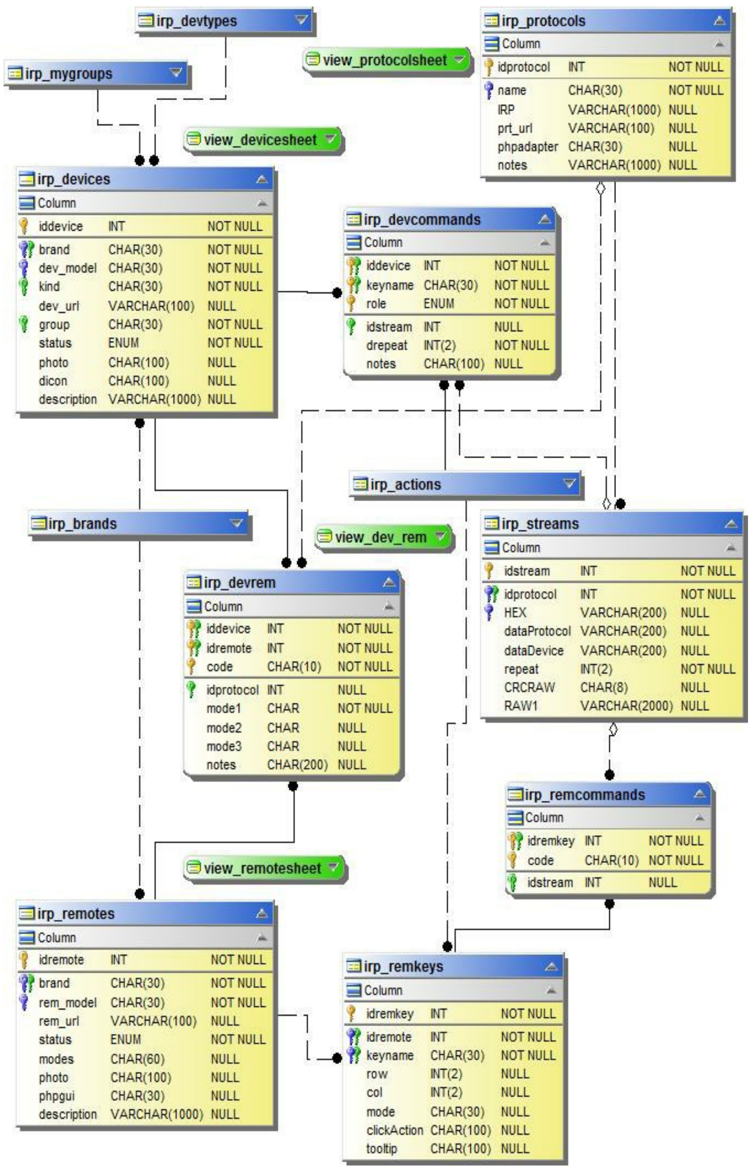
remoteDB *concepts 2*



irp_protocols

Column		
idprotocol	INT	NOT NULL
name	CHAR(30)	NOT NULL
IRP	VARCHAR(1000)	NULL
prt_url	VARCHAR(100)	NULL
phpadapter	CHAR(30)	NULL
notes	VARCHAR(1000)	NULL

irp_devices

Column		
iddevice	INT	NOT NULL
brand	CHAR(30)	NOT NULL
dev_model	CHAR(30)	NOT NULL
kind	CHAR(30)	NOT NULL
dev_url	VARCHAR(100)	NULL
group	CHAR(30)	NOT NULL
status	ENUM	NOT NULL
photo	CHAR(100)	NULL
dicon	CHAR(100)	NULL
description	VARCHAR(1000)	NULL

irp_devcommands

Column		
iddevice	INT	NOT NULL
keyname	CHAR(30)	NOT NULL
role	ENUM	NOT NULL
idstream	INT	NULL
drepeat	INT(2)	NOT NULL
notes	CHAR(100)	NULL

irp_streams

Column		
idstream	INT	NOT NULL
idprotocol	INT	NOT NULL
HEX	VARCHAR(200)	NULL
dataProtocol	VARCHAR(200)	NULL
dataDevice	VARCHAR(200)	NULL
repeat	INT(2)	NOT NULL
CRCRAW	CHAR(8)	NULL
RAW1	VARCHAR(2000)	NULL

irp_devrem

Column		
iddevice	INT	NOT NULL
idremote	INT	NOT NULL
code	CHAR(10)	NOT NULL
idprotocol	INT	NULL
mode1	CHAR	NOT NULL
mode2	CHAR	NULL
mode3	CHAR	NULL
notes	CHAR(200)	NULL

irp_remcommands

Column		
idremkey	INT	NOT NULL
code	CHAR(10)	NOT NULL
idstream	INT	NULL

irp_remotes

Column		
idremote	INT	NOT NULL
brand	CHAR(30)	NOT NULL
rem_model	CHAR(30)	NOT NULL
rem_url	VARCHAR(100)	NULL
status	ENUM	NOT NULL
modes	CHAR(60)	NULL
photo	CHAR(100)	NULL
phpgui	CHAR(30)	NULL
description	VARCHAR(1000)	NULL

irp_remkeys

Column		
idremkey	INT	NOT NULL
idremote	INT	NOT NULL
keyname	CHAR(30)	NOT NULL
row	INT(2)	NULL
col	INT(2)	NULL
mode	CHAR(30)	NULL
clickAction	CHAR(100)	NULL
tooltip	CHAR(100)	NULL

simple lookup tables

irp_devtypes

lookup table for device types (TVset, DVD...) user definite: `kind, ticon.`

irp_mygroups

lookup table to group device (e.g. by location), application dependent: `group, icon.`

irp_actions

lookup table for keys: `keyname, screen, kicon, definition`
You must use, as possible, standard names and standard definitions
standard key: `KEY_NAME` user key: `NEWNAME_KEY`
`screen` is for translations, UI...
see "import/export standard KEY list files"demo tool.

irp_brands

lookup table for electronic brands: `brand, brn_url, bicon`

note: group, keyname, brand are PK, so do not change it

basic entities

irp_devices

code: `iddevice`
classifiers: `brand, dev_model, group, kind`
interface: `dicon, status` *(visible/hidden)*
documentation: `dev_url, photo, description`

irp_remotes

code: `idremote`
classifiers: `brand, rem_model`
interface: `modes, status` *(visible/hidden)*, `phpgui (for status remote GUI)`
documentation: `rem_url, photo, description`

irp_protocols

Required also to differentiate unknown protocols: IRP = NULL is the flag for RAW only 'learn and repeat' strategies.
code: `idprotocol`
classifiers: `name`
IRstream: `IRP, phpadapter` (used with *status remotes*)
documentation: `prt_url, notes`

irp_stream

In case of unknown protocols, the fields HEX, dataProtocol, dataDevices are NULL.
In learning phase (CRCRAW = NULL) you can also set only one field between HEX, dataProtocol, dataDevice and RAW1. (this can also be done using imported *remote sheet*). The "tool fill" demo can calculate all fields (if IRP is known) and sets CRCRAW.
code: `idstream`
ref: `idprotocol`
data: `HEX, dataProtocol, dataDevice, RAW1,`
`repeat` *(n° of repetions stored, usually 1)*
check: `CRCRAW` *(used as index and flag: if not NULL data are ok)*

ralation tables

irp_remkey

This table defines the virtual keys for a remote control, using the `mode` as selector when a real key can have more than one meaning. In case of *dynamic key*, a fake key is also defined for device, using it as status storage, with `row = 0` and `col = 0` and keyname `STATUS_D??_KEY` (?? is `iddevice`).
code: `idremkey`
ref: `idremote, keyname`
position: `row, col`
selector: `mode`
dynamic: `clickAction`
interface: `tooltip`

irp_remcommand

This table links a static remkey to the stream that the key sends, using `code` as selector in case of multi-code remote controls.
ref: `idremkey, idstream`
selector: `code`

ralation tables

irp_devrem

This table match remotes and devices, limiting the match with `code`, in case of multi-code remote controls, and with three accepted modes to select only useful keys.
ref: `iddevice, idremote`
selectors: `code, mode1, mode2, mode3`
ref: `idprotocol`

irp_devcommands

This table identifies all keys/streams a device can receive, using `role` as selector when a device can accept more than one set of commands. This is typically done allowing more than one 'device' (D) value in reception.
The "device update" demo tool can set this table, using information on remote and irp_devrem, but the tool sets default `role ('USE')` and `drepeat (1)..`
ref: `iddevice, keyname, idstream` (can be NULL for dynamic keys).
selector: `role` (*'USE', 'primary', 'broadcast', 'unused'* but it can be changed)
data: `drepeat` (repetitions required by the device)
documentation: `notes`

views

The views gives you a large and flat look on remoteDB.
They can be used by applications as real tables in queries, and this semplifies the application development:

view_remotesheet: all remote contorols and keys and related information, protocol etc.
view_devicesheet: all keys (static and dynamic) and streams that a device can receive
view_protocolsheet: all infos about static keys and streams
view_dev_rem: extends the information in irp_devrem.

The SQL code for all *views* is in dir `/sql/`

stored procedures

fnlimitdeletestream	FUNCTION	delete a stream, cascade with limits
fnsetupdatestream	FUNCTION	sets or update a stream, returns idstream
limitdeleteremkey	PROCEDURE	delete a remkey, cascade with limits
limitdeletestream	PROCEDURE	same as fnlimitdeletestream
remote2device	PROCEDURE	sets irp_devcommands records (used by "device update")
replacestream	PROCEDURE	replace a stream
setstreamkey	PROCEDURE	set/update irp_remcommands, having the key and the stream

The SQL code for all *stored procedures* is in dir `/sql/`
In libraries `irp_remoteDB_tools.php` and `irp_remoteDB_stream.php` you can found php wrappers for any stored procedure.

Notes: repetions

Some protocols may require the sending of multiple IR packets, in this case the information is in IRP.
But even some commands may need repetions, if required by the device or for some purpose, such as a command key: 'VOL +5'

Case learned RAW (no IRP):

`irp_streams.repeat`:
Indicates the number of ripetitions on stored RAW (by modifying some parameters in the Arduino sketch, you can also capture a second IR packet, e.g. a ditto).
`irp_devcommands.drepeat`:
Number of repetion required by the device or command.
if irp_devcommands.drepeat <= irp_streams.repeat => send the stored RAW
if irp_devcommands.drepeat > irp_streams.repeat => merge RAW stored multiple times as single RAW

Case IRP:

`irp_streams.repeat`:
Indicates the number of ripetitions on stored RAW. (e.g. 2 in case of one mandatory ditto, but only for some commands)
`irp_devcommands.drepeat`:
Number of repetion required by the device or command.
if irp_devcommands.drepeat <= irp_streams.repeat => send the stored RAW (fast)
if irp_devcommands.drepeat > irp_streams.repeat => rebuild the RAW using irp_classes, then send it. (slow)