OVLÁDÁNÍ INTELIGENTNÍHO DOMU POMOCÍ CHYTRÝCH HODINEK

Michal Fišar

Diplomová práce

2019

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Fišar** Jméno: **Michal** Osobní číslo: **475453**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Inteligentní budovy**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Ovládání inteligentního domu pomocí chytrých hodinek

Název diplomové práce anglicky:

Intelligent house control using smart watches

Pokyny pro vypracování:

Na základě analýzy provedené v předchozím projektu, realizujte aplikaci umožňující ovládání inteligentního domu řízeného PLC systémem Foxtrot společnosti Teco a.s.. K ovládání využijte hodinky s operačním systémem WatchOS společnosti Apple. Vytvořte demonstrační aplikaci pro mobilní telefon s operačním systémem iOS, která bude pro hodinky zajišťovat komunikaci se systémem Foxtrot přes komunikační server PlcComS. Řešení aplikace navrhněte tak, aby bylo snadno přenositelné do již existující aplikace iFoxtrot. Implementujte vhodné uživatelské rozhraní hodinek pro prezentaci dat získaných z mobilní aplikace. Rozhraní bude poskytovat základní přehled o jednotlivých místnostech v domě. Pro každou místnost pak budou k dispozici z uživatelského hlediska zajímavá data (např. teplota/vlhkost v místnosti, stav topení/chlazení, stavy světel/zásuvek). V optimálním případě budou hodinky také ovládat základní prvky inteligentní domácnosti, jako jsou světla, nebo korekce teploty v místnosti. Při návrhu se zaměřte především na reálnou využitelnost ovládání a uživatelský komfort.

Seznam doporučené literatury:

[1] LACKO, Ľuboslav. Vývoj aplikací pro iOS. Brno: Computer Press, 2018. ISBN 978-80-251-4942-3.
[2] Teco a.s. Programování PLC podle normy IEC 61 131-3 v prostředí Mosaic. https://www.tecomat.cz/download/get/txv00321_01_mosaic_progiec_cz/163/
[3] Teco a.s. Knihovna iControlLib. https://www.tecomat.cz/download/get/txv00359_01_mosaic_icontrollib_cz/163/
[4] Teco a.s. Komunikační server PLCCOmS. https://www.tecomat.cz/download/get/txv13863_01_plccoms_cz/156/

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Jan Martinec, Teco, a.s.

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **05.02.2019** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce:
do konce letního semestru 2019/2020

Ing. Jan Martinec podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání _____ Podpis studenta _____

Prohlášení

Prohlašuji:

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně s použitím uvedené literatury a podkladů. Její výsledky mohou být dále použity podle uvážení vedoucího diplomové práce jako jejího spoluautora.

Souhlasím také s případnou publikací výsledků diplomové práce nebo její podstatné části, pokud budu uveden jako její autor.

V Praze dne 23. 5. 2019

Michal Fišar

Poděkování

Touto cestou bych rád poděkoval vedoucímu práce Ing. Janovi Martincovi. za cenné odborné rady a připomínky, které mi byly velmi nápomocné při kompletaci této práce, a také panu Pavlovi Kohoutkovi, autorovi aplikace iFoxtrot.

Dále bych chtěl poděkovat své přítelkyni Alexandře Bubnové a celé mé rodině za neustálou podporu, při vypracovávání této práce.

V Praze dne 23. 5. 2019

Michal Fišar

ANOTACE

Práce je věnována návrhu a vývoji aplikace na hodinky Apple Watch, pomocí které lze ovládat PLC firmy Teco a.s., umístěného v inteligentním domě. Tato aplikace bude využívat standardy již existující iOS aplikace iFoxtrot tak, aby bylo v budoucnu možné tyto aplikace sloučit.

KLÍČOVÁ SLOVA

ovládání inteligentního domu, chytré hodinky, Apple watch, PLC Teco

TITLE

Inteligent house control using smart watches

ANOTATION

The work focuses on designing and developing an application for smart Apple Watch, which will serve to control a PLC of company Teco a.s., which is located at the house. The application will use standards of already existing iOS app called iFoxtrot so in the future it will be possible to merge them together.

KEYWORDS

Inteligent house control, Smart watches, Apple watch, PLC Teco

Obsah

Seznam zkratek a značek

API	rozhraní pro programování aplikace (application programming interface)
GUI	grafické uživatelské rozhraní (graphical user interface)
IP	internetový protokol (internet protocol)
PLC	programovatelný automat pro řízení technologických procesů (programmable logic controller)
TCP/IP	primární přenosový protokol/protokol síťové vrstvy (transmission control protocol/internet protocol)
UI	uživatelské rozhraní (user interface)

Seznam obrázků

Seznam tabulek

ÚVOD

Výsledkem této práce je návrh a realizace aplikace pro chytré hodinky od firmy Apple, které byly vybrány na základě Projektu 2, která využívá serveru PLCComS, ke vzdálenému ovládání inteligentního domu, osazenému libovolným PLC z řady Foxtrot firmy Teco a.s.

Aplikace je rozdělena na dvě části, z důvodu omezení hodinek samotným výrobcem, proto je komunikace s PLCComS serverem pomocí TCP/IP řešeno výhradně v mobilní části, zatímco hodinky obstarávají GUI.

Kvůli složitějšímu ovládání a menším možnostem, které chytré hodinky nabízejí, se aplikace zabývá ovládáním pouze vybraných funkčních bloků z knihovny iControlLib pro Mosaic.

1 VZDÁLENÉ OVLÁDÁNÍ INTELIGENTÍHO DOMU

V dnešní době je již celkem normální vzdálený přístup k téměř všem zařízením, a proto i tato možnost existuje u inteligentních domů, které jsou osazeny a řízeny pomocí PLC. V této práci se zabývám PLC od firmy Teco a.s., řady Foxtrot.

Teco a.s. nabízí několik možností, jak vzdáleně PLC ovládat. Mezi ty nejzajímavější patří WEB server nacházející se v PLC, služba TecoRoute poskytovaná přímo společností a jako poslední pak server PLCComS. Přičemž většina z těchto možností vyžaduje veřejnou IP adresu pro připojení k zařízení z vnější sítě.

WEB server

Umožňuje vytvořit vizualizaci pro zobrazení v internetovém prohlížeči. Vizualizace se nachází přímo v PLC, respektive na SD kartě. Pro tvorbu této internetové stránky se používá software s názvem Mosaic, který slouží k programování libovolného PLC firmy Teco a.s. a všeho okolo něj. V Mosaicu je k dispozici nástroj jménem WebMaker, ten je určen právě pro tvorbu grafické internetové stránky, přes kterou je možné provádět monitoring a s oprávněním i nastavovat libovolné dostupné proměnné [1].

TecoRoute

Jedná se o službu, která slouží ke vzdálené správě PLC pomocí internetového připojení. V PLC se musí nastavit přihlašovací údaje pro tuto službu, tím se zajistí automatické přihlašování PLC k serverům TecoRoute. Díky tomu tak dochází ke tvorbě pomyslného virtuálního tunelu mezi PLC a klientem. Služba TecoRoute tak zvyšuje dostupnost PLC pro programátory a koncové uživatele. Díky vzdálenému přístupu je tedy možné provádět samotné nastavování PLC, nebo třeba i WEB server, bez nutnosti fyzického přístupu. Navíc díky virtuálnímu tunelu PLC nepotřebuje ani veřejnou IP adresu [2].

1.1 PLCCOMS

Je komunikační server, jenž poskytuje TCP/IP připojení mezi klientským zařízením a PLC. Ke komunikaci serveru s klientem je použit jednoduchý protokol typu dotaz/odpověď. Ze strany klienta dochází k tázání na konkrétní hodnoty proměnných, server PLCComS obratem zjistí požadovanou hodnotu z PLC a navrátí ji straně klienta. Ze strany serveru pak klient automaticky dostává pouze chybové hlášení, respektive varování anebo změny hodnot proměnných, které si klient sám navolil [3].

PLCComS komunikuje s PLC přes protokol EPSNET, respektive v případě SoftPLC pomocí sdíleného modulu a dotazuje se za využívání absolutního adresování v periodě 100 ms [3]. Z tohoto hlediska je tento komunikační server zcela jistě nejvhodnější k použití pro vzdálené ovládání PLC Foxtrot pomocí aplikace.

„Nastavení PLCComS serveru se provádí přes inicializační soubor, který je možno editovat běžným textovým editorem. Nastavuje se v něm mimo jiné adresa a port PLC, ukončení řádku či umístění PUBFILE, což je soubor, který je generován vývojovým prostředím Mosaic při programování PLC.“ [18, s. 20]

1.1.1 Klientský protokol

Jedná se o textově orientovaný protokol. Jakýkoliv příkaz musí být ukončen znakem „:“ a konec řádku pomocí „\r\n“ nebo „\n“ – každý klient si může zvolit, jaký konec řádku si vyžádá, aniž by to mělo vliv na ostatní připojené klienty. Důležitým poznatkem je také fakt, že se nerozlišují velká a malá písmena [3].

Tab. 1.1 – Seznam příkazů pro PLCComS [3]

Příkaz	**Popis**
LIST:\n	Vypíše seznam všech proměnných
SET:<proměnná, hodnota>\n	Nastaví proměnnou v PLC na hodnotu
GET:<proměnná>\n	Získá hodnotu proměnné z PLC
EN:<proměnná> <delta>\n	Povolí proměnnou a nastaví změnu delta
DI:<proměnná> <delta>\n	Zakáže proměnnou a nastaví změnu delta
HIDE:<proměnná>\n	Skryje proměnnou z PUBFILE
UNHIDE:<proměnná>\n	Odkryje proměnnou z PUBFILE
GETMEM:<proměnná velikost>\n	Získá blok paměti z PLC
GETFILE:<soubor>\n	Získá soubor z PLC nebo PC
GETFILEINFO:<soubor>\n	Získá informace o souboru z PLC nebo PC
WRITEFILE:<soubor><vel. bl.>= data\n	Zapíše soubor do PLC nebo PC
WRITEFILEINFO:<soubor><vel. bl.>= data\n	Zapíše informace o souboru do PLC nebo PC
GETINFO:\n	Vypíše informace o serveru
SETCONF:<proměnná, hodnota>\n	Nastaví proměnnou v konfiguraci

1.1.2 Důležité příkazy

Pro ovládání PLC z hodinek samozřejmě není potřeba všech dostupných příkazů, kterými server disponuje. Dále se tak budu zabývat jen těmi co mají pozdější využití ve vztahu s aplikací. Hlavní informace ohledně příkazů a jejich použití je následující:

„Na každý dotaz může, ale nemusí, být vrácen jiný formát odpovědi. Odpověď vždy začíná stejným příkazem jako dotaz.“ [18, s. 21]

Pro zadávání jmen proměnných je možné použít znaku „*“, který nahrazuje neomezený počet libovolných znaků [3]. Lze tak využít např. získání více informací najednou, nebo si usnadnit vypisování celého řetězce s názvem proměnné.

Příkaz LIST:

Pomocí tohoto příkazu se vypíše seznam názvů všech proměnných tak, jak jsou zapsány v PUBFILE PLC. Ke každé proměnné je pak navíc vypsán i její datový typ, v kterém ji PLC uchovává. Tento příkaz slouží k získání základních informací dostupných možností ovládání daného PLC, při správném použití příkazu GET: ho ale lze úplně vynechat [3].

Příkaz GET:

Je jedním z těch nejdůležitějších příkazů, který PLCComS nabízí. Lze ho využít k získání hodnoty jedné konkrétní proměnné, ale i k jednorázovému výpisu celého soupisu PUBFILE (při použití „*“). To však platí pouze za předpokladu, že jsou všechny proměnné povoleny pomocí příkazu EN:*, v opačném případě dochází k pouhému navrácení ukončovacího řádku „GET:\n“.

Typy použití jsou 3:

- GET:*\n vypíše všechny povolené proměnné
- GET:zasuvky*\n vypíše všechny proměnné začínající na „zasuvky“
- GET:zasuvky.z1 vypíše konkrétní zásuvku z1

Zatímco první dva případy navracejí více řádků, přičemž poslední z nich je „GET:\n“ a symbolizuje úspěšné dokončení požadavku, poslední případ navrátí jen jeden řádek, je tedy třeba rozlišovat, mezi víceřádkovým a jednořádkovým voláním této funkce [3].

Příkaz SET:

Tento příkaz slouží k nastavování hodnot vybraných proměnných a při jeho volání je nutné počítat s tím, že nemusí přijít žádná zpětná vazba. Příčinou tohoto stavu je, že samotný příkaz nevrací odpověď, jako tomu je u příkazů ostatních.

Reakce ze strany serveru na tento příkaz tedy mohou být 2:

- Chybové hlášení – za předpokladu, že je příkaz zadán chybně, respektive tedy při snaze o nastavení neexistující proměnné. Server navrací odpověď ve tvaru „ERROR:<specifikace chyby>“.
- Povolená proměnná – tento případ nastane při nastavování proměnné, u která je povoleno vypisování změny stavu (DIFF:). V takovém případě server navrátí odpověď ve tvaru „DIFF:<proměnná>,<hodnota>\n“ [3].

Příkaz EN: a DI:

Při použití příkazu EN dochází k aktivaci automatického výpisu změny stavů zvolené proměnné. Pokud je zadána i delta, automatický výpis se omezí a je zasílán vždy až po dosažení změny delta. Výhodou při používání povolení automatických výpisů změn je příkaz „GET:*\n“, který pak následně vypíše seznam všech hodnot povolených proměnných.

Opačným příkazem je DI, který na bázi stejného principu automatický výpis změny stavu vypíná. [3].

Příkaz GETFILE:

Slouží pro stažení libovolného souboru, nebo celé složky, v případě že jméno souboru končí na „/“, z PLC nebo PC. Získávaný soubor je přenášen v blocích, přičemž maximální velikost bloku je nastavena v konfiguračním souboru pod proměnnou FFILE_BLKSIZE. Délka právě přenášeného bloku je dána v hranatých závorkách. Podobně, jako příkaz LIST:, nebo GET:* i tento příkaz zasílá po odeslání celého souboru řádek ve tvaru GETFILE:jmenoSouboru[0]= a je tedy snadné rozlišit, zdali je soubor kompletní [3].

Dotaz: GETFILE:/www/ifox/scn_74bf.sc1\n

Odpověď: GETFILE:/www/ifox/scn_74bf.sc1[47]=Toto je soubor s konfigurací scény v místnosti.\n

GETFILE:/www/ifox/scn_74bf.sc1[0]=\n

Podobným příkazem je pak GETFILEINFO:, který navrací jednořádkovou odpověď na stejném principu jako GETFILE:. Za „=“ jsou ale informace, které jsou rozděleny mezerou a

jsou zapsány v tomto pořadí – velikost souboru v bytech, atributy, čas vytvoření a čas změny [3].

Díky tomuto příkazu je tak možné dopředu zjistit velikost souboru, v případě, že je nutné řešit omezené místo paměti, popřípadě pak kontrolovat, jestli nedošlo ke změně souboru poté co již byl stažen, kdy velkou výhodou je, že není potřeba stahovat celý soubor pro porovnání.

1.1.3 Chybové zprávy

PLCComS informuje protistranu v případech, kdy se vyskytne problém, nebo když na PLC došlo ke změně, která může být důležitá pro následující správnou komunikaci.

Základní dělení těchto chybových hlášek je podle jejich typu, z kterého pak následně vyplývá hlavička posílaného příkazu. Chybové zprávy začínají pomocí „ERROR:“ zatímco výstražné mají „WARNING:“ [3].

Celá zpráva se skládá z hlavičky, kódu chyby a jejího popisu. Zprávy jsou rozděleny do skupin po deseti dle jejich významu. V tab. 1.2 jsou uvedeny pouze ty nejdůležitější chybové zprávy, podle předpokládaného využití v aplikaci.

Tab. 1.2 – Chybové zprávy PLCComS [3]

Kód chyby	Zpráva	Skupina
10	Unable to conect to PLC.	Síťová komunikace
11	Maximum connections reached.	
20	Unable to get data from PLC.	Komunikace s PLC
21	Unable to send data to PLC.	
30	Bad client request:	Klientské dotazy
31	Incomplete client request.	
32	Unknown command name:	
33	Unknown register name:	
34	Disabled register name:	
35	Wrong parameter value:	
41	Unable to get file:	Souborové operace
1024	Unknown error	Nespecifikováno

Výstražné zprávy jsou pouze dvě a jejich kódové označení je 250 respektive 251. Jeden z těchto kódů je odeslán vždy po obdržení nového CRC, což je aktuální hodnota uživatelského PUBFILE. CRC mění hodnotu pokaždé, když je provedena libovolná změna v Mosaicu a následně použito přeložení. Změna CRC tedy neznamená, že nutně došlo ke změně PUBFILE.

Zatímco warning 250 hlásí opravdovou změnu PUBFILE v PLC, tedy informuje o nutnosti opětovného zjišťování dostupných objektů k ovládání, kód 251 je ze strany PLCComS odeslán v případě, kdy se sice CRC změnilo, ale PUBFILE zůstal nezměněn [3]. Pro funkci aplikace bude nutné reagovat pouze na kód 250.

1.1.4 Zabezpečení

V současné době bohužel PLCComS nedisponuje žádným zabezpečením komunikačního protokolu. Používáním se tak uživatel může vystavit potencionálnímu nebezpečí, kdy útočníkovi stačí získat IP adresu, na které server běží, a může se bezproblémově připojit k ovládání PLC v domě. Výchozí nastavení portu je totiž 5010 a dá se předpokládat, že tento port bude využívat většina uživatelů.

Dle rozhovoru s tvůrcem PLCComS serveru se však někdy během tohoto roku chystá zavedení autorizace, s kterou by mělo přijít i šifrování samotné komunikace. Tato aplikace se však tímto krokem zabývat nebude a pro správnou funkci ji bude potřeba do budoucna doplnit.

1.2 MOSAIC – ICONTROLLIB

Knihovna iControlLib pro Mosaic přidává funkční bloky, které jsou svým využitím převážně pro inteligentní budovy. Bloky jsou připraveny na interakci s webovým rozhraním (web server) nebo aplikací iFoxtrot, tedy jedná se o bloky, které je možné ovládat a sledovat pomocí serveru PLCComS.

V současné době jsou dostupné dvě verze této knihovny – iControlLib_v10 a iControlLib_v20, přičemž druhá ze zmíněných je dostupná od verze Mosaicu v2017.1. Podpora této knihovny není u všech PLC od firmy Teco a.s. [4].

V knihovně je k dispozici 24 funkčních bloků, některé z nich však mají velice podobné funkce, a je tedy mnohem lepší je dělit dle povahy daného zařízení na kategorie. V této práci se zmiňuji pouze o kategoriích, tedy blokách, které mají přínos pro ovládání skrze hodinek.

1.2.1 Bloky scén

Slouží pro nastavování různých scén, scénou je myšleno konkrétní nastavení osvětlení, žaluzií, zásuvek atp. v místnosti. Pro využívání scén je nutné používat aplikaci iFoxtrot, protože pomocí ní se scény sestavují. Do scény je pak možné použít všechny prvky, které je možné ovládat skrze tuto aplikaci [4].

Pro ovládání scén existují dva bloky – fb_iScene4 a fb_iScene8. Oba bloky jsou totožné, krom jedné vlastnosti, kterou je maximální množství scén. Pro fb_iScene4 je možné nastavit maximálně 4 scény a pro fb_iScene8 jich lze nastavit až 8. V praxi to znamená, že můžeme mít 3 scény nebo 7 scén, je totiž vhodné si nechat poslední volnou scénu na vypnutí všech zařízení [4].

K dispozici jsou vstupní proměnné *scene1On* až *scene4On* (nebo *scene8On* pro fb_iScene8), pomocí kterých se volí konkrétní scéna a jedná se o proměnné typu BOOL, které reagují na náběžnou hranu.

Výstup *file* pak uchovává univerzální název všech scén daného bloku (např. SCN_D8A3.sc?). Konkrétní název souboru scény se pak získá dosazením pořadového čísla scény za „?“.

Poslední proměnnou je *name*, ta uchovává název daného bloku a je možné ji měnit [4].

Tab 1.3 – Sada proměnných z fb_iScene pro PLCComS [4]

	Proměnná	**Typ**	**Význam**
RW	GTSAP1_SCENE_NAME	STRING	Načtení/zápis jména
R	GTSAP1_SCENE_FILE	STRING	Získání umístění souboru
R	GTSAP1_SCENE_NUM	USINT	Získání počtu scén
RW	GTSAP1_SCENE_ENABLE	BOOL	Výpis změn
W	GTSAP1_SCENE_SET1-8	BOOL	Nastavení scény 1-8

R – čtení, W – zápis, RW – čtení i zápis

1.2.2 Bloky světel

Knihovna iControlLib rozlišuje světla podle jejich funkcí a pro každý typ světla má vlastní funkční blok, dohromady jsou 4 a dají se rozdělit do dvou kategorií.

První z nich je pro ovládání obyčejného spínaného světla, pro něj je v knihovně dostupný funkční blok se jménem fb_iLight.

Do druhé kategorie pak patří všechna stmívatelná světla a jsou zastoupeny bloky fb_iDimmer – ovládání obyčejného stmívatelného světla, fb_iDimmerLED – stmívatelné LED světlo a fb_iDimmerRGB – stmívatelné RGB LED světlo.

Každý z těchto bloků má své vlastní proměnné, některé z nich jsou stejné, jiné jsou pak rozdílné, podle funkce daného bloku. PLCComS má však sjednocený název „LIGHT“ pro všechny světelné funkční bloky. Rozlišení je pak uložené pomocí proměnné TYPE a DIMTYPE, viz tab. 1.4 [4].

Tab. 1.4 – Sada proměnných z bloků pro ovládání světel [5]

Proměnná		Typ	Význam
RW	GTSAP1_LIGHT_NAME	STRING	Načtení/zápis jména
RW	GTSAP1_LIGHT_ONOFF	BOOL	Zapnutí/vypnutí světla
R	GTSAP1_LIGHT_TYPE	BOOL	0 = on/off 1 = dimming
RW	GTSAP1_LIGHT_DIMLEVEL	REAL	Stmívač, pouze při TYPE == 1
R	GTSAP1_LIGHT_DIMTYPE	BOOL	0 = jedna barva 1 = rgb
RW	GTSAP_LIGHT_RGB	UDINT	Barva světla, pouze při DIMTYPE == 1

R – čtení, W – zápis, RW – čtení i zápis

1.2.3 Blok žaluzií

Pro ovládání žaluzií je dostupný funkční blok pod jménem fb_iJalousie. Disponuje možnostmi k překlopení lamel o přednastavený krok (*rotUp* a *rotDw)*, popřípadě aktivováním, respektive deaktivováním, pohybu žaluzií nahoru/dolu. Slouží k tomu proměnné *up* a *dw*.

Pozice, ve které se žaluzie nachází není známa a jediná možnost zpětné vazby je pomocí výstupních proměnných *sig*, *sigUP* a *sigDW*, které, při stavu logické „1“, symbolizují pohyb stínidla, kompletní vytažení a kompletní zatažení lamel. Stejně jako ostatní bloky, i tento funkční blok je možné ovládat z webového rozhraní nebo pomocí centrálního ovládání. K těmto účelům jsou vlastní vstupní proměnné pro obě možnosti, které však jen „duplikují“ proměnné již zmíněné [4]. Proměnné pro PLCComS a vysvětlení jejich funkcí se nachází v tab. níže.

Tab. 1.5 – Sada proměnných z fb_iJalousie pro PLCComS [5]

	Proměnná	Typ	Význam
RW	GTSAP1_SHUTTER_NAME	STRING	Načtení/zápis jména
R	GTSAP1_SHUTTER_UP	BOOL	1 = v pohybu nahoru
R	GTSAP1_SHUTTER_DOWN	BOOL	1 = v pohybu dolu
R	GTSAP1_SHUTTER_RUN	BOOL	1 = žaluzie v pohybu
R	GTSAP1_SHUTTER_UPPOS	BOOL	1 = žaluzie vytažena
R	GTSAP1_SHUTTER_DOWNPOS	BOOL	1 = žaluzie zatažena
W	GTSAP1_SHUTTER_UP_CONTROL	BOOL	1 = vytahovat/zastavit žaluzii
W	GTSAP1_SHUTTER_DOWN_CONTROL	BOOL	1 = stahovat/zastavit žaluzii
W	GTSAP1_SHUTTER_ROTDOWN_CONTROL	BOOL	1 = sklopit lamely
W	GTSAP1_SHUTTER_ROTUP_CONTROL	BOOL	1 = otevřít lamely

R – čtení, W – zápis, RW – čtení i zápis

1.2.4 Blok zásuvek

Zásuvky je možné vzdáleně ovládat skrze funkční blok pod jménem fb_iSocket, ten disponuje několika vstupními proměnnými, důležitými z nich jsou *socketToggle,* respektive *webToggle*, kterými se mění stav zásuvky. Doba, po kterou je zásuvka sepnuta, je deklarována v proměnné se jménem *pulseTime*. Pro neomezenou dobu sepnutí je nutné přiřadit jí hodnotu T#0s. Centrální rozepínání všech zásuvek je řešeno pomocí proměnné *socketReset* [4].

Z každého tohoto bloku jsou do PUBFILE generovány dvě proměnné a PLCComS tak může manipulovat pouze s názvem dané zásuvky a jejím stavem. Nastavení maximální doby sepnutí zásuvky je tedy možné pouze v Mosaicu [5]. K využívání centrálního rozpínání pomocí PLCComS je možné použít blok fb_iRelay.

Tab. 1.6 – Proměnné z fb_iSocket pro PLCComS [5]

Proměnná		Typ	Význam
RW	GTSAP1_SOCKET_NAME	STRING	Načtení/zápis jména
RW	GTSAP1_SOCKET_ONOFF	BOOL	Načtení/zápis stavu

1.2.5 Blok relé

Tento blok je pro použití univerzálního spínání, kdy se nemusí jednat pouze o zásuvku. Podle původních záměrů se u tohoto bloku měly rozlišovat tři funkce na základě aktuální hodnoty proměnné TYPE:

- 0: režim on/off
- 1: dimmer/analogová hodnota
- 2: cyklování mezi přednastavenými hodnotami

V případě, že TYPE == 0 nebo 1 je pak v proměnné ONOFF možné vyčíst aktuální stav relé. Proměnná LEVEL, při TYPE == 1, obsahuje analogovou hodnotu v rozmezí 0–100. Dále měl tento blok v PUBFILE generovat proměnné NEXTMODE (pro přepnutí na další režim) a OUTNUMBER (aktuální režim), které se měly zjišťovat pouze při LEVEL == 2. Poslední proměnnou je SYMBOL, ve které je uložena informace o zobrazovaném symbolu konkrétního relé [5].

Bohužel k realizaci výše zmíněného prozatím nedošlo, a tak je blok fb_iRelay funkcí téměř úplně stejný jako výše zmíněný fb_iSocket. Jediným rozdílem je možnost přiřazení konkrétního symbolu [4].

Tab. 1.7 – Současné proměnné z fb_iRelay pro PLCComS [5]

Proměnná		Typ	Význam
RW	GTSAP1_RELAY_NAME	STRING	Načtení/zápis jména
RW	GTSAP1_RELAY_ONOFF	BOOL	Načtení/zápis stavu
R	GTSAP1_RELAY_TYPE	USINT	Aktuálně vždy rovno 1
R	GTSAP1_RELAY_SYMBOL	UINT	

1.2.6 Bloky displejů

Součástí knihovny je i několik funkčních bloků sloužících pro ovládání displejů. Na aplikaci pro hodinky by ale jejich používání bylo dosti komplikované, z tohoto důvodu jsem se rozhodl je využít jen jako informační panely, u kterých nebude možné provádět žádné změny.

K tomu se nejvíce hodí bloky se jménem fb_iDisplay_Val a fb_iSensorTemp. První ze zmíněných je určen k zobrazení obecné hodnoty, ta je typu real a uložena v proměnné *value*. *Precission* udává počet zobrazených desetinných míst, zatímco v *unit* je uloženo nastavení

fyzikální jednotky. Poslední důležitou proměnnou je *symbol*, ve kterém je uložen číselný kód ikony. Je-li *symbol* roven 0, bude zobrazena výchozí ikonka pro displeje [4].

Oproti obecnému displeji blok fb_iSensorTemp může zobrazit pouze teplotu. Hodnota z teplotního čidla se připojuje na vstup proměnné *in* a lze ji následně korigovat o proměnnou *offset*. Současně s tím je také možné hodnoty filtrovat pomocí filtru 1. řádu, jehož časová konstanta se nachází v proměnné *filterTime*. Pro vypnutí filtru se konstanta musí nastavit na #T0s [4].

Pro PLCComS však nedochází k rozlišování mezi žádným z typů funkčních bloků určených pro displeje. Každý blok tak generuje do PUBFILE stejné proměnné, některé z nich se však zjišťují pouze v případě, že je displej možné editovat [5]. V následující tabulce jsou uvedeny pouze proměnné, které jsou důležité pro funkci aplikace.

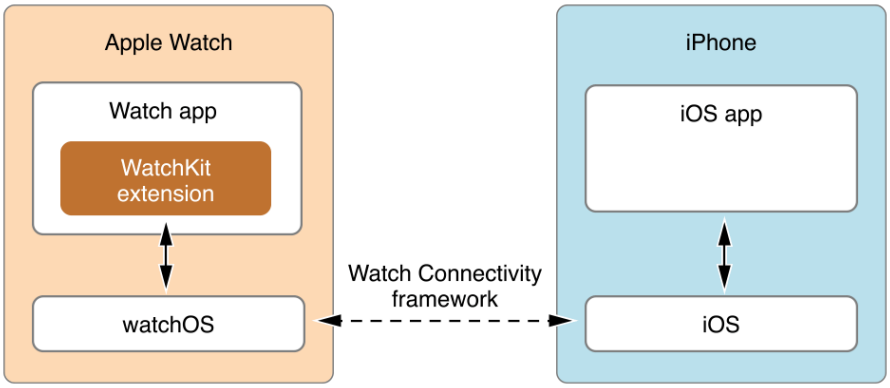
Tab. 1.8 – Vybrané proměnné z bloků displeje pro PLCComS [5]

Proměnná		Typ	Význam
RW	GTSAP1_DISPLAY_NAME	STRING	Načtení/zápis jména
R	GTSAP1_DISPLAY_EDIT	BOOL	Možnost editace displeje
R	GTSAP1_DISPLAY_VALUE	REAL	Aktuální hodnota na displeji
R	GTSAP1_DISPLAY_UNIT	STRING	Nastavená jednotka
R	GTSAP1_DISPLAY_PRECISSION	INT	Počet desetinných míst
R	GTSAP1_DISPLAY_SYMBOL	STRING	Zobrazovaný symbol

2 PRINCIP A FUNKCE APLIKACE

Pro správné rozvržení aplikace je nejprve potřeba se zaobírat principem funkčnosti aplikací pro watchOS, které se nechovají úplně stejně, jako všem známé aplikace z mobilních zařízení. Jsou totiž poměrně hodně omezeny ze strany výrobce, tedy Apple, pravděpodobně především kvůli zajištění delší pohotovostní doby chytrých hodinek na jedno nabití.

Při využívání hodinek je potřeba jejich spárování s mobilním zařízením od stejné firmy, přičemž je nutné mít zařízení iPhone 5s nebo vyšší [6]. Po spárování totiž dochází k veškerým instalacím aplikací skrze mobil, na kterém pak navíc musí být aplikace doinstalována na právě spárované hodinky.



Obr. 2.1 – Relace mezi Watch app, WatchKit extension a iOS app [7]

Na obr. 2.1 je vyobrazeno blokové schéma probíhající komunikace mezi jednotlivými částmi aplikací. Aplikace hodinek tedy obsahuje dva balíčky – Watch app a WatchKit extension. Ve Watch app jsou uloženy všechny storyboardy (obecný název každého UI, které je možné na hodinkách zobrazit), naopak WatchKit extension obsahuje vlastní funkční kód a ostatní zdroje. Současně je možné vyměňovat data mezi aplikací na hodinkách a její mateřskou aplikací na iOS pomocí funkcí nazvaných Watch Connectivity [7].

Přestože WatchKit extension obsahuje programovou funkční část, není schopen navázat TCP spojení. Pro získávání, ale i odesílání, jakýchkoliv informací tedy hodinky potřebují komunikovat s mateřskou aplikací na iPhone, která má možnost používat právě potřebnou komunikaci TCP/IP a obstarávat tak data z PLCComS [8].

2.1 WATCH CONNECTIVITY

V době, kdy hodinky disponovaly systémem watchOS 1 byl WatchKit extension umístěn na straně spárovaného iOS zařízení. Tím bylo umožněno velice snadné sdílení dat mezi rozšířením a samotnou aplikací pro iOS. Po přesunu WatchKit extension do hodinek však tato možnost odpadla, a proto Apple představil nové API, které nese název Watch Connectivity [9].

Pomocí tohoto rozhraní je k dispozici mnohem více informací o současném statusu navázaného spojení mezi hodinkami a spárovaným mobilem. Pro mou práci je nejdůležitější vlastností možnost interaktivního „dopisování“ mezi oběma aplikacemi.

Watch Connectivity nabízí několik možností, jak sdílet data mezi iOS aplikací a WatchKit extension. Každá z nich je vhodná pro úplně jiné účely a je proto velice důležité zvolit správnou možnost [10].

2.1.1 updateApplicationContext

První z těchto možností je aktualizace kontextu aplikace, která umožňuje zaslat menší objem stavových informaci do protější aplikace. Tato metoda by se měla využívat pro synchronizaci anebo aktualizaci obsahu. Odeslaná data jsou považována za volatilní a každé úspěšné zavolání metody přepíše předchozí slovník dat [10].

Data jsou posílána na pozadí ve vhodný okamžik tak, aby se minimalizovalo energetické využití, to znamená, že se přenos může uskutečnit kdykoliv, po zavolání metody a není tak zcela nejvhodnější pro rychlou komunikaci [10].

2.1.2 transferUserInfo

Tato možnost přenosu dat disponuje hned dvěma metodami a nevyžaduje, aby běžel přijímací proces pro přenos dat. K přenášení dat dochází opět na pozadí a datové slovníky jsou řazeny do fronty, na jejímž základě se pak postupně odesílají [10].

Metoda transferCurrentComplicationUserInfo

Při aktuální komplikaci, kdy je třeba odeslat data z iOS aplikace do WatchKit extension se využívá této metody. Po použití je odeslána zpráva s vysokou prioritou, která v případě nutnosti probudí rozšíření a doručí data současně s aktualizací časové osy komplikace.

Bohužel je tato metoda velice náročná na spotřebu energie v hodinkách, a proto je omezeno její denní používání. Před použitím je tedy velmi vhodné zavolat metodu

remainingComplicationUserInfoTransfers, která navrací počet zbývajících volání této funkce po zbytek dne. Pokud je limit již vyčerpán, systém automaticky použije metodu **transferUserInfo** [10].

Metoda transferUserInfo

Vlastně funguje na stejném principu jako výše zmíněná metoda. Pouze s tím rozdílem, že k přenosu dat dojde ve vhodnou dobu, aby opět došlo k minimalizaci spotřeby energie [10].

Touto metodou už tedy není zaručeno okamžité doručení dat, není proto nejvhodnější pro posílání informací určených k okamžitému zpracování.

2.1.3 transferFile

Pro přenos celých souborů lze využívat tuto metodu. Pokud si chce aplikace přijatý soubor ponechat, musí jej přesunout na jiné umístění předtím, než dojde k navrácení metody **session:didReceivedFile**, poté je totiž soubor okamžitě odstraněn. Opět dochází k přenosu v nejvhodnější dobu a k přenosu tak může dojít kdykoliv, třeba i když není ani jedna z aplikací aktivní [10].

2.1.4 sendMessage

Poslední metoda výměny dat mezi iPhonem a Apple Watch. Nabízí okamžitou exekuci přenosu posílaných dat a součástí této metody je i tzv. **replyHandler**, který je odeslán nazpět do zařízení, z kterého byla metoda **sendMessage** použita. Jedná se tak o odpověď na vyslanou zprávu/požadavek a pokud zařízení odpověď nepotřebuje, může tento handler ponechat nulový. Stejně jako v případě **transferUserInfo** jsou zprávy uchovávány ve frontě a doručovány v přesně daném pořadí.

Předtím, než je možné tuto metodu použít, je vždy nutné ověřit dostupnost obou zařízení. iOS aplikace je považována za dostupnou vždy, ovšem watchOS aplikace je dostupná pouze v případě, že je na spárovaných hodinkách nainstalována a současně zrovna v běžícím stavu [10].

2.2 IOS ČÁST APLIKACE

iOS aplikace je nedílnou součástí jakékoliv aplikace pro watchOS, dá se říct, že watchOS aplikace je takovým rozšířením aplikace mobilní. Na její straně se bude muset odehrávat veškerá komunikace pomocí TCP/IP se serverem PLCComS, veškeré zpracování dat a také správa uživatelských účtů. Pro všechny tyto účely by ale v budoucnu měla sloužit již existující aplikace se jménem iFoxtrot, na kterou se pouze přidá rozšíření aplikace pro hodinky.

Z tohoto důvodu je zbytečné se zaobírat grafickou stránkou do hloubky. K výše zmíněním funkcím aplikaci postačí pouze jednoduchý systém správy účtů, u kterého je vhodné, aby se podobal funkčnímu principu v aplikaci iFoxtrot.

Nejdůležitějším obsahem této části tak bude řešení komunikace mezi mobilním zařízením a hodinkami, která se jako jediná bude muset přenášet do aplikace iFoxtrot.

2.2.1 Představa GUI iOS aplikace

Představa o grafické stránce aplikace vychází ze skutečnosti, že s aplikací na iPhone se očekává jen minimální interakce – vytvoření a nastavení účtu. Zbytek je obstaráván na pozadí a okamžitě odesílán do aplikace na hodinkách.

Po spuštění bude na hlavním UI zobrazen výčet všech dosavadních účtů s možností přepínání mezi výchozím účtem. Dále bude vytvořeno rozhraní s možností přidání nového účtu, tak jak je možno vidět na obr. 2.2 vpravo.

Obr. 2.2 – Představa GUI na iPhone

Aplikace iFoxtrot nabízí mnohem více parametrů k nastavení, mimo jiné včetně názvu PUBFILE umístěného v PLC, pro testovací účely komunikace s PLCComS však tyto parametry nejsou potřebné.

Seznam účtů by měl být uložen v zařízení a při zjištění jeho nepřítomnosti by se mělo uživateli zobrazit rovnou UI pro přidání nového účtu. Po přidání prvního účtu by také mělo dojít k jeho automatickému nastavení jako výchozího.

2.2.2 Komunikace s PLCComS

Od spuštění aplikace na hodinkách je nutné, aby mobilní část navázala komunikaci s PLCComS, která bude aktivní až do samotného vypnutí aplikace na hodinkách. TCP/IP připojení by tedy mělo být spuštěno na pozadí tak, že jej bude možné využít v libovolný moment.

Samotnou komunikaci je možné řešit dvěma způsoby, synchronním nebo asynchronním. **Synchronní** způsob umožňuje snadnější zpracování dat, na které se v předchozí chvíli uživatel ptal – po odeslání příkazu dochází k zaktivování čtení dat příchozích. Nevýhodou však je nutnost čekání aplikace v době provádění tohoto čtení. Za největší problém u synchronní komunikace lze považovat komplikované získávání změny stavů proměnných (DIF:). O jejich příchodu totiž aplikace při použití právě této komunikace neví, a tak by se jako jediná možnost naskytovalo použití časovače, který by ve zvolené periodě kontroloval data na vstupu TCP/IP. Právě při použití časovače pak může dojít k prodlevě mezi vysláním příkazu a odezvou, kterou uživatel uvidí.

Asynchronní komunikace by umožnila okamžité zpracování všech příchozích dat, aniž by musela aplikace při jejich čtení čekat. Odezva pro uživatele by tak mohla být minimalizována na jejich vyhodnocení a přenos do hodinek. Na rozdíl od synchronní komunikace, kdy aplikace přesně ví, na jaké data čeká, je v tomto případě nutné nějakým způsobem tuto informaci pro zpracování udržovat.

2.3 WATCHOS ČÁST APLIKACE

Hlavní částí celé této práce je aplikace pro hodinky, očekává se od ní co nejsnadnější a uživatelsky nejpříjemnější ovládání, současně s využitím co nejvíce možností k ovládání, které server PLCComS nabízí. Velkou inspirací je aplikace iFoxtrot, která k ovládání nabízí všechny dostupné objekty, některé, jako např. kamera, nebo nastavování týdenního diagramu vytápění, jsou však pro využití na hodinkách nevhodné. Výsledná aplikace pro hodinky má totiž být

součástí současné aplikace iFoxtrot a umožňovat tak jen rychlejší možnost nastavení důležitých objektů, aplikace se bude zabývat jen funkčními bloky Mosaicu vypsanými v kap. 1.2.

Vzhled aplikace by měl být intuitivní, minimalistický a příjemný pro používání ve dne, ale i v noci. GUI aplikace bude vhodné rozdělit na 3 základní typy, podle následného fungování aplikace.

2.3.1 Výchozí rozhraní

Jak už bylo zmíněno výše, aplikace pro svou správnou funkci potřebuje získávat data prostřednictvím mobilního telefonu, který je přijímá od serveru PLCComS. Získávání a následné zpracování dat však může nějakou dobu trvat, především jedná-li se o obsáhlejší projekt k ovládání. Pro informování uživatele tak bude sloužit právě toto rozhraní, které bude zobrazeno ihned po spuštění aplikace jako „výchozí“ rozhraní, tedy uživatel z něj nebude moci samovolně odejít, ale ani se do něj samovolně navrátit.

Při používání aplikace můžu dojít k několika chybám – výpadku komunikace mezi PLCComS a mobilem, vypnutím PLC nebo chybně načteným souborem. Ve všech těchto případech bude zobrazeno právě toto UI, na kterém bude k dispozici řešení nastalé situace za předpokladu, že se situace řešit dá. Posledním případem zobrazení tohoto GUI je při varování ze strany PLCComS o změně PUBFILE.

Předpokládaný vzhled je na obr. 2.3. a bude se drobně lišit v závislosti na právě probíhající akci.

Obr. 2.5 – Stavy předpokládaného vzhledu výchozího GUI aplikace hodinek

2.3.2 Rozhraní místností

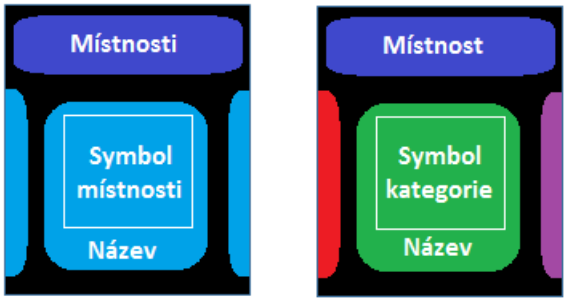
Ve chvíli, kdy dojde k dokončení úvodní inicializace dostupných objektů a místností by mělo dojít k přesměrování do GUI s výpisem všech dostupných místností pro současný projekt v PLC. Toto rozhraní bude načteno jako „výchozí“, aby se uživatel mohl vrátit vždy maximálně na toto rozhraní.

Na konec celého seznamu s místnostmi bude navíc umístěna místnost s názvem „VŠE“, která musí vždy obsahovat všechny hodinkami používané objekty. Uživatel totiž nemusí mít všechny objekty do místností rozřazeny.

Soubor s celým nastavením místností si musí sám uživatel vygenerovat prostřednictvím mobilní aplikace iFoxtrot. Pro každou místnost je v souboru uložena informace o obsažených objektech, názvu místnosti a zvolenému symbolu pro místnost [11]. Případně si uživatel může soubor sám vytvořit ve správném formátu a uložit ho do PLC pod jménem „iFOX/ifoxtopo.jsn“.

Po výběru konkrétní místnosti se pak uživateli zobrazí daná místnost s výpisem v ní umístěných kategorií objektů. Pořadí zobrazených kategorií bude pevně dáno, ale zobrazovat by se měli pouze kategorie, které obsahují alespoň jeden objekt.

Při zvolení kategorie se pak uživateli načte příslušné GUI kategorie a bude zobrazen seznam objektů.



Obr. 2.6 – Předpokládaný vzhled GUI místností a jedné místnosti

2.3.3 Rozhraní kategorií a samotných objektů

Každá kategorie objektů bude mít své vlastní GUI, a to i přestože by se některé mohli výrazně podobat, nebo být přímo totožné (rozhraní pro zásuvky a relé). Je totiž lepší počítat s možností snadné úpravy a případného rozšíření možností každé z kategorií.

Pouze jediná kategorie – displeje – bude výrazně jiná než všechny zbylé. Její vzhled by měl vypadat podobně jako GUI s přehledem místností. Tím by měl mít uživatel možnost pohodlného prohlížení všech dostupných objektů bez nutnosti dalšího výběru.

Na obr. 2.5 je vyobrazena obecná představa přehledu všech ostatních kategorií. Mělo by se jednat o jednoduchou tabulku, která bude zobrazovat všechny objekty dané kategorie. Na základě kategorie by pak mohl objekt obsahovat informaci o jeho aktuálním stavu, která by v řádku byla zarovnána napravo.

Obr. 2.7 – Představa GUI kategorie

Kategorii zásuvek, relé a částečně i světel by mělo být možné ovládat rovnou z vyobrazené tabulky objektů. Při stisknutí objektu by tak došlo k jeho zapnutí/vypnutí.

Pro žaluzie, scény a světla bude navíc potřeba zajistit přechod do GUI samotného objektu.

Objekt scény

Každá ze scén může obsahovat až 8 volitelných možností. Při výběru objektu scény by tedy mělo dojít k zobrazení podobné tabulky, která bude zobrazovat aktuálně dostupné možnosti nastavení.

Objekt světel

Světlo může mít i další parametry, nežli jen zapnuto/vypnuto. V zobrazeném GUI pro světlo tak bude možnost nastavení stmívače a výběru barvy světla z předpřipravené nabídky barev. Zobrazení těchto prvků bude podmíněné. V případě, že světlo žádnou z těchto nastavitelných možností disponovat nebude, mělo by být zakázáno toto rozhraní otevřít.

Obr. 2.8 – Představa GUI světla

Objekt žaluzií

Pro ovládání žaluzií jsou dostupné 4 volby, které musí být uživateli k dispozici najednou. Na obr. 2.7 je možné vidět představu tohoto GUI, horní dvě tlačítka by měla sloužit k pohybu žaluzií nahoru a dolů, spodní tlačítka pak pro otáčení lamel. Aktuální stav pohybu žaluzií by měl být uživateli zobrazen uprostřed, přičemž bych rád využil stejných symbolů, jaké budou na tlačítkách.

Obr. 2.9 – Představa GUI žaluzie

3 KOMUNIKACE MEZI MOBILEM A HODINKAMI

Ze všech metod, které Watch Connectivity nabízí, jsem pro tuto komunikaci zvolil metodu **sendMessage**, která zajišťuje odeslání zprávy do protějšího zařízení v nejbližší možný moment. Tato metoda je jednou z nejpoužívanějších, a proto existuje na internetu již mnoho vytvořených tříd, které se komunikací zabývají.

Rozhodl jsem se použít třídu od iOS vývojáře, který na internetu vystupuje pod pseudonymem Lito. Jeho třída se jménem *WatchSessionManager* umožňuje správu WCSession, která inicializuje komunikaci mezi WatchKit Extension a iOS aplikací, z jednoho místa – tedy dané třídy [12].

3.1 AKTIVACE KOMUNIKACE

Před samotným používáním komunikace mezi zařízeními je nutná její aktivace. Tu je vhodné volat ihned po spuštění obou aplikací. K deaktivaci pak může dojít pouze v případě, že uživatel iOS zařízení změní hodinky. V takovém případě se volá funkce *sessionDidBecomeInactive()* a dochází k zablokování možnosti nového datového přenosu. Data, která již čekají na odeslání, jsou v co nejbližší době přenesena a dochází k volání funkce *sessionDidDeactivate()*, která je určena k opětovnému aktivování spojení, tentokrát s jinými hodinkami. Vytvoření relace je podporováno pouze pro hodinky, nebo vybrané druhy iPhone, před získáním singletonu je tak potřeba vždy podporu ověřit pomocí funkce *isSupported()* [13].

Lito má ve své třídě pro aktivaci komunikace funkci jménem *startSession()*, po jejím zavolání dochází k aktivaci delegování a následně i celé komunikace.

```
fileprivate let session: WCSession? = WCSession.isSupported() ? WCSession.default : nil

func startSession() {
        session?.delegate = self
        session?.activate()
}
```

Obr. 3.1 – Ověření podpory relace a její aktivace [12]

Informace o stavu relace je následně zpracována pomoci rozšíření WCSessionDelegate, u kterého jsem navíc přidal delegování proměnné *ActivationState* pro hodinky.

3.2 ROZŠÍŘENÍ WATCHSESSIONMANAGER

Třída WatchSessionManager dále obsahuje rozšíření, které se stará o balení funkcí **sendMessage** pro odesílatele i příjemce. Při posílání zpráv je nutné vždy ověřit, že se jedná o platnou *Session* – na straně iOS se kontroluje, zda jsou hodinky spárované (*isPaired*) a současně se zjišťuje, jestli je na nich i nainstalovaná aplikace (*isWatchAppInstalled*).

U hodinek se tato kontrola provádět nemusí, protože musí být vždy spárovány s mobilním zařízením a instalace aplikace pak probíhá vždy z tohoto zařízení. Jsou-li tyto podmínky splněny, je navíc potřeba ověřit dostupnost obou zařízení pro použití živého zasálání zpráv. K tomuto účelu je připravena funkce *isReachable* [12].

```
var validSession: WCSession? {

    #if os(iOS)
    if let session = session, session.isPaired && session.isWatchAppInstalled {
        return session
    }
        return nil
    #elseif os(watchOS)
        return session
    #endif
    }
```

Obr. 3.2 – Ověření možnosti použití relace [12]

```
  private var validReachableSession: WCSession? {
      if let session = validSession, session.isReachable {
          return session
      }
      return nil
  }
```

Obr. 3.5 – Ověření dostupnosti zařízení [12]

3.2.1 Posílání zpráv

K odesílání zpráv jsou připraveny dvě metody. První odesílá slovník typu [String:AnyObject] a jmenuje se **sendMessage**, druhá pak, pod jménem **sendMessageData**, slouží k odesílání objektů typu Data. Pro obě metody je volitelně možné využít typ s očekávanou odpovědí [12]. Ke zpracování odpovědi pak dochází asynchronně uvnitř této metody a lze z ní opět volat poslání další zprávy.

```
connectivityHandler.sendMessage(message: ["zprava" : "data" as AnyObject] , replyHandler:{(response) in
    //odpoved response
    //prostor pro zpracovani odpovedi
}) {(error) in
    print(error)
    //prostor pro zpracování erroru
}
```

Obr. 3.6 – Použití metody sendMessage

Pro každé volání metody **sendMessage** s očekáváním odpovědi je vždy nutné zařídit odeslání odpovědi, tedy *replyHandler* podle obr. 3.4. V opačném případě totiž po nějaké době komunikace vypíše *error*, protože uplyne čas, do kterého musela být odpověď navrácena a nedojde k následnému spuštění kódu pro odpověď.

V případě použití metody bez odpovědi nedochází k žádné zpětné reakci a její využívání je tak vhodné v případech, kdy dochází pouze k informování protistrany o nastalé události.

3.2.2 Příjem zpráv

Z výše zmíněného vyplývá, že příjemce může očekávat dva typy přijaté zprávy – zprávu s požadavkem na odpověď nebo bez. Z tohoto důvodu je nutné zpracovávat oba nastalé eventy zvlášť, což by ale znamenalo komplikaci při delegování.

Lito proto do své třídy připravil zabalení obou těchto možností do jedné protokolové funkce, která je následně delegována podle typu OS, na který zpráva přišla [12].

```
func session(_ session: WCSession, didReceiveMessage message: [String : Any]) {
    handleSession(session, didReceiveMessage: message)
}

func session(_ session: WCSession, didReceiveMessage message: [String : Any], replyHandler: @escaping ([String : Any]) -> Void) {
    handleSession(session, didReceiveMessage: message, replyHandler: replyHandler)
}

func handleSession(_ session: WCSession, didReceiveMessage message: [String : Any], replyHandler: (([String : Any]) -> Void)? = nil) {

    #if os(iOS)
        iOSDelegate?.messageReceived(tuple: (session, message, replyHandler))
    #elseif os(watchOS)
        watchOSDelegate?.messageReceived(tuple: (session, message, replyHandler))
    #endif
}
```

Obr. 3.7 – Metody pro příjem zpráv a jejich zabalení s následnou delegací [12]

3.3 PROTOKOLY PRO DELEGACE

Třída obsahuje protokoly, pomocí kterých je vždy každá přijatá zpráva delegována. Z tohoto důvodu pak musí každý kontrolér, který tuto třídu využívá, obsahovat příslušné rozšíření se všemi metodami, které jsou pro daný OS deklarovány, nezávisle na tom, jestli jich následně využije [12].

Protokol pro watchOS jsem navíc rozšířil o mnou přidanou funkci, která informuje o úspěšném aktivování komunikace. Její využívání bude důležité především pro „výchozí“ rozhraní aplikace. Hodinky jsou totiž první, kdo se snaží aktivně zaslat zprávu. Jak už ale víme, před odesláním musí dojít k aktivaci komunikace. Delegace stavu aktivace tak umožní reakci a následné odeslání první zprávy.

```
typealias MessageReceived = (session: WCSession, message: [String : Any], replyHandler: (([String : Any]) -> Void)?)

protocol WatchOSDelegate: AnyObject {
    func messageReceived(tuple: MessageReceived)
    func connectionStatus(status: WCSessionActivationState)
}

protocol iOSDelegate: AnyObject {
    func messageReceived(tuple: MessageReceived)
}
```

Obr. 3.8 – Delegační protokoly pro iOS a watchOS

4 TVORBA APLIKACE PRO WATCHOS

Vývoj aplikace na hodinky jsem začal tvorbou tříd pro každou kategorii objektů k ovládání. Třídy obsahují proměnné podle jednotlivých podkapitol 1.2. Přičemž jsou zaznamenávány pouze proměnné, které jsou určeny ke čtení skrze PLCComS a navíc je vždy přidána unikátní proměnná **id** daného objektu.

Při inicializování se musí povinně zadat **id** a **jméno** objektu, ostatní proměnné mohou zůstat jako **nil** a k jejich naplnění dochází při přechodu do kategorie. Pro třídu *Scene* je pak navíc, místo proměnné **num**, přidán slovník typu [String:String], který je připraven pro uložení čísla a jména scény. K tomuto řešení jsem přistoupil z důvodu, že **num** udává pouze maximální možný počet scén, které lze pro blok nastavit. Neznamená to tedy, že je uživatel má skutečně všechny nastavené.

V současné době má aplikace k dispozici tyto třídy objektů: *Scene*, *Light*, *Shutter*, *Socket*, *Relay* a *Display*. Každá třída obsahuje statickou proměnnou pojmenovanou množným číslem od jména třídy, tedy např. **Scenes**, která je deklarovaná jako pole proměnných dané třídy a jako jediná obsahuje všechny inicializované objekty kategorie. Součástí tříd jsou také dvě funkce, první slouží k inicializaci objektu a současnému přidání do výše zmíněné statické proměnné. Jméno funkce vychází vždy z názvu třídy, např. func **addScene**, kterou je možné vidět na obr. níže.

```
class func addScene(id: String, name: String, file: String? = nil, data: [String:String]? = nil){
    Scenes.append(Scene.init(id: id, name: name, file: file, data: data))
}
```

Obr. 4.1 – Funkce třídy Scene pro přidání nové scény do statické proměnné

Druhá funkce slouží k vyhledávání objektu ve statické proměnné dle **id** a navrací **index** typu *Int* odkazující na daný objekt, respektive **nil** v případě, že neexistuje objekt této třídy s vyhledávaným **id**. Funkce je pojmenována opět na základě příslušné třídy, např. **getSceneIndex**.

Využití této funkce je důležité pro správné zpracování souboru ifoxtopo.jsn, ve kterém jsou uloženy pouze **id** proměnných, bez informace o kategorii, ale i pro zpracování příchozích změn DIF: z PCComS.

```
class func getSceneIndex(id: String) -> Int? {
    for i in 0...(Scenes.count-1){
        if Scenes[i].id == id{
            return i
        }
    }
    return nil
}
```

Obr. 4.2 – Funkce třídy Scene pro získání indexu objektu dle jeho id

Protože projekt může obsahovat i uživatelem nastavené místnosti s rozmístěnými objekty libovolných kategorií, rozhodl jsem se vytvořit i samostatnou třídu jménem *Room*, která slouží pro ukládání informací každé místnosti. Obsahuje jméno místnosti, uložené v proměnné **name**, obrázek vybraný uživatelem v proměnné **symbol** a následně pro každou z kategorií objektů jedno pole typu Integer pro odkazování na přiřazené objekty v této místnosti.

Stejně jako ostatní třídy má i tato svou statickou proměnnou **Rooms** ve které jsou všechny dostupné místnosti uloženy.

4.1 VÝCHOZÍ ROZHRANÍ

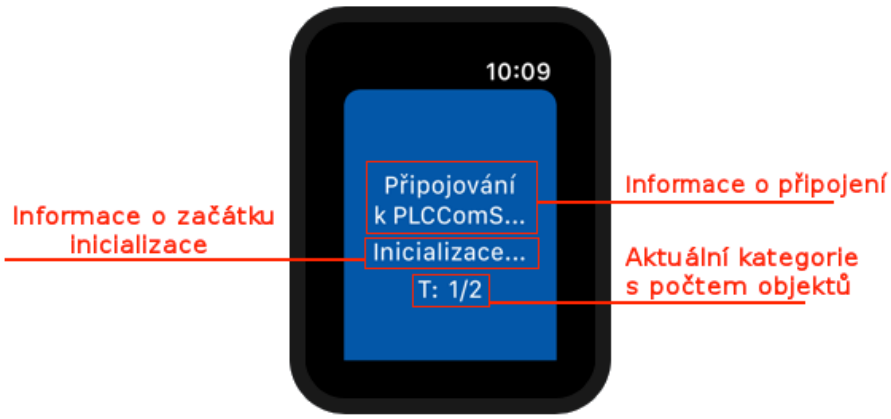
Tento kontrolér s názvem *MainMenuInterfaceController* funguje jako informační obrazovka pro uživatele a díky funkci překrývání disponuje několika možnými vzhledy. K jeho zobrazení dochází okamžitě po spuštění aplikace. Načtený vzhled závisí na proměnné **opt**, která je typu String a nastavuje se z přijatého kontextu při zobrazení. V případě prvotního startu aplikace, a pouze v tento moment, je kontext **nil**. V aplikace při této možnosti dochází pouze k aktivování komunikace s mobilním zařízením. K ukončení tohoto stavu dochází po úspěšném navázání komunikace, na což je aplikace informována prostřednictvím protokolu zmíněném v kap. 3.3.

Následně dochází k opětovnému načtení tohoto kontroléru, tentokrát s kontextem „inicialization“.

Obr. 4.3 – Obrazovka při navazování komunikace s mobilem

4.1.1 Možnost inicialization

Při načtení kontroléru s touto možností dochází nejprve k zaslání požadavku o navázání spojení mezi mobilem a PLCComS a následně ke kompletní inicializaci všech objektů sledovaných kategorií. Poté dojde i k inicializaci uživatelem navolených místností pro daný projekt v PLC. V průběhu celého procesu nemá uživatel žádnou možnost interakce, je ale informován o aktuálním stavu děje.



Obr. 4.4 – Informace pro uživatele po čas úvodní inicializace

Požadavek o připojení k PLCComS

Aplikace vyšle zprávu ve tvaru *[„connection“ :„connect“]* do mobilního zařízení, kterou se vyvolá pokus o navázání TCP/IP spojení mobilu s PLCComS. Následně se čeká na odpoveď, která musí být ve tvaru *[„connection“ : stav]*.

Podle přijatého stavu se pak vyhodnocují následující situace:

- „1“ – připojení proběhlo úspěšně, pokračuje se v inicializaci zasláním požadavku o načtení objektů
- „0“ – připojení se nezdařilo a není znám důvod selhání. Dochází k opětovnému načtení tohoto kontroléru, jehož kontext je nastaven na **connection.**
- „-1“ – připojení se nezdařilo, na mobilním zařízení není nastaven výchozí účet pro navázání spojení. V takovém případě je nutné otevřít mobilní aplikaci a zkontrolovat správnost nastavení. Kontrolér je načten znovu s kontextem **nodefa**.

Požadavek na inicializaci všech objektů v PUBFILE

Po úspěšném připojení, tedy při obdržení odpovědi *connection* == „1“ je vyslána zpráva ve tvaru *[„connection“ : „initialization“]* a jako odpoveď se tentokrát ve slovníku očekávají dva klíče.

První klíč se jménem *inicialization* musí obsahovat string, na základě kterého se určuje, jestli byla inicializace na straně mobilu úspěšně provedena a zpracována. Stav „0“ znamená selhání této akce, aplikace pak opět načítá stejný kontrolér a jako kontext nastaví **objects.**

Pokud je stav roven „1“, načtou se data pro klíč *connection*, které musí odpovídat typu [String:[String:String]], tedy slovníku slovníků. Klíči prvního slovníku jsou názvy kategorií objektů a každý z nich pak obsahuje další slovník. Tento slovník už obsahuje informace potřebné pro prvotní inicializaci objektů ve tvaru [**id**:**name**].

Tento obsáhlý slovník slovníků je předán funkci se jménem **inicialization**, která se postará o celkovou inicializaci. Funkci je možné najít v příloze B na straně 3. Následně dochází k inicializaci místností.

Požadavek na načtení místností projektu

Dojde k odeslání zprávy *[„connection“ : „rooms“]* s očekávanou odpovědí tvaru [Integer:[String:Anyobject]], která je předána místní funkci jménem **rooms**, její kód s komentáři je možné najít v příloze B na straně 4.

V ní dochází nejprve ke kontrole klíče **0**, na základě kterého se vyhodnocuje, zdali nastala chyba při čtení souboru. Chyby jsou rozděleny na dva typy – ERROR 1 a ERROR 2.

ERROR 1 znamená, že se na PLC nenachází soubor určený pro uložení místností. V takovém případě se inicializuje pouze univerzální místnost se jménem „VŠE“, která obsahuje výčet úplně všech dostupných zařízení.

ERROR 2 naopak značí, že soubor přítomný je, ale nepovedlo se jeho správné čtení. Nastane-li tento případ, dochází k opětovnému načtení kontroléru s možností **Rooms.**

Naopak pokud ani jedna z výše zmíněných chyb nenastane, spustí se plnění místností. Pro každou místnost byly načteny **id** objektů, bohužel ale bez informace o jejich kategorii. Proto se každé **id** postupně kontroluje pro každou z kategorií až do doby, než je úspěšně navrácen **index**, čímž dochází k roztřídění.

Při inicializaci místností jsou pak na základě **id** roztříděny objekty do kategorií a dochází k uložení jejich **indexu**. Místnost tak disponuje pouze odkazem na pozici daného objektu a umožňuje snazší manipulaci s daty (nedochází k duplikaci a stačí tak informace změnit na jednom místě – ve statické proměnné třídy). Výhodou tohoto stylu odkazování se je možnost přidělení záporného čísla, které nemůže být nikdy indexem, jako symbolizace obsažení všech objektů dané třídy pro místnost. Pro zapsání všech objektů třídy využívám hodnoty [-1].

Na konci rozřazování každé místnosti dochází k její inicializaci, tedy použití funkce **Room.addRoom**, s jejím názvem, symbolem a roztříděnými číselnými odkazy na objekty. Poté co jsou všechny místnosti inicializovány je navíc přidána místnost „VŠE“ se všemi objekty.

Jestliže se celá inicializace dostane až do tohoto bodu, následně dochází k načtení kontroléru obstarávajícím zobrazení výčtu místností.

4.1.2 Možnost Disconnected

Tato možnost je připravena pro stav kdy se přeruší TCP/IP spojení mobilního zařízení s PLCComS serverem. Uživateli je zobrazena krátká zpráva, kterou je možno vidět na obr. níže. Uživatel se pak pomocí tlačítka může pokusit o obnovení spojení. Jestliže dojde k jeho obnovení, načte se tento kontrolér s kontextem **inicialization.** V opačném případě se pouze opět načte tento stejný stav.

Obr. 4.5 – Obrazovka při přerušení TCP/IP

4.1.3 Možnost Nodefa

K zobrazení této části dochází, pokud aplikace pošle požadavek o navázání TCP/IP spojení, ale mobilní strana neobsahuje výchozí účet. V takovém případě se na hodinkách nedá nic dělat a jediným řešením je přidat výchozí účet do mobilní aplikace. Následně může uživatel opakovaně požádat o pokus navázání spojení.

Obr. 4.6 – Obrazovka při nevyplněném účtu

4.1.4 Možnost WatchConnectivity

Při výpadku komunikace mobilu a hodinek se zobrazí informační hláška, kterou lze vidět na obr. níže. Tento stav může nastat při změně hodinek. Aplikace se automaticky pokusí znovu navázat spojení

4.1.5 Možnost Plcoff

Ve chvíli, kdy je PLC uvedeno do režimu halt, není možné provádět změny v jeho programu. Z tohoto důvodu je potřeba, aby byl uživatel informován. K tomu slouží právě tento kontext. Po jeho obdržení musí uživatel vyčkat, dokud se PLC znovu nespustí do režimu run. Následně je provedena kompletní inicializace. Stejně jako při kontextu **WatchConnectivity** je nutné zajistit přesměrování kdykoliv dojde k vypnutí PLC. Ošetření tohoto stavu se tak nachází v úplně každém kontroléru.

Obr. 4.7 – Obrazovka při vypnutém PLC

4.1.6 Možnost Rooms

Jak už bylo zmíněno v kapitole 4.1.1, při selhání načtení souboru místností je uživatel přesměrován na tuto část. Je zde informován o nastalém problému a současně s tím dostává na výběr ze dvou možností.

První je pokus o opakované načtení souboru s místnostmi a jejich následná inicializace. V případě, že tato možnost selže, uživatel se dostane opět na stejné místo. Naopak druhá možnost umožňuje, v případě nastalé chyby, celý soubor s místnostmi ignorovat a pouze tak dojde k deklaraci místnosti „VŠE“.

Obr. 4.8 – Obrazovka při chybném načtení místností

4.2 ROZHRANÍ MÍSTNOSTÍ

GUI s výčtem místností je pro uživatele prvním rozcestníkem. Aplikací je načten jako rootInterfaceController, uživatel tak následně nemá možnost se v aplikaci dostat níže než na tento kontrolér. Před jeho načtením musí vždy dojít k naplnění třídy *Room*. Kontrolér pro toto rozhraní je pojmenován *RoomsInterfaceController*.

Pro uživatele je k dispozici pouze jedno tlačítko, ve kterém je umístěn obrázek symbolu místnosti a pod ním její jméno. Pokud chce uživatel vybrat jinou místnost, musí využít gesta potáhnutím vpravo, nebo vlevo. To funguje pouze za předpokladu, že se před, respektive za, právě zvolenou místností nachází alespoň jedna další. Uživatel je o této možnosti vizuálně informován v podobě malé části „tlačítka“ po bocích.

Obr. 4.9 – Procházení seznamu místností

Místnosti jsou do tohoto kontroléru předány v podobě pole číselných indexů, jedná se tak opět o odkaz na proměnnou **Room.Rooms**. Kontrolér si pak uchovává aktuální pozici zobrazené místnosti, které se při použití gesta jednoduše inkrementuje nebo dekrementuje. Současně s tím dochází ke změně zobrazeného symbolu, jména a také pořadového čísla zobrazené místnosti.

```
@IBAction func swipeR(_ sender: AnyObject){
    if R>0{
        R -= 1 //R obsahuje poradove cislo mistnosti
        //skryti leve casti pri zobrazeni prvni mistnosti
        if R == 0{
            leftG.setHidden(true)
        }
        rightG.setHidden(false)
        //nastaveni poradoveho cisla mistnosti
        cM.setText(String(R+1))
        //nastaveni jmena mistnosti
        btLabel.setText(Room.Rooms[R].name)
        //nastaveni obrazku mistnosti
        btImage.setImageNamed("R" + String(Room.Rooms[R].symbol))
    }
}
```

Obr. 4.10 – Funkce místností při swipe vpravo

V případě stisknutí tlačítka s místností dojde k načtení kontroléru se jménem *RoomInterfaceController* a jako kontext je předán objekt celé zvolené místnosti.

4.3 ROZHRANÍ MÍSTNOSTI

Kontrolér se jménem *RoomInterfaceController* slouží ke zobrazení všech dostupných kategorií, které obsahují objekty. Při jeho volání se jako kontext očekává proměnná třídy *Room*.

Kontrolér obsahuje 3 předem připravené pole dat, které obsahují jména kategorií, jména obrázků a přiřazené barvy. Při zobrazení pak dochází k roztřídění dat z kontextu tak, že se naplní předtím prázdné číselné pole nenulovými kategoriemi z předané místnosti. Uživatel má k dispozici stejné ovládání jako v případě rozhraní s místnostmi.

Po zvolení konkrétní kategorie dochází k blokaci gest a zobrazení animace v pravém horním rohu. V tuto chvíli se do mobilního zařízení odešlou všechny dostupné **id** objektů patřících do dané kategorie s následným čekáním na odpověď. Aplikace totiž nejprve potřebuje aktivovat příjem změn proměnný pro tyto objekty a současně s tím načíst všechny dostupné informace, při inicializaci bylo zjištěno pouze **id** a **name**.

Po přijetí odpovědi tak následuje roztřídění dat, které je řešené pro každou z kategorií zvlášť, nakonec dochází k načtení kontroléru, který se stará o vykreslování vybrané kategorie.

Celý nastalý děj po stisknutí tlačítka je možné vidět v okomentované části programu, která je součástí přílohy B na straně 6.

Obr. 4.11 – Procházení kategorií v místnosti

4.4 KATEGORIE SCÉN

První zobrazovanou kategorií jsou scény. Jak už bylo zmíněno dříve, výpis všech objektů je prováděn pomocí tabulky. Kontrolér se jmenuje *ScenyInterfaceController*, data mu jsou předána v podobě číselného pole, a každé číslo je **indexem** odkazujícím na objekt třídy *Scene*, uloženým v **Scene.Scenes**.

V případě, že se přijatý kontext rovná [-1] je potřeba načíst úplně všechny scény. K tomu stačí pouze spočítat velikost pole proměnné **Scene.Scenes** a odečíst 1, protože index začíná 0.

```
if Scene.Scenes.count>0 {
    if data == [-1] {
        data = []
        for a in 0...(Scene.Scenes.count-1) {
            data.append(a)
        }
    }
}
```

Obr. 4.12 – Kontrola dat kontroléru u scén

Následně je potřeba naplnit tabulku daty. Práci s tabulkami se věnuje Audrey Tam ve svých tutoriálech, kterými jsem se inspiroval [14].

Plnění tabulky se odehrává pomocí kódu na obr. 4.12. Tabulka má ve storyboardu nastaven jeden řádek, který má identifikátor „ScenesRow“ a je použit pro každý řádek v tabulce. Také je vytvořen kontrolér pro tyto řádky, který se stará o jejich plnění obsahem. V tomto případě pouze jménem, u kterého se hlídá počet písmen. Pokud je text delší než 11 znaků, dochází ke zmenšení fontu na velikost 13, jinak je zachována standardní velikost – 16.

```
ScenesTable.setNumberOfRows(data.count, withRowType: "ScenesRow")
for index in 0..<ScenesTable.numberOfRows {
    guard let controller = ScenesTable.rowController(at: index) as? SceneRowController else {continue}
    controller.scene = Scene.Scenes[data[index]]
}
```

Obr. 4.13 – Plnění tabulky se scénami

4.4.1 Výběr scény

Pro výběr jedné ze scén je možné využít jednoduchého kliknutí. Reaguje na to funkce tabulky, která dostane informaci o indexu řádku, díky čemuž je následně možné dohledat konkrétní scénu.

Poté dochází k odeslání zprávy ve tvaru [„getF“:scene.file], přičemž scene.file obsahuje přístupovou cestu, pod kterou mají být uloženy soubory s nastavením pro daný funkční blok. Uživatel musí čekat na odpověď ze strany mobilu. Ta kontroluje dostupnost všech možností pro soubory. Jako odpověď jsou očekávány data ve tvaru [číslo scény: jméno scény]. Celý tento slovník je pak následně uložen do daného objektu.

Následuje načtení kontroléru se jménem *ScenaInterfaceController* s předaným kontextem objektu scény.

```
override func table(_ table: WKInterfaceTable, didSelectRowAt rowIndex: Int) {
    let scene = Scene.Scenes[data[rowIndex]] //vyber sceny z pole
            //odeslani pozadavku pro mobilni stranu
            self.connectivityHandler.sendMessage(message: ["getF" : scene.file as AnyObject ], replyHandler: {(response) in
                print(response)
                //pri uspesne odpovedi dochazi ke zpracovani dat a nacteni GUI sceny
                if let d = response["getF"] as? [String:String]{
                    Scene.Scenes[self.data[rowIndex]].data = d
                    self.presentController(withName: "Scene", context: self.data[rowIndex])
                }
            }) {(error) in
                print("error")
            }
}
```

Obr. 4.14 – Program při výběru scény z tabulky

4.4.2 Rozhraní scény

Jedná se o stejné GUI, které obsahuje také tabulku. Ta je plněna stejným způsobem jako bylo již zmíněno výše. Vizuální rozdíl je pouze v barvách, které jsou prohozené. Při jednoduchém výběru scény dochází k odeslání požadavku na její zapnutí, který je tvaru [„update“:[„SCENE“, scene.id, „1“]] a po úspěšném obdržení mobilním zařízením je zobrazena krátká animace jako zpětná vazba uživateli.

Obr. 4.15 – Scéna s možnostmi jejího nastavení

4.5 KATEGORIE SVĚTEL

Stejně jako kategorie scén, je i kategorie světel zpracována stejným systémem. Rozdílem je, že každý řádek obsahuje několik informací navíc – zabarvení, stmívač světla a jeho stav.

Stav, který je možno vidět na řádku vpravo určuje, jestli je světlo v zapnutém, nebo vypnutém stavu. Stmívač světla se nachází uvnitř řádku a jeho stav je vidět pouze za předpokladu, že je dané světlo rozsvícené. Původním záměrem mělo být, aby bylo toto nastavení možno vidět i při vypnutém světle, bohužel to však nepodporuje daný funkční blok pro ovládání z PLCComS.

Pokud světlo podporuje režim RGB, dochází ke zabarvení vnitřní části řádku na právě nastavenou barvu světla.

Obr. 4.16 – Kategorie světel

4.5.1 Rozhraní světla

Do rozhraní světla je možné se dostat dlouhým stiskem daného řádku. Bohužel jsem však nepřišel na způsob, kterým by se dalo rozlišování tohoto stisku používat pro každý řádek zvlášť, proto je zavedeno pro úplně celý displej hodinek. Následně je získávána pozice **y**, kde bylo na displeji kliknuto, podle které se dopočítává, o který řádek v tabulce šlo.

Po dlouhém stisku řádku v tabulce se světly se uživateli vyvolá GUI, které je řízeno kontrolérem *SvetloInterfaceController*. Vzhled tohoto rozhraní je možné vidět na obr. 4.17. Jestliže je světlo stmívatelné, může uživatel využít „crown“, tedy otáčivé tlačítko hodinek, pro jeho snížení, respektive zvýšení. Tato informace je pak okamžitě odesílána na server PLCComS. Uživatel tak může sledovat okamžitou změnu. Ovládání pomocí tlačítka „crown“ (označeno na obr. 4.17) se věnoval swift developer Truss [15].

V tomto GUI je možné také měnit stav zapnutí světla, i to se odehrává okamžitě po stisknutí příslušného tlačítka. Poslední možností je nastavení RGB barvy světla. K tomu je využit picker s předem připravenými hodnotami typu Int. Jelikož je při výběru barev nutno procházet přes ostatní možnosti, je potřeba po finálním vybrání potvrdit odeslání nastavení pomocí tlačítka „Nastavit“.

Obr. 4.17 – GUI jednoho světla

4.5.2 Změna stavu světel

Kontroléry disponují rozšířením z kap. 3.3, které deleguje každou přijatou zprávu z mobilního zařízení. Jestliže zpráva obsahuje klíč „data“, dojde k načtení obsahu, který se očekává ve tvaru slovníku slovníků. Klíčem tohoto načteného slovníku je **id** objektu, u kterého nastala změna. Uložená data pod tímto klíčem se následně načtou jako slovník typu [String:String]. V takovém slovníku je pak uložen klíč odpovídající změněnému stavu, tedy „ONOFF“, „DIMLEVEL“, „RGB“ a zmíněný změněný stav.

V jeden okamžik by neměl nastat případ, že přijatá data obsahují informace o více objektech a současně s tím by měla přijít i pouze jedna změněná hodnota. Aplikace je však připravena i na možnost, že přišlo změn více. V případě, že je zobrazeno GUI s tabulkou světel, dochází k její aktualizaci.

Tento princip změn je použit také u všech ostatních kategorií objektů.

4.5.3 Zavření kategorie světel

Jelikož byl na začátku při otevření kategorie zaslán požadavek pro delegování změn zobrazených objektů do PLCComS, je potřeba tuto delegaci ukončit. Ukončování delegování musí probíhat ve funkce jménem **willDissapear**, ovšem při kategorii světel pouze za daných okolností. V případě, že uživatel přechází do rozhraní jednoho světla, dochází k zavolání funkce **willDissapear**, avšak delegování zůstat stále zapnuto.

Řešením je nastavení proměnné **keepDel** na logickou „1“ a „0“. V případě dlouhého stisknutí řádku, a tedy přechodu na rozhraní světla je **keepDel** nastaven na „1“.

Funkce **willDissapear** pak obsahuje kontrolu stavu **keepDel**, jestliže je nulový, vyšle se do mobilního zařízení zpráva, která neočekává žádnou odpověď a je tvaru [„di“:„*“].

4.6 KATEGORIE ŽALUZIÍ

Pro zobrazení GUI kategorie žaluzií je připraven kontrolér jménem *ZaluzieInterfaceController*. Oproti světlům se na každém řádku vpravo nachází aktuální stav jejich pohybu, nebo symbol chybí, jestliže jsou žaluzie v klidovém stavu.

Pro ovládání žaluzií je potřeba vybrat konkrétní objekt žaluzií a jednoduše kliknout na jeho řádek. Proběhne tak přesměrování do GUI, které obstarává kontrolér *Zaluzie1InterfaceController*. Stejně jako v případě světel se nastavuje proměnná **keepDel** aby nedošlo k vypnutí delegací změn.

4.6.1 Rozhraní žaluzie

Do GUI jedné žaluzie se uživatel dostane pomocí jednoduchého stisku řádku v tabulce všech žaluzií. Následně mu jsou k dispozici 4 tlačítka, kterými lze kompletně žaluzie ovládat. Po stisknutí vybraného tlačítka dochází k okamžitému odeslání požadavku na PLCComS. Pro informování uživatele o právě probíhajícím ději žaluzií je uprostřed připraven menší obrázek, symbolizující pohyb žaluzií. Ten je totožný se vzhledem tlačítek.

Obr. 4.18 – GUI žaluzie

4.7 KATEGORIE ZÁSUVEK

Ovládání zásuvek může uživatel provádět z této obrazovky. GUI je obstaráváno kontrolérem pod jménem *ZasuvkyInterfaceController*, ten využívá k zobrazování stejného principu jako všechny ostatní kategorie, tedy jednoduché tabulky. Ovládání pak probíhá přímo z této obrazovky. Pomocí jednoduchého kliknutí dojde k odeslání zprávy ve tvaru [„update“:[„LIGHT“, light.**id**, stav]], přičemž stav je před samotným odesláním nejprve znegován.

Obr. 4.19 – Kategorie zásuvek

4.7.1 Změna stavu zásuvky

Rozšíření pro protokol z kap. 3.3 tentokrát obsahuje kód, který při zprávě, obsahující klíč „data“, prochází **Socket.Sockets** a při shodném **id** zařídí změnu stavu podle právě přijaté hodnoty v „ONOFF“ podobně jako tomu je u kategorie světel. Následně i tentokrát dojde k aktualizaci tabulky, aby uživatel viděl projevení změny.

4.8 KATEGORIE RELÉ

U této kategorie jsem byl nucen zůstat u stejného využívání jako při zásuvkách, tedy kódy a i GUI se chovají úplně stejně s výjimkou toho, že se tentokrát operuje s třídou *Relay* a proměnnou **Relay.Relays**. Kategorie jsem nesjednotil pro případ, kdyby se v budoucnu doplnili očekávané funkce pro relé.

Obr. 4.20 – Kategorie relé

4.9 KATEGORIE DISPLEJŮ

Tato kategorie slouží pouze jako pasivní zobrazovač. Rozhodl jsem se aplikovat stejný vzhled jako při procházení místnostmi a kategoriemi. Je to z důvodu zobrazování obrázku, který určuje typ zařízení. V současné době aplikace pro hodinky zobrazuje pouze displeje s teplotou a vlhkostí, tedy symbol musí být roven 100, respektive 101.

Tlačítko, které je použito u ostatních dvou kontrolérů, bylo nahrazeno pouhým vykreslením obdélníku, displej se totiž nedá nastavovat a slouží pouze ke zobrazování dat.

Obr. 4.21 – Kategorie displejů

4.10 SPOLEČNÉ ROZŠÍŘENÍ PRO KONTROLÉRY

Každý z kontrolérů musí v protokolovém rozšíření, zmíněném v kap. 3.3, povinně obsahovat ošetření pro případné chybové stavy, které by mohli ze strany mobilní aplikace přijít jako zpráva. Následně na ně musí reagovat a provést nucené načtení kontroléru *MainMenuInterfaceController* s kontextem odpovídajícímu nastalé chybě, dle kap. 4.1.

```
if let message = tuple.message["disconnected"] as? String {
    DispatchQueue.main.sync {
        WKInterfaceController.reloadRootControllers(withNames: ["MainMenu"], contexts: ["Disconnected"] as [AnyObject])
    }
}
if let message = tuple.message["plc"] as? String {
    if message == "0" {
        DispatchQueue.main.sync {
            WKInterfaceController.reloadRootControllers(withNames: ["MainMenu"], contexts: ["Plcoff"] as [AnyObject])
        }
    }
}
```

Obr. 4.22 – Ošetření chyb přijatých z mobilu

5 TVORBA APLIKACE PRO IOS

iOS aplikace slouží primárně ke komunikaci s PLCComS pomocí TCP/IP. Práci jsem tedy začal na navázání komunikace. V původním záměru byla použita synchronní komunikace, ke které jsem využíval knihovnu pod názvem SwiftSocket. Ta je volně dostupná ke stažení na GitHubu. Při jejím využívání jsem vždy po odeslání dat serveru vyvolával čtení, u kterého jsem si byl přesně vědom typu navrácených dat.

Nevýhodou této knihovny však byla postrádající možnost asynchronního přijímání dat, které je potřeba pro obdržení informací o změnách v objektech. Tento problém jsem řešil kontrolováním příchozích dat skrze časovač s periodou 100 ms. Tento proces však způsoboval prodlevu při aktualizaci stavu, což bylo dosti nevhodné převážně při změně stavu z hodinek.

Jelikož má být aplikace pro hodinky v budoucnu implementována do aplikace iFoxtrot, docházelo by navíc k dalším komplikacím při jejím přesunu, a tak jsem nakonec pro komunikaci využil převzatou třídu z aplikace iFoxtrot napsanou Pavlem Kohoutkem, u které jsem udělal několik menších změn. Třída zajišťuje asynchronní komunikaci s okamžitým delegováním přijatých vstupních dat a využívá 3 protokoly pro delegaci.

Navázání komunikace

Pro otevření komunikace slouží funkce **connect**, očekávaná vstupní data jsou IP adresa hosta a port. Po úspěšném navázání spojení dochází k delegování metody **streamsOpened** a mobilní zařízení v této delegované metodě odesílá zpětnou odpověď hodinkám o úspěšném navázání komunikace.

Odesílání dat

K odesílání dat se používá funkce **send**, která navíc nejprve zkontroluje, jestli zpráva obsahuje ukončení řádku (\n) a popřípadě jej do zprávy doplní. Následně dojde ke kontrole, zdali je výstup připraven k zápisu dat. Pokud není, dojde k uložení odesílané zprávy do pořadníku.

Události komunikace

Třída je informována o nastalých událostech, které se v komunikaci odehrávají a následně na ně reaguje vhodně zvoleným řešením. Tím dochází k pokrytí všech nastalých situací a je tak vždy možné v případě nutnosti informovat uživatele.

Události jsou následující:

- **errorOccured** – při spojení nastala chyba a je tak nutné spojení ukončit. Delegována je metoda **disconnected** se statusem informujícím o chybě.
- **endEncountered** – spojení bylo ukončeno, opět je delegována metoda **disconnected**, status ale informuje o ukončení spojení.
- **hasBytesAvailable** – při této události jsou čekající data na vstupu, dochází k jejich okamžitému čtení, zakódování do znakové sady WindowsCP1250 a ukládání do zásobníku. Čtení probíhá po celou dobu platnosti tohoto eventu, následně dochází k delegování celého zásobníku do funkce **dataReceived** a poté k jeho vynulování.
- **openCompleted** – otevření komunikace bylo dokončeno, probíhá ověření streamů a jejich statusů. Následně je odsud zavolána výše zmíněná metoda **streamsOpened**.
- **hasSpaceAvailable** – znamená, že výstupní stream je připraven k zápisu dat. Nejprve se prověří zásobník a pokud není prázdný, dochází k odeslání první čekající zprávy. V opačném případě je třída informována o možnosti okamžitého odesílání po přijetí požadavku.

Celá tato třída je s komentáři v příloze B na straně 14.

Komunikace mobilu s hodinkami je odbavena v části aplikace se jménem *AppDelegate*, což je vstupní část aplikace, ze které je nutné ovládat jak komunikaci s hodinkami, tak s PLCComS. K navázání obou spojení a zároveň k načtení uživatelem uložených účtů dochází při ukončení spouštění aplikace.

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        WatchSessionManager.shared.startSession()  //spusteni komunikace Watch Connectivity
        connectivityHandler.iOSDelegate = self     //nastaveni delegace zprav Watch Connectivity
        client1.delegate = self                    //nastaveni delegace zprav z PLCComS
        Accoun.loadAccounts()                      //nacteni ulozenych uctu v mobilu
        return true
}
```

Obr. 5.1 – Aktivace komunikací a načtení účtů na mobilu

5.1 KOMUNIKACE S HODINKAMI

Mobil je při komunikaci s hodinkami převážně pasivní, tedy odesílá pouze odpovědi na předtím obdržené zprávy z hodinek. Ty jsou rozděleny do několika kategorií, podle jejich následného zpracování mobilem. Protože je komunikace s PLCComS asynchronní dochází k zaslání odpovědi z delegačního rozšíření třídy s TCP/IP komunikací. Aplikace si tak musí uchovávat stav právě probíhajícího děje a zároveň i proměnnou pro odpověď, tedy replyHandler. Právě zpracovávaný děj je uložen v proměnné **status** a je typu [String]?, replyHandler je nazván jménem **handler** a musí být typu (([String : Any]) -> Void)?, tedy funkce.

Při obdržení zprávy z hodinek dochází k delegování funkce **messageReceived** podle protokolu zmíněném v kap. 3.3. V této funkci pak rozšíření získá první klíč z obdrženého slovníku. Na jeho základě se pak určuje následující děj v aplikaci. Aplikace rozlišuje 5 hlavních klíčů a jejich význam pro funkci aplikace je popsán v následujících podkapitolách.

5.1.1 Klíč „connection“

Tento klíč přichází z hodinek při prvotním spuštění, nebo v případě nastalé chyby. Mobilní aplikace pak při něm rozlišuje ještě 3 druhy operací, na základě druhého přijatého klíče – connect, initialization a rooms. Popis tohoto programu je v příloze B na straně 9.

Connect

Znamená požadavek hodinek na navázání komunikace s PLCComS, aplikace v takovém případě nejdříve ověří svou proměnnou **connectionStatus**. Jestliže není rovna log. „1“, ověří dostupnost výchozího účtu. Pokud je dostupný, dojde k nastavení proměnné **status** na [connect] a spustí se připojení k PLCComS, v opačném případě je do hodinek navrácena odpověď ve tvaru [„connection“ : „-1“].

Po navázání komunikace dochází k odeslání odpovědi [„connection“ : „1“] z delegované funkce **streamsOpened** a následně dochází k nastavení proměnných **status** a **handler** na **nil**.

Initialization

Tímto příkazem se zahajuje výchozí inicializace všech dostupných objektů projektu v PLC. Aplikace v této části nastaví hodnotu proměnné **status** na [„objects“ : „init“] a odešle 3 zprávy pro PLCComS v tomto pořadí: „EN:*GTSAP1*NAME“ – povolení výpisu změn všech

názvů objektů určených pro ovládání skrze PLCComs, „GET:*GTSAP1*NAME“ – požadavek o vypsání všech jmen a následně „DI:*“ – požadavek na vypnutí delegace všech změn z PLCComS. K dalšímu zpracování dat dochází v delegované funkci **dataReceived**.

Rooms

Požadavek hodinek na načtení souboru s místnostmi z paměti PLC. Na PLCComS je odeslána zpráva ve tvaru "GETFILE:ifox/ifoxtopo.jsn" a **status** je nastaven na [„rooms“], zbytek je opět řešen v rozšíření skrze funkci **dataReceived**.

5.1.2 Klíč „get“

Tento klíč je obdržen v případě, že byla na hodinkách vybrána konkrétní kategorie s objekty. Protože dosud nebyly načteny ostatní informace o objektech a hodinky znají pouze **id** a **name** objektů, zažádali o zjištění ostatních hodnot. Přijatá data tak obsahují pole textových řetězců obsahující **id**.

Aplikace nejdříve zakáže delegaci všech změn z PLCComS a následně postupně delegaci povoluje pro každý z objektů. Poté je zažádáno o získání hodnot všech povolených proměnných. **Status** je tentokrát nastaven na [„objects“].

```
client1.send(data: "DI:*") //zakazani delegace zmen
for a in req[1] as! [String]{ //povoleni delegace zmen pro kazdy objekt
    client1.send(data: "EN:" + a + "*")
}
status = ["objects"]
handler = reply
client1.send(data: "GET:*") //pozadavek na ziskani vsech povolenych hodnot
```

Obr. – 5.2 Kód pro klíč "get"

5.1.3 Klíč „getF“

Použití tohoto klíče je výhradně pro kategorii scén, kdy je potřeba načíst dostupné soubory, kterých může být až 8. V této části je však zadán požadavek na načtení pouze prvního souboru. Vyhodnocení načteného souboru a požadavek na čtení souboru následujícího je opět řízeno z metody **dataReceived**. Proměnná **status** tentokrát obsahuje rozlišení akce, přístupovou cestu souboru a číslo právě načítaného souboru.

```
let file = tuple.message[key] as! String //ziskani pristupove cesty soubory
handler = tuple.replyHandler
status = ["getF", file, "1"] //ulozeni deje, pristuove cesty a cisla souboru
//vyslani pozadavku do PLCComS s nahrazenim "?" za poradove cislo souboru
client1.send(data: "GETFILE:" + file.replacingOccurrences(of: "?", with: "1"))
```

Obr. 5.3 – Kód pro klíč „getF“

5.1.4 Klíč „update“

Při tomto klíči jsou přijata data sloužící pro aktualizaci hodnot objektů v PLC. Jsou ve tvaru [String] a délka tohoto pole je vždy rovna 3. První je v poli uložena informace o kategorii přijatého objektu, druhou hodnotou je samotné **id** objektu a jako poslední je obsažena hodnota. Informace o nastavované proměnné dodána není, protože se u většiny kategorií dá nastavovat pouze jedna. Výjimkou je pak kategorie světel, u které je možné nastavovat až 3 proměnné. Pro tento případ jsem zvolil sloučení všech hodnot do jednoho řetězce v přesně daném pořadí – ONOFF, DIMLEVEL a RGB. Hodnoty jsou rozděleny pomocí znaku „;“ a na základě délky pole po rozdělení, je pak vyhodnoceno, kolik proměnných by se mělo nastavovat. Popsaný funkční kód této části se nachází v příloze B na straně 10. Hodinky v tomto případě neočekávají žádnou odpověď.

5.1.5 Klíč „di“

Posledním využívaným klíčem dochází k požadavku deaktivace delegace všech aktuálně aktivovaných objektů. Tento požadavek je ze strany hodinek obdržen pokaždé, když se uživatel navrátí z vybrané kategorie do přehledu místnosti.

Mobil odešle na PLCComS zprávu ve tvaru „DI:*“.

5.2 DELEGOVANÁ FUNKCE DATARECEIVED

Jak už bylo výše zmíněno, všechna obdržená data z PLCComS jsou přijata skrze tuto funkci. Na začátku funkce dochází k rozdělení dat podle řádků a ověření, zdali je poslední řádek kompletní. Pokud tomu tak není, dochází k jeho uložení a odstranění ze současného zpracování.

Následně dochází k rozlišení typu přijaté zprávy pomocí vytvořené funkce jménem **checkPrefix**. Ta provede kontrolu pro aplikaci podstatných prefixů a v případě shody pak tento prefix navrátí jako svůj výstup. Nedojde-li k rozpoznání prefixu, navrátí se hodnota „UNK“.

```
func checkPrefix(data: String)->String{
    if data.hasPrefix("GET:"){
        return "GET"
    }else if data.hasPrefix("WARNING:"){
        return "WARNING"
    }else if data.hasPrefix("ERROR:"){
        return "ERROR"
    }else if data.hasPrefix("DIFF:"){
        return "DIFF"
    }else if data.hasPrefix("GETFILE:"){
        return "GETFILE"
    }
    return "UNK"
}
```

Obr. 5.4 – Funkce checkPrefix

Na základě rozlišeného prefixu pak dochází k dalšímu zpracování dat.

5.2.1 Prefix GET

Jestliže je rozpoznán prefix GET, dochází ke zpracování přijatých dat na základě několika parametrů. V případě, že přijatá data nejsou rovna „GET:“, tedy nebyl obdržen ukončovací řádek z PLCComS, nastává zpracování přijatého řádku pomocí funkce **modifyData**, které je popsána v příloze B na straně 11, a navrací data ve formátu [String:[String:String]]. Tato funkce navíc disponuje dvěma režimy, které jsou voleny podle aktuálního stavu proměnné **status**. První slouží pro inicializování a navrácená hodnota obsahuje [kategorie:[id objektu:jméno objektu]], v opačném případě se vrací data ve formátu [id objektu[jméno proměnné:hodnota proměnné]]. Z PLCComS byla přijata data, která obsahují výčet úplně všech dostupných objektů, hodinky ale pracují jen s vybranými.

```
func isImportantObject(object: String)->Bool{
    let ImportantObjects = ["SCENE","LIGHT","SHUTTER","SOCKET","RELAY","DISPLAY"]
    for a in ImportantObjects{
        if object == a{
            return true
        }
    }
    return false
```

Obr. 5.5 – Funkce isImportantObject

Proto se ve funkci kontroluje, zdali má dojít ke zpracování dat, pomocí funkce **isImportantObject.** Po zpracování se tyto data ukládají do proměnné **get**, která slouží k následnému odeslání zpět do hodinek.

Pokud došlo k přijetí ukončovacího řádku z PLCComS, dochází nejprve k ověření, zda je proměnná **get** naplněna. V opačném případě dochází k opětovnému požadavku na výpis dat. Následně dochází k přepsání proměnné **status** z :[„objects“, „init“] na :[„objects“, „disp“]. Je tomu tak z důvodu, že v předchozí inicializaci došlo k načtení všech displejů z PLC, hodinky se ale zajímají pouze o displeje, jejichž SYMBOL je roven 100 nebo 101. Mobil tak požádá PLCComS o vypsání hodnoty symbolu pro všechny displeje. Dojde k odstranění nevyhovujících displejů a při dalším přijetí ukončujícího řádku dochází k odeslání všech zpracovaných dat do hodinek. Kód obstarávající zpracování dat s prefixem GET je k dispozici v příloze B na straně 12.

5.2.2 Prefix GETFILE

Při tomto prefixu je aplikace informována o začínajícím přijetí souboru. PLCComS ale posílá tento prefix pouze na začátku každého odesílaného datového bloku. Aplikace tak zpracování příchozího souboru řeší ze dvou různých míst.

Rozlišuje se mezi dvěma různými typy souboru, a to na základě proměnné **status.** V případě, že je **status[0]** nastaven na "rooms", dochází ke zpracování souboru s místnostmi. Jestliže nebyl obdržen konec souboru (GETFILE:ifox/ifoxtopo.jsn[0]=), dochází k odstranění hlavičky dat, rozdělením pomocí znaku „=“. Data jsou následně zapsána do připravené proměnné pro uchovávání souboru – **getfile**. Při obdržení informace o dokončení celého souboru s místnostmi se volá funkce **getRooms**, která se pokusí o zpracování souboru nejprve na jsonObject a následně jej roztřídí do formátu [Int:[String:AnyObject]], přičemž Int určuje pořadí místnosti, String název kategorie dat a AnyObject pak obsahuje samotná data. Pokud se nepovede soubor převést na jsonObject, dochází k navrácení hodnoty [0:[“ERROR“:["2"]]]. Navrácený slovník slovníků je nakonec odeslán do hodinek.

Pokud je **status[0]** roven „getF“, jedná se o zpracování 8 souborů obsahujících nastavení scén. Funkční princip je stejný jako byl u místností, při přijetí informace o konci souboru dochází k jeho zpracování funkcí **processSceneFile**. Ta soubor také převede na jsonObject, ale tentokrát se ze souboru získává pouze jméno a index scény. Tyto data jsou ukládány do slovníku **sceneName**. Poté co dojde k ukončení čtení všech souboru je odeslán obsah slovníku do hodinek.

5.2.3 Prefix DIF

Při příchodu informace o změně stavu některého ze sledovaných objektů PLCComS dochází nejprve k ověření, zdali nebyla přijata změna stavu PLC, např. jeho vypnutí. Pokud se tak stalo, dochází k nastavení proměnné **cancel** na log. „1“ díky čemuž se následně zruší všechny dosud probíhané děje a do hodinek je odeslána zpráva o nastalé události.

V opačném případě opět dochází ke zpracování dat funkcí **modifyData**, bez nastavení inicializace, a jejich následnému odeslání do hodinek.

Oba případy využívají k odesílání dat funkci posílající zprávu do hodinek. Nejedná se tedy o odpověď na předchozí požadavek ze strany hodinek a hodinky musí tyto data zpracovávat v rozšíření WatchSessionManager.

5.2.4 Prefix ERROR

Při informaci ze strany PLCComS o chybě dochází k jejímu zpracování právě v této části kódu. Z přijatých dat je získán číselný kód, který odkazuje na konkrétní chybu podle tab. 1.2. V současné době aplikace reaguje pouze na kód 41 – tedy na chybu při získávání souboru z PLC. Ta může nastat při získávání souboru místností, v takovém případě dochází jen k odeslání odpovědi ve tvaru [0:“ERROR“: [“1“]] do hodinek, nebo při získávání souboru scény.

Pokud je přijata chyba právě z důvodu neexistujícího souboru scény, musí dojít k ověření, jestli se přijatá chyba týká již posledního, tedy osmého, souboru. Na základě toho pak aplikace vyhodnotí, jestli má již odeslat data **sceneName** do hodinek anebo požádat o další soubor v pořadí.

5.2.5 Prefix WARNING

Jak bylo zmíněno dříve, aplikace reaguje pouze na výstrahu s kódovým označením 250. Při zjišťování kódu se postupuje stejně jako v případě přijaté zprávy s prefixem ERROR. Po zjištění této výstrahy dojde opět k nastavení proměnné **cancel** na log. „1“ a následně je do hodinek odeslána zpráva vynucující opětovnou inicializaci celého souboru PUBFILE.

5.2.6 Prefix UNK

Situace nastane za předpokladu, že přijatý řádek neobsahuje žádný ze sledovaných prefixů. V takovém případě se provede kontrola aktuální hodnoty proměnné **status** a jestliže je nastaven na získávání souboru místností, nebo scény. Dochází k přidání všech obdržených dat do proměnné **getFile**.

5.3 GUI PRO PŘIDÁNÍ NOVÉHO ÚČTU

Slouží k přidání nového účtu pro připojení k PLCComS. Vzhled tohoto rozhraní jsem se pokusil vytvořit podle předchozí představy, která vychází z aplikace iFoxtrot. Uživatel musí vždy povinně vyplnit všechna dostupná vstupní pole, přičemž jméno účtu musí být vždy unikátní a port datového typu Int32.

Při stisknutí tlačítka **Uložit** dochází nejprve ke kontrole vyplnění všech polí. Pokud aplikace zjistí nevyplněné pole, informuje uživatele o nutnosti jeho vyplnění a následně se kurzor přesune do prázdného textového pole.

V případě, že jsou všechna pole vyplněna, dochází k validaci vstupních dat. Nejprve je ověřena unikátnost zvoleného jména účtu, jestliže je v pořádku, následuje kontrola vyplněného portu. Poté je učet inicializován a přidán do pole již existujících účtů.

Obr. 5.6 – GUI pro přidání nového účtu

Aplikace využívá vlastní třídu pro spravování všech dostupných účtů. Ta je napsána ve stejném stylu, jako třídy pro kategorie objektů v hodinkách. Navíc však obsahuje ještě funkce pro načtení účtů – **loadAccounts** a také jejich uložení – **saveAccounts.**

5.4 GUI S PŘEHLEDEM VŠECH ÚČTŮ

Výchozím rozhraním mobilní aplikace je přehled všech doposud uložených účtů. Ty jsou ihned po spuštění načteny do paměti pomocí funkce **loadAccounts**. Následně má uživatel přehled o všech informacích z jednotlivých účtů současně s možností změny účtu výchozího. To se provádí stisknutím tlačítka „VÝCHOZÍ“ u požadovaného účtu.

Navíc je možné vyvolat úpravu účtu, která otevře GUI pro přidání nového účtu s již předvyplněnými údaji. Po vzoru aplikace iFoxtrot se neprovádí modifikace účtu jako takového, ale vytváří se účet nový, který může starý účet duplikovat, krom stejného názvu.

Poslední možností u účtů je jeho smazání. V takovém případě dojde k jeho odstranění a následně také k okamžitému přepsání uloženého souboru funkcí **saveAccounts.**

K otevření rozhraní pro přidání nového účtu dojde při zjištění chybějícího souboru „wFoxtrotAccounts.jsn“ v kořenovém adresáři aplikace. Nebo jej může uživatel vyvolat sám, pomocí kliknutí na tlačítko „+“ v pravém horním rohu.

Při práci s tabulkami na iOS se postupuje jiným způsobem, než tomu je u hodinek, z tohoto důvodu jsem se řídil postupem podle [16]. Problematiku spjatou s používáním tlačítka v daném řádku tabulky jsem řešil použitím návodu [17].

6 ZHODNOCENÍ

Vytvořená mobilní aplikace zajišťuje komunikaci s PLCComS a zprostředkovává data pro hodinky. Pro komunikaci mobilu s hodinkami se používá WatchConnectivity, pomocí které jsou odesílány předem zpracovaná data. Jelikož aplikace iFoxtrot pro svou funkci také obstarává data z PLCComS a zároveň je i uchovává, je možné do aplikace pouze přidat rozšíření WatchConnectivity pro odesílání těchto dat do aplikace na hodinkách. Tím je tak zajištěna snadná integrace.

Aplikace na hodinkách nabízí přehledné menu všech místností, ve kterém se uživatel pohybuje pomocí jednoduchých gest potahem vpravo nebo vlevo. Navíc je přidána místnost obsahující výčet všech objektů, které hodinky obsluhují.

Po výběru místnosti je uživatel přesměrován na velmi podobné GUI, ve kterém se nachází výčet všech objekty naplněných kategorií. Hodinky jsou v tuto chvíli schopny spravovat kategorie scén, světel, žaluzií, zásuvek, relé a displejů, mezi nimi se uživatel pohybuje pomocí stejného principu, jako v přehledu místností.

V případě výběru kategorie dochází ke zobrazení přiřazených objektů kategorie do tabulky. Díky tomu je uživatel schopen rychle procházet celý seznam objektů. Na základě vybrané kategorie je pak někdy možné ovládat objekty přímo z tohoto přehledu. V případě, kdy je ovládání komplikovanější, je nutné objekt rozkliknout.

Současná verze aplikace je schopná ovládat scény, rozsvěcet světla, nastavovat jejich stmívání a také barvu. Dále je možné kompletní ovládání žaluzií a zapínání, respektive vypínání, zásuvek a relé. Kategorie s displeji slouží k výpisu teploty, nebo vlhkosti, v místnosti.

LITERATURA

[1] BLAŽEK J. PLC Foxtrot s vestavěným WEB serverem. [Online]. *Blaja automation portal*. 2010. [cit. 2019-05-12]. Dostupné z: https://www.blaja.cz/archiv-clanku/plc-foxtrot-s-vestavenym-web-serverem.html

[2] Služba TecoRoute druhé vydání. [Online]. Tecomat.cz. 2013. [cit. 2019-05-12]. Dostupné z: https://www.tecomat.cz/download/get/txv00338_01_tecoroute_cz/96

[3] Komunikační server PLCComS 19. vydání. [Online]. *Tecomat.cz*. 2018. [cit. 2019-05-12]. Dostupné z: https://www.tecomat.cz/download/get/txv13863_01_plccoms_cz/156

[4] Knihovna iControlLib 2. vydání. [Online]. *Tecomat.cz*. 2017. [cit. 2019-05-12]. Dostupné z: https://www.tecomat.cz/uploads/files/DOCS/cze/TXV00359_01_Mosaic_iControlLib.pdf

[5] Nastavení PLCComS a názvy proměnných v .PUB (pro použití s mobilní aplikací), verze 25. [Online]. *Tecomat.cz* [cit. 2019-05-12]. Dostupné z: https://www.tecomat.cz/uploads/files/DOCS/cze/PRINTS/iFoxtrot-PLCComS-protocol.v26.pdf

[6] MARSHALL C. Can the Apple Watch work without an iPhone? [Online]. *Techradar.com*. 2018. [cit. 2019-05-12]. Dostupné z: https://www.techradar.com/news/wearables/can-the-apple-watch-work-without-an-iphone-1288140

[7] App Programming Guide for watchOS: The Watch App Architecture [Online]. *Developer.apple.com*. 2016. [cit. 2019-05-12]. Dostupné z: https://developer.apple.com/library/archive/documentation/General/Conceptual/WatchKitProgrammingGuide/DesigningaWatchKitApp.html

[8] ESKYMO. Networking: Socket communication does not work real Apple Watch (wOS 2) [Online]. *Forums.developer.apple.com*. 2015. [cit. 2019-05-12]. Dostupné z: https://forums.developer.apple.com/thread/19399

[9] Watch Connectivity in Swift – Application context. [Online]. *CodingExplorer*. 2019. [cit. 2019-05-12]. Dostupné z: https://www.codingexplorer.com/watch-connectivity-swift-application-context/

[10] App Programming Guide for watchOS: Sharing Data. [Online]. *Developer.apple.com*. 2012. [cit. 2019-05-12]. Dostupné z: https://developer.apple.com/library/archive/documentation/General/Conceptual/WatchKitProgrammingGuide/SharingData.html

[11] iFoxtrot uživatelská příručka, v2.4.1-2110. [Online]. *Tecomat.cz*. [cit. 2019-05-12]. Dostupné z: https://www.tecomat.cz/uploads/files/DOCS/cze/PRINTS/iFoxtrot-manual-2.4.1-2110-v2-cs.pdf

[12] LITO. WatchOS 5 — Communication between iPhone and Apple Watch and vice versa on Swift — Part 4. [Online]. *Medium*. 2018. [cit. 2019-05-12]. Dostupné z: https://medium.com/@litoarias/watchos-5-communication-between-iphone-and-apple-watch-and-vice-versa-on-swift-part-4-394df1d47644

[13] WCSession - WatchConnectivity | Apple Developer Documentation. [Online]. *Developer.apple.com*. [cit. 2019-05-12]. Dostupné z: https://developer.apple.com/documentation/watchconnectivity/wcsession

[14] AUDREY, T. watchOS 4 Tutorial Part 2: Tables. *raywenderlich.com* [online]. 2017. [cit. 2019-05-13]. Dostupné z: https://www.raywenderlich.com/288-watchos-4-tutorial-part-2-tables

[15] TRUSS, K. Using the crown to control a slider (and label) in WatchOS3. [Online]. *Medium*. 2017. [cit. 2018-05-18]. Dostupné z: https://medium.com/@swiftkicksoft/using-the-crown-to-control-a-slider-and-label-in-watchos3-1729cf9ba0dd

[16] KNOPPER, A. Add Row to Table View iOS Tutorial. [Online]. *iOScreator*. 2018. [cit. 2019-05-19]. Dostupné z: https://www.ioscreator.com/tutorials/add-rows-table-view-ios-tutorial-ios12

[17] ZOVE, J. How to properly do buttons in table view cells using Swift closures. [Online]. *CandyCode.io*. 2016. [cit. ¨2019-05-19]. Dostupné z: https://web.archive.org/web/20160730215620/http://candycode.io/how-to-properly-do-buttons-in-table-view-cells-using-swift-closures/

[18] FIŠAR, M. *Ovládání inteligentního domu pomocí chytrých hodinek*. Praha, 2019. Projekt 2. ČVUT. Vedoucí práce Ing. Jan Martinec.

PŘÍLOHY

A – CD

B – Vybrané části kódů

C – Uživatelský manuál

Příloha k diplomové práci

OVLÁDÁNÍ INTELIGENTNÍHO DOMU POMOCÍ CHYTRÝCH HODINEK

Michal Fišar

CD

Příloha A

Obsah

Příloha k diplomové práci

OVLÁDÁNÍ INTELIGENTNÍHO DOMU POMOCÍ CHYTRÝCH HODINEK

Michal Fišar

Vybrané části kódů

Příloha B

Obsah

1 Funkce initialization

```
1.  //funkce tridi objekty na zaklade kategorie a nasledne provadi jejich prvotni inicializaci.
2.  //soucasne s tim informuje uzivatele o aktualnim deji
3.      func initialization(data: [String:[String:String]]) {
4.          //vyprazdneni vsech doposud inicializovanych objektu
5.          Scene.Scenes = []
6.          Light.Lights = []
7.          Shutter.Shutters = []
8.          Socket.Sockets = []
9.          Relay.Relays = []
10.         Display.Displays = []
11.         let objects = data
12.         var cur = 0 //poradi aktualniho objektu z kategorie
13.         let types: [String] = ["SCENE","LIGHT","SHUTTER","SOCKET","RELAY","DISPLAY"]
14.         let names: [String] = ["Scény: ","Světla: ","Žaluzie: ","Zásuvky: ","Relé:
    ", "Displeje: "]
15.         for a in 0...5 { //nacteni dat ze slovniku podle typu objektu
16.             if objects[types[a]] != nil {
17.                 self.initObj.setText(names[a]) //vypsani kategorie
18.                 let data = objects[types[a]] //nacteni slovniku dat pro kategorii
19.                 cur = 0
20.                 //zobrazeni poradi objektu / z kolika objektu pro uzivatele
21.                 self.initNum.setText(String(cur) + "/" + String(data!.keys.count))
22.                 self.initFinal.setHidden(false)
23.                 for key in data!.keys {
24.                     cur += 1
25.                     self.initNum.setText(String(cur) + "/" + String(data!.keys.count))
26.                     //roztrideni dat podle kategorii a jejich nasledna inicializace
27.                     switch types[a] {
28.                     case "SCENE":
29.                         Scene.addScene(id: key, name: data![key]!)
30.                         break
31.                     case "LIGHT":
32.                         Light.addLight(id: key, name: data![key]!)
33.                         break
34.                     case "SHUTTER":
35.                         Shutter.addShutter(id: key, name: data![key]!)
36.                         break
37.                     case "SOCKET":
38.                         Socket.addSocket(id: key, name: data![key]!)
39.                         break
40.                     case "RELAY":
41.                         Relay.addRelay(id: key, name: data![key]!)
42.                         break
43.                     case "DISPLAY":
44.                         Display.addDisplay(id: key, name: data![key]!)
45.                     default:
46.                         break
47.                     }
48.                 }
49.             }
50.         }
51.     }
```

2 Funkce rooms

```
1.  //inicializace mistnosti z prijatych dat
2.      func rooms(data: [Int:[String:AnyObject]]) -> Int{
3.          //vyprazdneni doposud ulozenych mistnosti
4.          Room.Rooms = []
5.          initG.setHidden(false)
6.          roomsG.setHidden(true)
7.          //informovani uzivatele o probihanem deji
8.          initObj.setText("Místnosti: ")
9.          initNum.setText("0/" + String(data.count))
10.         initNum.setHidden(false)
11.         initFinal.setHidden(false)
12.         //kontrola jestli nastal error
13.         if data[0]!["ERROR"] != nil{
14.             let error = data[0]!["ERROR"]! as! [String]
15.             switch error {
16.             //ERROR 1 - soubor s mistnostmi neexistuje, inicializace obecne mistnosti
17.             case ["1"]:
18.                 Room.Rooms.append(Room.init(name: "VŠE", symbol: 1, scenes: [-1], lights: [-
    1], shutters: [-1], sockets: [-1], relays: [-1], displays: [-1]))
19.                 return 0
20.                 break
21.             //ERROR 2 - chyba pri zpracovani souboru mistnosti na strane mobilu
22.             case ["2"]:
23.                 return -1
24.                 break
25.             default:
26.                 break
27.             }
28.         } else {
29.             //vlastni inicializace mistnosti
30.             for a in 0...(data.count-1){
31.                 initNum.setText(String(a) + "/" + String(data.count))
32.                 let b = data[a] as! [String:AnyObject]
33.                 let c = b["devices"] as! [String]
34.                 //priprava poli pro zapis objektu mistnosti
35.                 var lights: [Int] = []
36.                 var relays: [Int] = []
37.                 var scenes: [Int] = []
38.                 var shutters: [Int] = []
39.                 var sockets: [Int] = []
40.                 var displays: [Int] = []
41.                 //roztrideni kazdeho z objektu mistnosti na zaklade kategorie
42.                 for i in c {
43.                     if Scene.getSceneIndex(id: i) != nil {
44.                         scenes.append(Scene.getSceneIndex(id: i)!)
45.                     } else if Light.getLightIndex(id: i) != nil {
46.                         lights.append(Light.getLightIndex(id: i)!)
47.                     } else if Shutter.getShutterIndex(id: i) != nil {
48.                         shutters.append(Shutter.getShutterIndex(id: i)!)
49.                     } else if Socket.getSocketIndex(id: i) != nil {
50.                         sockets.append(Socket.getSocketIndex(id: i)!)
51.                     } else if Relay.getRelayIndex(id: i) != nil {
52.                         relays.append(Relay.getRelayIndex(id: i)!)
53.                     } else if Display.getDisplayIndex(id: i) != nil {
54.                         displays.append(Display.getDisplayIndex(id: i)!)
55.                     }
56.                 }
57. //inicializace mistnosti s objekty
58. Room.Rooms.append(Room.init(name: b["name"]! as! String, symbol: Int(b["symbol"]! as! String
    )!, scenes: scenes, lights: lights, shutters: shutters, sockets: sockets, relays: relays, di
    splays: displays))
59.             }
```

```
60.             //pridani obecne mistnosti na konec seznamu
61.             Room.Rooms.append(Room.init(name: "VŠE", symbol: 1, scenes: [-1], lights:
                [-1], shutters: [-1], sockets: [-1], relays: [-1], displays: [-1]))
62.             return 1
63.         }
64.         return -1
65.     }
```

3 Stisknutí tlačítka kategorie

```
1.  //pri stisknuti tlacitka se zada o nacteni zbylych informaci mobilem
2.  //nasledne jsou data doplnena a nactena kategorie dle predchoziho vyberu
3.      @IBAction func btPressed(){
4.          var data: [String] = []
5.          //zakazani gest = nelze zmenit kategorii
6.          canSwipe = false
7.          //zobrazeni animace v pravem hornim rohu GUI
8.          loadingG.setHidden(false)
9.          loadingG.setBackgroundImageNamed("Loading")
10.         loadingG.startAnimatingWithImages(in: NSRange(location: 0, length: 20), duration: 1.
    0, repeatCount: 20)
11.         //na zaklade zvolene kategorie dochazi k vyberu dat
12.         //pokud kategorie mistnost obsahuje [-1], dojde k nacteni vsech objektu kategorie
13.         switch av[group]{
14.         case 0:
15.             if Room!.scenes != [-1]{
16.                 for a in Room!.scenes{
17.                     data.append(Scene.Scenes[a].id)
18.                 }
19.             }else{
20.                 for a in Scene.Scenes{
21.                     data.append(a.id)
22.                 }
23.             }
24.             break
25.         case 1:
26.             if Room!.lights != [-1]{
27.                 for a in Room!.lights{
28.                     data.append(Light.Lights[a].id)
29.                 }
30.             }else{
31.                 for a in Light.Lights{
32.                     data.append(a.id)
33.                 }
34.             }
35.             break
36.         case 2:
37.             if Room!.shutters != [-1]{
38.                 for a in Room!.shutters{
39.                     data.append(Shutter.Shutters[a].id)
40.                 }
41.             }else{
42.                 for a in Shutter.Shutters{
43.                     data.append(a.id)
44.                 }
45.             }
46.             break
47.         case 3:
48.             if Room!.sockets != [-1]{
49.                 for a in Room!.sockets{
50.                     data.append(Socket.Sockets[a].id)
51.                 }
52.             }else{
53.                 for a in Socket.Sockets{
54.                     data.append(a.id)
55.                 }
56.             }
57.             break
58.         case 4:
59.             if Room!.relays != [-1]{
60.                 for a in Room!.relays{
61.                     data.append(Relay.Relays[a].id)
```

```
62.                 }
63.             }else{
64.                 for a in Relay.Relays{
65.                     data.append(a.id)
66.                 }
67.             }
68.             break
69.         case 5:
70.             if Room!.displays != [-1]{
71.                 for a in Room!.displays{
72.                     data.append(Display.Displays[a].id)
73.                 }
74.             }else{
75.                 for a in Display.Displays{
76.                     data.append(a.id)
77.                 }
78.             }
79.             break
80.         default:
81.             break
82.         }
83.         //odeslani dat do mobilu ve tvaru ["get" : [kategorie, data]
84.         self.connectivityHandler.sendMessage(message: ["get" :
            [self.images[self.av[self.group]], data] as AnyObject] , replyHandler: {(response)in
85.              //nacteni odpovedi do odpovidajiciho slovniku
86.             let res = response["get"] as! [String:[String:String]]
87.             let dat = res.keys
88.             for a in dat{ //cyklicke zpracovani prijatych dat pro vsechny prijate id
89.                 let obj = res[a] as! [String:String] //nacteni podslovniku pro dane id
90.                 //zpracovani prijatych dat v zavislosti na vybranem typu objektu
91.                 switch self.av[self.group]{
92.                 case 0:
93.                     let ind = Scene.getSceneIndex(id: a)!
94.                     Scene.Scenes[ind].name = obj["NAME"]!
95.                     Scene.Scenes[ind].file = obj["FILE"]!
96.                     break
97.                 case 1:
98.                     let ind = Light.getLightIndex(id: a)!
99.                     Light.Lights[ind].name = obj["NAME"]!
100.                            if obj["ONOFF"]! == "1" {
101.                                Light.Lights[ind].onoff = true
102.                            } else {
103.                                Light.Lights[ind].onoff = false
104.                            }
105.                            //overeni stmivatelnosti
106.                            if obj["TYPE"]! == "1" {
107.                                Light.Lights[ind].type = true
108.                                Light.Lights[ind].dimlevel = Double(obj["DIMLEVEL"]!)
109.                                //overeni moznosti nastaveni barvy
110.                                if obj["DIMTYPE"]! == "1" {
111.                                    Light.Lights[ind].dimtype = true
112.                                    Light.Lights[ind].RGB = UInt(obj["RGB"]!)
113.                                } else {
114.                                    Light.Lights[ind].dimtype = false
115.                                }
116.
117.                            } else {
118.                                Light.Lights[ind].type = false
119.                            }
120.                            break
121.                        case 2:
122.                            let ind = Shutter.getShutterIndex(id: a)!
123.                            Shutter.Shutters[ind].name = obj["NAME"]!
124.                            Shutter.Shutters[ind].run = obj["RUN"]!
125.                            Shutter.Shutters[ind].up = obj["UP"]!
126.                            Shutter.Shutters[ind].uppos = obj["UPPOS"]!
```

```
127.                            Shutter.Shutters[ind].down = obj["DOWN"]!
128.                            Shutter.Shutters[ind].downpos = obj["DOWNPOS"]!
129.                            break
130.                        case 3:
131.                            let ind = Socket.getSocketIndex(id: a)!
132.                            Socket.Sockets[ind].name = obj["NAME"]!
133.                            if obj["ONOFF"]! == "1" {
134.                                Socket.Sockets[ind].onoff = true
135.                            } else {
136.                                Socket.Sockets[ind].onoff = false
137.                            }
138.                            break
139.                        case 4:
140.                            let ind = Relay.getRelayIndex(id: a)!
141.                            Relay.Relays[ind].name = obj["NAME"]!
142.                            Relay.Relays[ind].symbol = obj["SYMBOL"]!
143.                            Relay.Relays[ind].type = Int(obj["TYPE"]!)
144.                            if obj["ONOFF"]! == "1" {
145.                                Relay.Relays[ind].onoff = true
146.                            } else {
147.                                Relay.Relays[ind].onoff = false
148.                            }
149.                            break
150.                        case 5:
151.                            let ind = Display.getDisplayIndex(id: a)!
152.                            Display.Displays[ind].name = obj["NAME"]!
153.                            Display.Displays[ind].symbol = obj["SYMBOL"]!
154.                            Display.Displays[ind].precision = obj["PRECISION"]!
155.                            Display.Displays[ind].value = obj["VALUE"]!
156.                            Display.Displays[ind].unit = obj["UNIT"]!
157.                            break
158.                        default:
159.                            break
160.                                }
161.                            }
162.            DispatchQueue.main.async {
163.              switch self.av[self.group]{
164.              case 0:
165.                 self.presentController(withName: "Scenes", context: self.Room?.scenes)
166.                 break
167.              case 1:
168.                 self.presentController(withName: "Lights", context: self.Room?.lights)
169.                 break
170.              case 2:
171.                 self.presentController(withName: "Shutters", context: self.Room?.shutters)
172.                 break
173.              case 3:
174.                 self.presentController(withName: "Sockets", context: self.Room?.sockets)
175.                 break
176.              case 4:
177.                 self.presentController(withName: "Relays", context: self.Room?.relays)
178.                 break
179.              case 5:
180.                 self.presentController(withName: "Displays", context: self.Room?.displays)
181.                 break
182.              default:
183.                                    break
184.                                }
185.                            }
186.                        }) {(error) in
187.          DispatchQueue.main.async {
188.               WKInterfaceController.reloadRootControllers(withNames: ["MainMenu"], contexts:
                     ["WKc"] as [AnyObject])
189.                            }
190.                        }
191.                    }
```

4 Klíč connection

```
case "connection":
            let req = tuple.message[key] as! String
            switch req {
            //pozadavek na pripojeni k PLCComS
            case "connect":
                //overeni jestli je mobil pripojen k PLCComS
                if connectionStatus == false {
                    if Accoun.Default != nil { //kontrola defaultniho uctu pro PLCComS
                        status = ["connect"]
                        client1.delegate = self
                        //pripojeni k PLCComS
                        client1.connect(host: Accoun.Default!.ipadr, port: UInt32(Accoun.Defa
ult!.port))
                    } else {
                        handler!(["connection" : "-1" as AnyObject])
                        handler = nil
                    }
                } else {
                    handler!(["connection" : "1" as AnyObject, "alias" : Accoun.Default!.alia
s as AnyObject])
                    handler = nil
                }
                break
            //pozadavek na uvodni inicializaci
            case "initialization":
                client1.send(data: "EN:*GTSAP1*NAME")
                status = ["objects", "init"]
                client1.send(data: "GET:*")
                break
            default:
                break
            }
            break
```

5 Klíč update

```
case "update":
            let req = tuple.message[key] as! [String]
            //odeslani pozadavku na PLCComS na zaklade prijate kategorie
            if req[0] == "SCENE"{
                client1.send(data: "SET:" + req[1] + ".GTSAP1_SCENE_SET" + req[2] + ",1\n")
                handler!(["update" : "OK" as AnyObject])
            }
            if req[0] == "SOCKET"{
                client1.send(data: "SET:" + req[1] + "_SOCKET_ONOFF," + req[2] + "\n" )
                handler!(["update" : "OK" as AnyObject])
            }
            if req[0] == "RELAY" {
                client1.send(data: "SET:" + req[1] + "_RELAY_ONOFF," + req[2])
                handler!(["update" : "OK" as AnyObject])
            }
            if req[0] == "SHUTTER" {
                client1.send(data: "SET:" + req[1] + "_SHUTTER_" + req[2] + "_CONTROL,1")
                handler!(["update" : "OK" as AnyObject])
            }
            if req[0] == "LIGHT" {
                let dx = req[2].split(separator: ";")
                //v pripade svetla je potreba overit kolik hodnot se ma nastavit
                switch dx.count {
                case 1:
                    client1.send(data: "SET:" + req[1] + "_LIGHT_ONOFF," + dx[0])
                    break
                case 2:
                    client1.send(data: "SET:" + req[1] + "_LIGHT_ONOFF," + dx[0])
                    client1.send(data: "SET:" + req[1] + "_LIGHT_DIMLEVEL," + dx[1])
                    break
                case 3:
                    client1.send(data: "SET:" + req[1] + "_LIGHT_ONOFF," + dx[0])
                    client1.send(data: "SET:" + req[1] + "_LIGHT_DIMLEVEL," + dx[1])
                    client1.send(data: "SET:" + req[1] + "_LIGHT_RGB," + dx[2])
                    break
                default:
                    break
                }
                handler!(["update" : "OK" as AnyObject])
            }
            handler = nil
            break
```

6 Funkce modifyData

```
1.  func modifyData(data: String, inic: Bool)->[String:[String:String]]?{
2.          let a = data.split(separator: ":", maxSplits: 1) // odstraneni prefixu, data v a[1]
3.          // ziskani id objektu (v b[0]), zbytek b[1]
4.          let b = a[1].components(separatedBy: "GTSAP1_")
5.          var c: Array<Substring> = []
6.          if b.count>1{
7.               // ziskani typu objektu (c[0]), jmena a hodnoty promenne (c[1])
8.               c = String(b[1]).split(separator: "_", maxSplits: 1)
9.              if c.count>2{
10.                 return nil
11.             }
12.         }else{
13.             return nil
14.         }
15.         //kontrola jestli je objekt podstatny
16.         if  isImportantObject(object: String(c[0])){
17.             var out: [String:[String:String]] = [:]
18.             var dat: [String:String] = [:]
19.             // rozdeleni jmena (d[0]) a hodnoty (d[1]) promenne
20.             var d = c[1].split(separator: ",", maxSplits: 1)
21.             //odstraneni uvozovek v pripade ze se jedna o NAME nebo FILE
22.             if d[0] == "NAME" || d[0] == "FILE"{
23.                 d[1] =
    d[1][d[1].index(d[1].startIndex, offsetBy: 1)...d[1].index(d[1].endIndex, offsetBy: -2)]
24.             }
25.             //pokud neni inicializace, zapise se jmeno promenne a jeji hodnota
26.             //a nasledne se pridaji do slovniku pod id objektu
27.             if inic == false {
28.                 dat[String(d[0])] = String(d[1])
29.                 print(data)
30.                 out[String(b[0]) + "GTSAP1"] = dat
31.                 print(out)
32.                 return out
33.             } else {
34.                 //pri inicializaci se zapise id objektu a jeho jmeno
35.                 if out[String(c[0])] != nil {
36.                     dat = out[String(c[0])]!
37.                     dat[String(b[0]) + "GTSAP1"] = String(d[1])
38.                 } else {
39.                     dat[String(b[0]) + "GTSAP1"] = String(d[1])
40.                 }
41.                 out[String(c[0])] = dat
42.                 return out
43.             }
44.         }
45.         return nil
46.     }
```

7 Prefix GET

```
case "GET":
                //overeni jestli prisel ukoncujici radek
                if a == "GET:" {
                    //kontrola zda doslo k naplneni
                    //jestlize ne, zada se o ne znovu
                    if get == [:] && status![1] == "objects" {
                        if askAgain {
                            client1.send(data: "GET:*")
                            askAgain = false
                        } else {
                            client1.send(data: "DI:*")
                        }
                        break
                    } else if displays == [:] && status == ["objects","disp"] {
                        if askAgain {
                            client1.send(data: "GET:*")
                            askAgain = false
                        } else {
                            client1.send(data: "DI:*")
                        }
                    } else {
                        if status != ["objects"] {
                            client1.send(data: "DI:*")
                        }
                    }
                    print(status)
                    if stat[0] == "objects" {
                        if self.cancel {
                            handler!(["cancel" : "1" as AnyObject])
                        } else {
                            if stat.count == 2 {
                                if stat[1] == "disp" {
                                    get["DISPLAY"] = displays
                                    displays = [:]
                                  handler!(["connection" : get as AnyObject, "initialization"
                                   : "1" as AnyObject])
                                    get = [:]
                                    handler = nil
                                    status = nil
                                    askAgain = true
                                }
                                if stat[1] == "init" {
                                    status![1] = "disp"
                                    client1.send(data: "DI:*")
                                    client1.send(data: "EN:*DISPLAY*SYMBOL")
                                    client1.send(data: "GET:*")
                                    askAgain = true
                                }
                            } else {
                                print(get)
                                handler!(["get" : get as AnyObject])
                                get = [:]
                                handler = nil
                                status = nil
                            }
                        }
                    }
                    break
                }
                //pokud nebyl prijat ukoncujici radek, dochazi ke zpracovani dat
                var x: [String:[String:String]]?
                print(stat)
                if stat.count == 2 {
```

```
63.                     //provadeni inicializace
64.                     if stat[1] == "init" {
65.                         x = modifyData(data: a, inic: true)
66.                     }
67.                     //kontrola jestli displej obsahuje symbol 100/101
68.                     if stat[1] == "disp" {
69.                         let data = a.components(separatedBy: ":")
70.                         let dispD = data[1].components(separatedBy: "_")
71.                         print(dispD[0])
72.                         let disp = get["DISPLAY"]!
73.                         print(disp[dispD[0]])
74.                         if dispD.count > 1 {
75.                             let symbol = dispD.last!.components(separatedBy: ",")
76.                             if symbol[1] == "100" || symbol[1] == "101" {
77.                                 displays[dispD[0]] = disp[dispD[0]]!
78.                             }
79.                         }
80.                     }
81.                 } else {
82.                     //ziskavani informaci o objektu
83.                     x = modifyData(data: a, inic: false)
84.                 }
85.                 //prirazeni dat do slovniku, tak aby nedoslo k prepsani
86.                 //jiz existujicich dat
87.                 if x != nil {
88.                     if get[x!.keys.first!] != nil {
89.                         var gt = get[x!.keys.first!]!
90.                         let dx = x![x!.keys.first!]!
91.                         gt[dx.keys.first!] = dx[dx.keys.first!]
92.                         get[x!.keys.first!] = gt
93.                     } else {
94.                         let gt = x![x!.keys.first!]
95.                         get[x!.keys.first!] = gt
96.                     }
97.                 }
98.                 break
```

8 Komunikace s PLCComS

```
1.  //delegacni protokol
2.  protocol SocketConnectionDelegate {
3.      func dataReceived(_ data: String)
4.      func streamsOpened()
5.      func disconnectStatus(_ status: connectionStatusCodes)
6.  }
```

```
1.   //otevreni pripojeni
2.      func connect(host: String, port: UInt32) {
3.          var rxStream: Unmanaged<CFReadStream>?
4.          var txStream: Unmanaged<CFWriteStream>?
5.
6.          CFStreamCreatePairWithSocketToHost(nil, host as CFString, port, &rxStream, &txStream)
7.
8.          self.inStream = rxStream?.takeRetainedValue()
9.          self.outStream = txStream?.takeRetainedValue()
10.
11.         if let rxStream = inStream, let txStream = outStream {
12.             rxStream.delegate = self
13.             txStream.delegate = self
14.             rxStream.schedule(in: RunLoop.main, forMode: RunLoop.Mode.default)
15.             txStream.schedule(in: RunLoop.main, forMode: RunLoop.Mode.default)
16.             rxStream.open()
17.             txStream.open()
18.         }
19.     }
```

```
1.      func send(data: String) {
2.          var dx: String
3.          if (data.hasSuffix("\n")){
4.              dx = data
5.          }else{
6.              dx = data + "\n"
7.          }
8.          //kontrola jestli lze zapsat data na vystupni Stream
9.          //jinak dojde k zapsani do poradniku
10.         if(outHasSpace){
11.             let data = dx.data(using: .windowsCP1250, allowLossyConversion: true)!
12.             if (data.count>0) {
13.                 outHasSpace = false
14.                 //odeslani dat
15.                 _ = data.withUnsafeBytes { self.outStream?.write($0, maxLength: data.count) }
16.             }
17.         }else{
18.             Queue.append(data)
19.         }
20.     }
```

```
1.      func disconnectStatus(status: connectionStatusCodes) {
2.          if let stream = self.inStream {
3.              stream.close()
4.              stream.remove(from: RunLoop.current, forMode: RunLoop.Mode.default)
5.          }
6.          if let stream = self.outStream {
7.              stream.close()
8.              stream.remove(from: RunLoop.current, forMode: RunLoop.Mode.default)
9.          }
10.         inStream = nil
11.         outStream = nil
12.         outHasSpace = false
13.         delegate?.disconnectStatus(status)
14.     }
```

```
1.  func stream(_ aStream: Stream, handle eventCode: Stream.Event){
2.          switch eventCode {
3.          case Stream.Event.errorOccurred:
4.              disconnectStatus(status: connectionStatusCodes.CONNECTION_STATUS_DISCONNECTED_WIT
    H_ERROR)
5.              break
6.          case Stream.Event.endEncountered:
7.              disconnectStatus(status: connectionStatusCodes.CONNECTION_STATUS_DISCONNECTED)
8.              break
9.          case Stream.Event.hasBytesAvailable:
10.             //dostupna data na vstupu, pripravi se buffer do ktereho jsou data nacteny
11.             //nacitani dat probiha dokud plati tento event, prijata data jsou ukladana
12.             //dokud probiha cteni, nasledne jsou delegovana
13.             if aStream == inStream {
14.                 let maxLenght = 4096
15.                 let buffer = UnsafeMutablePointer<UInt8>.allocate(capacity: maxLenght)
16.                 var Received = ""
17.                 while inStream!.hasBytesAvailable {
18.                     let bytes = inStream!.read(buffer, maxLength: maxLenght)
19.                     if bytes<0 {
20.                         if let _ = inStream!.streamError{
21.                             break
22.                         }
23.                     }
24.                     Received.append(String(bytesNoCopy: buffer, length: bytes, encoding: .win
    dowsCP1250, freeWhenDone: false)!)
25.                 }
26.                 delegate?.dataReceived(Received)
27.             }
28.             break
29.         case Stream.Event.openCompleted:
30.             //informovani pri uspesnem navazani spojeni
31.             if aStream == inStream {
32.                 inStatus = aStream.streamStatus
33.                 if outStatus == Stream.Status.open {
34.                     delegate?.streamsOpened()
35.                 }
36.             }else if aStream == outStream {
37.                 outStatus = aStream.streamStatus
38.                 if inStatus == Stream.Status.open {
39.                     delegate?.streamsOpened()
40.                 }
41.             }
42.             break
43.         case Stream.Event.hasSpaceAvailable:
44.             //vystup pripraven k zapisu. Odeslani prvnich cekajicich dat
45.             if aStream == outStream {
46.                 if Queue.count>0 {
47.                     outHasSpace = true
48.                     send(data: Queue.removeFirst())
49.                 } else {
50.                     outHasSpace = true
51.                 }
52.             }
53.             break
54.         default:
55.             break
56.         }
57.     }
58. }
```

Příloha k diplomové práci

OVLÁDÁNÍ INTELIGENTNÍHO DOMU POMOCÍ CHYTRÝCH HODINEK

Michal Fišar

Uživatelský manuál

Příloha C

Obsah

1 Ovládání GUI Místností

Pro přecházením mezi jednotlivými místnostmi slouží gesto táhnutí prstem směrem vlevo, respektive vpravo. Pokud je skryta levá část vedle tlačítka (na obr. vlevo), gesto směrem vpravo nefunguje. Naopak pokud je skryta pravá část vedle tlačítka, nefunguje gesto směrem vlevo.

Pro zvolení místnosti klikněte na tlačítko s názvem vámi vybrané místnosti uprostřed.

2 Ovládání GUI Místnosti

Procházení kategorií objektů v místnosti funguje stejně jako výběr místnosti. Gesta jsou ale navíc zablokována po vybrání kategorie a dochází k zobrazení animace v pravém horním rohu. Aplikace čeká na příchozí data z mobilního zařízení.

3 Ovládání scén

V kategorii scén je zobrazena tabulka s výpisem všech scén. Jednoduchým stisknutím řádku se scénou dojde k zobrazení animace v pravém horním rohu. Současně s tím je zablokován výběr další scény. Po obdržení dat z mobilního zařízení dojde k zobrazení GUI scény.

GUI scény zobrazuje výpis dostupných nastavení pro právě prohlíženou scénu. Jednoduchým kliknutím dojde k nastavení vybrané scény. Dochází ke zobrazení ✓ v pravé části zvoleného řádku. Symbol po krátké době zmizí.

4 Ovládání světel

V kategorii světel je zobrazena tabulka se všemi světly. Každý řádek obsahuje v pravé části informaci o aktuálním stavu daného světla. Světlo svítí, pokud je vyobrazen zelený symbol. Uprostřed řádku je vymezeno aktuální nastavení stmívače, které může být navíc zbarveno různou barvou, na základě aktuální barvy světla.

Jednoduchým stisknutím řádku dochází ke změně stavu světla. Při dlouhém stisku se pak otevře GUI s podrobným nastavením vybraného světla.

Na základě vlastností světla může, ale nemusí, GUI obsahovat všechny prvky, uvedené na obrázku. Jeli světlo stmívatelné, je možné nastavit jeho úroveň pomocí + nebo – . Nebo je možné využít otáčení tlačítka Crown. Nastavování probíhá s krokem 5 % a k nastavování dochází okamžitě po změně. Pokud je možné nastavit světlu barvu, bude zobrazen tzv. picker, který obsahuje několik předem navolených barev. Po vybrání barvy je nutné použít tlačítko nastavit.

Poslední možností je opět změna stavu světla. K její změně také dochází okamžitě po stisku.

5 Ovládání žaluzií

Kategorie žaluzií obsahuje také tabulku s výčtem všech objektů. V pravé části řádku se může nacházet symbol šipky nahoru, nebo dolu. Ten značí pohyb dané žaluzie tímto směrem. V případě, že je žaluzie v jednom z koncových bodů, je šipka zbarvena do žluté barvy. Jednoduchým stisknutím řádku dojde k otevření ovládání vybrané žaluzie.

Čtyři velké šipky v rozích slouží k ovládání. V případě stisku, a tedy aktivaci pohybu, dochází k jejich zabarvení na červeno. Spodní šipky jsou určeny k otáčení lamel o jeden krok.

Menší šipka uprostřed slouží k informování o aktuálním stavu žaluzií. V případě pohybu je červená a směřuje vždy po směru. Pokud žaluzie dosáhne jednoho z konečných stavů, dojde k jejímu zbarvení na žlutou barvu a její směr zůstává stejný.

Pokud se žaluzie nehýbou a současně nejsou ani v jednom koncovém stavu, malá šipka uprostřed není zobrazena.

6 Ovládání zásuvek a relé

Kategorie zásuvek a relé jsou v současné době stejné. Obsahují tabulku s výpisem všech objektů. Pro změnu stavu objektu slouží jednoduché kliknutí na vybraný řádek. V budoucnu se může ovládání kategorie relé změnit.

7 Ovládání displejů

Kategorie displejů v současné době slouží pouze k zobrazování informací o teplotě a vlhkosti. Princip ovládání je totožný s místnostmi, tedy pomocí gest. Na rozdíl od místností se ale nedá na žádný ze zobrazených displejů kliknout.