaces)
* Control.XML (XML Serialisation helper)
* Control.Settings (Settings Structures for accessing the server and saving xml files)
* Control.Utility.Hex (Hex utility functions)

Threading when possible to maximise CPU usage:

Function Requests, IO all threaded

All communications asynchronous non-blocking

Shared COM ports (For IR devices ect)

## Design of COM and IP communication modules:

Create an interface for IP, COM and USB a user writing a module can then decide how that device is to be connected by including public interfaces

iCOM

iNET

iUSB

iCOM and iNET will supply the default settings for the device which will be loaded if none are provided in the system settings

Both iCOM and iNET will have classes that control and maintain communication between devices and the control system. (Raise event ‘local or to interfaces’)

* Mainline maintainer class can hold all the module information including links to the local modifiers
  + Raise events (‘This will be global’)
  + Disconnect / Connect modules (Redundancy mainline will have to do this)
  + Register macros (events)
    - As functions? Can we do this?

There will also be an iInterfaceProvider which will be attached to an interface maintainer class (which can sync displays as well as pass interface status information to the user)

Use rails, php, .net ect – push from server

Standardise on CHOMP?

Silverlight Interfaces (Super simplified as always .net to .net backend ect)

Flash Interfaces (Much like rails however resides on computer)

.Net Application interface (extended version of Silverlight interface)

Java Interface (Same functionality as .Net)