



System SZARP - dokumentacja techniczna

SZARP v.3.1

Michał Blajerski

Sławomir Chyłek

Marcin Goliszewski

Jarosław Janik

Paweł Kolega

Daniel Majewski

Dariusz Marcinkiewicz

Paweł Pałucha

Adam Smyk

Niniejszy dokument opisuje różne zagadnienia związane z systemem SZARP, nieinteresujące z punktu widzenia zwykłego użytkownika. Jest przeznaczony dla osób rozwijających aplikacje czy też konfigurujących system.

\$Id\$

1. Instalacja systemu SZARP

Instalacja systemu SZARP wymaga instalacji i przygotowania systemu Linux. Wspierana dystrybucja to Debian GNU/Linux w wersji najnowszej stabilnej (Lenny) lub niestabilnej (Sid). Zainstalowany powinien być podstawowy system wraz ze środowiskiem graficznym KDE lub GNOME (ew. inne wspierające standard Freedesktop). Do prawidłowego działania części programów zwłaszcza konsolowych wymagane jest ustawienie kodowania w systemie (locales) na pl_PL.ISO-8859-2 (a nie pl_PL.UTF-8). Prawdopodobnie bez większych problemów SZARP powinien działać także na Ubuntu, choć nie jest to regularnie testowane. Przystosowanie SZARP'a do działania pod inną dystrybucją jest oczywiście możliwe, ale może wymagać sporo pracy.

Bezpośrednia kompilacja oprogramowania ze źródeł jest oczywiście możliwa, ale podstawową metodą instalacji jest instalacja z pakietów Debiana. W związku z tym standardowa instalacja ze źródeł może wymagać ręcznych poprawek (zakładanie użytkowników, ustawianie uprawnień), które normalnie wykonywane są podczas instalacji pakietów.

Repozytorium SVN ze źródłami dostępne jest na stronie SourceForge projektu - sourceforge.net/projects/szarp (<http://sourceforge.net/projects/szarp>). Programy klienckie dostępne są też w wersji dla Windows - ze strony powyżej można ściągnąć instalator.

1.1. Wymagane biblioteki i programy

W chwili obecnej zalecaną metodą instalacji systemu SZARP jest użycie pakietów binarnych, zamiast bezpośredniej kompilacji ze źródeł. Zwalnia to użytkownika z konieczności ręcznej instalacji wymaganych bibliotek.

Większość wymaganych przez system bibliotek jest standardowo obecna we współczesnych dystrybucjach Linuksa. Spis większości wymaganych do kompilacji narzędzi i bibliotek znajduje się w pliku `debian/control` w polu *Build-Depends*. Lista wymaganych bibliotek i programów jest wypisywana przez skrypt `./configure` w głównym katalogu źródeł - skrypt informuje o tym czego mu brakuje. Do tworzenia dokumentacji potrzebne są narzędzia `jade` (lub `openjade`) wraz z zainstalowanym `docbookiem`, `pdfjadetex` i `ImageMagic` (`convert`).

1.2. Kompilacja i instalacja systemu

Sposób kompilacji i instalacji systemu SZARP jest dokładnie opisany w pliku `INSTALL` znajdującym się w głównym katalogu ze źródłami systemu. W większości przypadków sprowadza się to do wydania w katalogu ze źródłami komend:

```
./autogen.sh
./configure
make
make doc
su
make install
```

1.3. Instalacja pakietów binarnych

1.3.1. Przygotowanie systemu

System SZARP dostępny jest także w postaci pakietów deb. Instalację należy rozpocząć od dodania wpisów do pliku `/etc/apt/sources.list`:

```
deb http://www.szarp.org/debian stable main non-free
deb http://www.szarp.org/debian unstable main non-free
```

Oczywiście, możemy dodać tylko jedną z w/w linii jeżeli zamierzamy korzystać tylko i wyłącznie z jednej wersji SZARPa. Zalecane jest jednak, by plik `sources.list` zawierał wszystkie wymienione wpisy, umożliwi to potem zmianę używanej wersji SZARPa za pomocą skryptów konfiguracyjnych paczki *szarp-updater*. Kolejność wpisów ma znaczenie, *apt-get* stara się instalować pakiety ze źródła, które występuje jako pierwsze, zachowanie to można zmienić opcją `-t` programu *apt-get* (np. *apt-get -t stable*)

Ściągamy indeksy repozytorium komendą:

```
apt-get update
```

Za pomocą komendy:

```
apt-cache search szarp
```

możemy przejrzeć listę dostępnych pakietów związanych z SZARP'em, szczegóły dotyczące konkretnego pakietu możemy obejrzeć wywołując komendę np.

```
apt-cache show szarp-server
```

1.3.2. Wybór roli

Podczas instalacji musimy zdecydować jaką rolę ma pełnić komputer, na którym uruchamiamy system SZARP: serwera, terminala lub też służyć będzie jedynie do przeglądania baz. Dla każdej z tych ról przewidziana jest oddzielna paczka, są to odpowiednio: *szarp-server*, *szarp-terminal* oraz *szarp-viewer*.

Instalujemy wybraną paczkę np. następującą komendą:

```
apt-get install szarp-terminal
```

W trakcie instalacji należy odpowiedzieć na kilka pytań, na podstawie udzielonych odpowiedzi zostanie stworzona konfiguracja systemu, niektóre pytania są specyficzne dla pewnych pakietów.

- *szarp-server*
 - Skrypt poprosi o podanie prefiksu konfiguracji, wpisujemy tutaj odpowiednią nazwę. Należy zaznaczyć, że skrypt startowy (parstart) wymaga by nazwa serwera, zwracana przez komendę **hostname -s**, była identyczna z prefiksem konfiguracji, w przeciwnym wypadku demony właściwe dla serwera nie zostaną uruchomione.

- Możemy także zostać zapytani, czy chcemy by został utworzony katalog szbase w katalogu `/opt/szarp/prefix`, gdzie `prefix` jest podaną nazwą prefiksu. Zwykle należy odpowiedzieć twierdząco na to pytanie.
- *szarp-terminal*

Proces konfiguracji paczki przebiega podobnie jak pakietu *szarp-server*, pojawiają się jedynie dwa dodatkowe pytania:

 - Czy chcemy by adres serwera został dodany do pliku `/etc/hosts`. W przypadku typowych instalacji odpowiedź na to pytanie powinna być twierdząca.
 - Jeżeli na poprzednie pytanie odpowiedzieliśmy twierdząco, zostaniemy poproszeni o wprowadzenie adresu serwera
- *szarp-viewer*

Podczas instalacji paczki zostaniemy poproszeni o wybranie sposobu aktualizacji danych z serwera.

1.3.3. Aktualizacja pakietów

Pakiet *szarp-updater* służy do aktualizacji systemu, podczas konfiguracji pakietu należy zdecydować jakiego wydania systemu SZARPa chcemy używać. Do wyboru są trzy standardowe: *stabilne*, *testowe* oraz *niestabilne*. Ponadto można wybrać opcję *inne*, wtedy samodzielnie musimy określić nazwę wydania. Próba aktualizacji odbywa się raz na dobę, jeżeli pojawi się nowa wersja systemu zostanie ona zainstalowana.

Wersja stabilna jest zalecana do użycia w środowisku produkcyjnym, zawiera poprawki wszystkich krytycznych błędów oraz w miarę regularnie dodawane nowe funkcje - po ich przetestowaniu. Wersja niestabilna jest często budowaną wersją deweloperską, zawierającą najnowsze funkcje. Zalecana jest do użycia w mniej krytycznych zastosowaniach - czyli np. tam gdzie realizowany jest tylko podgląd danych.

1.3.4. Opis pakietów

Poniżej zostały opisane główne pakiety składające się na system SZARP. Szczegółowy, najbardziej aktualny spis zawartości poszczególnych pakietów znajduje się w pliku `debian/control` w katalogu źródeł systemu.

- *szarp-base* - na pakiet ten składają się podstawowe komponenty systemu, takie jak: schematy DTD, skrypty do nawiązywanie połączeń ppp, aktualizacji pakietów deb, *szrsync*, skrypty xslt.
- *szarp-daemons* - zawiera demony systemu SZARP: *parcook*, *meaner3*, *analiza*, *netpard* oraz demony linii.
- *szarp-utils* - w tym pakiecie znajdują się narzędzia, służące do konwersji konfiguracji między formatami IPK oraz SZARP 2.1, edycji baz danych szbase, skrypty do odpytywania węzłów połączonych za pomocą modemów analogowych oraz ISDN, programy *sendhex* oraz *sendhex-wrapper*.

- *szarp-scripts* na pakiet składają się różne pomocnicze skrypty.
- *szarp-wx* to zbiór aplikacji klienckich SZARP wykorzystujących bibliotekę wxWidgets. W jego skład wchodzi programy *draw3*, *ipkedit*, *ekstraktor3*, *kontroler3*, *raporter3*, *wxhelp*, *SCC* oraz *synchronizator*, *filler*.
- *szarp-paramd* w pakiecie umieszczono serwer *paramd* udostępniający wartości parametrów za pomocą protokołu HTTP.
- *szarp-xsltd* zawiera pliki konfiguracyjne, konieczne do uruchomienia serwera dla programów *raporter3* i *kontroler*.
- *szarp-sss* zawiera skrypty dla pomocniczego serwera (mirrora) baz danych SZARP.
- *szarp-sssweb* zawiera interfejs WWW do administracji użytkownikami korzystającymi z synchronizatora danych SZARP.
- *szarp-doc-html* oraz *szarp-doc-pdf* to pakiety zawierające dokumentację systemu SZARP w formatach html oraz pdf.
- *szarp-doc-server* to dokumentacja systemu SZARP przeznaczona do instalacji na serwer WWW.
- *szarp-server*, *szarp-terminal*, *szarp-viewer* paczki służą do konfiguracji systemu do pracy w określonej roli, szczegółowo ich użycie zostało opisane w rozdziale Wybór roli Sekcja 1.3.2.
- *szarp-updater* to paczka do automatycznej aktualizacji wersji systemu SZARP.

1.4. Struktura katalogów

Całość systemu SZARP znajduje się domyślnie w katalogu `/opt/szarp`.

W podkatalogu `/opt/szarp/bin` znajdują się programy użytkowe systemu.

W podkatalogu `/opt/szarp/resources` znajdują się różnego rodzaju dane wykorzystywane przez system. W szczególności w podkatalogu `/opt/szarp/resources/documentation` znajduje się dokumentacja systemu.

W podkatalogu `/opt/szarp/lib` znajdują się biblioteki wykorzystywane przez programy SZARP, między innymi moduły Pythona.

Każda *konfiguracja* (czyli zestaw plików konfiguracyjnych a także bazy z danymi historycznymi systemu) znajduje się w katalogu `/opt/szarp/<prefix>`, gdzie *prefix* jest nazwą systemu, taką samą jak nazwa serwera SZARP obsługującego daną konfigurację. Tradycyjnie jest to czteroliterowy skrót nazwy miast, np. *zamo* dla Zamościa, *gcwp* dla ciepłowni WP w Gliwicach itp.

Katalogów z konfiguracjami może być na jednym komputerze wiele, gdyż system umożliwia przeglądanie danych z wielu konfiguracji. Przykładowo instalacja w Gliwicach składa się z trzech serwerów o nazwach *gliw*, *gcie* i *gcwp*. Na komputerze *gliw-t1* (terminal systemu SZARP) będą się znajdować konfiguracje ze wszystkich trzech serwerów, a więc katalogi `/opt/szarp/gliw`, `/opt/szarp/gcie` i `/opt/szarp/gcwp`.

W katalogu `/opt/szarp/<prefix>` mogą znajdować się następujące podkatalogi:

- `config` - katalog z konfiguracją dla danego systemu. Zwykle przy zmianach konfiguracji tworzy się nowy katalog o nazwie zawierającej informację o dacie modyfikacji konfiguracji, np. `config20040317` (konfiguracja z 17 marca 2004). Wtedy sam katalog `config` jest linkiem

symbolicznym na odpowiedni katalog. W nowszych instalacjach używany jest tylko katalog `config`, ale jest on kopią roboczą repozytorium SVN zawierającego całą historię konfiguracji.

- `szbase` - zawiera bazę danych historycznych w formacie `SzarpBase`.

O ile podkatalog `config` zawiera opis konfiguracji sterowników, parametrów, raportów, wykresów itp. związany z jakąś instalacją fizyczną systemu (zobacz Sekcja 5), to opcje konfiguracji programów wchodzących w skład systemu SZARP znajdują się w pliku `/etc/szarp/szarp.cfg` w formacie opisanym w Sekcja 4.1. W instalacjach systemu "na obiektach" (w przeciwieństwie do instalacji np. w domach użytkowników oglądających dane z kilku systemów) zwykle plik `szarp.cfg` jest wspólny dla wszystkich serwerów i terminali na obiekcie. Plik umieszcza się wtedy w katalogu `/opt/szarp/<prefix>` jednego z serwerów i na wszystkich komputerach na obiekcie tworzy się linki symboliczne `/etc/szarp/szarp.cfg` wskazujące na ten plik. Dodatkową zaletą takiego rozwiązania jest to, że dzięki mechanizmowi BODAS zmiany konfiguracji są przegrywane automatycznie.

Podobną konwencję linku od odpowiedniego katalogu `/opt/szarp/<prefix>` stosuje się w przypadku pliku `/etc/szarp/parstart.cfg`, opisującego składniki systemu SZARP jakie mają być uruchomione przy starcie systemu. Plik ten jest wykorzystywany przez skrypt startowy `/etc/init.d/parstart`.

1.5. Konfiguracja w środowisku KDE albo window managerze

Zalecanym obecnie sposobem jest, zamiast dodawania wpisów do menu menedżera okien, użycie programu SCC (Szarp Control Center). Szczegóły są opisane w dokumentacji programu SCC (<http://www.szarp.org/szarp/doc/scc/html/scc.html>).

1.6. Na skróty, czyli o czym pamiętać przy instalacji serwera/terminala systemu SZARP

Poniżej umieszczona lista obejmuje czynności, które trzeba wykonać podczas instalacji serwera/terminala SZARP. Należy zatem:

- Ustawić w BIOS'ie automatyczne wstawianie po zaniku napięcia.
- Zainstalować Debiana, w razie potrzeby zaktualizować do najnowszej wersji.
- Zainstalować system SZARP z paczkami odpowiednimi dla serwera lub terminala. Nie zapomnieć, że instalacja `szarp-server` nie powoduje automatycznej instalacji `szarp-wx`.
- Ustawić możliwość zdalnego logowania tylko na root'a, aby nie narażać systemu przez słabe hasła użytkowników - w `/etc/ssh/sshd_config`:

```
AllowUsers root
```
- Odblokować możliwość logowania się bez hasła w `/etc/pam.d/common-auth`.
- Zdalne logowanie - włączyć XDMCP, pozwolić na dostęp w Xaccess, włączyć nasłuchiwanie na TCP w serwerze fontów.
- Sprawdzić w KDE polskie czcionki.
- Skonfigurować Menedżera logowania ustawiając:

- tekst powitania,
- logo,
- obrazki użytkowników,
- automatyczne logowanie.
- Skonfigurować interfejsy sieciowe.
- Skonfigurować NTP (Network Time Protocol).
- Skonfigurować w razie potrzeby approx.
- Skonfigurować szarp-updater.
- W razie potrzeby skonfigurować dostęp (OpenVPN).
- Wymienić klucze.

2. Konfiguracja SZARP 2.1

Notatka: Obecnie praktycznie wszystkie programy nie korzystają z opisanych w tym rozdziale plików, ale z pojedynczego pliku XML, opisanego w rozdziale Sekcja 5. Ten rozdział zostawiono dla celów archiwalnych, pomaga on też zrozumieć strukturę konfiguracji systemu.

Rozdział zawiera opis formatu plików konfiguracyjnych SZARP w wersji 2.1. Wszystkie opisywane pliki konfiguracyjne są zwykłymi plikami tekstowymi, powinny zawierać znaki końca linii w formacie uniksowym. Na długość części pól nałożone są ograniczenia, wynikające z konstrukcji wczytujących je programów. Przekroczenie tych ograniczeń może spowodować zgłoszenie błędu przez program, obcięcie wczytywanych wartości do maksymalnej długości lub błąd programu. Jako separatory pól powinny być stosowane spacje (najlepiej pojedyncze), chyba że coś innego wynika bezpośrednio z innych wymagań. Trudno określić zachowanie programów w przypadku błędnego formatu pliku. Większość (zwłaszcza starszych) programów stara się interpretować nawet błędne pliki, co może być źródłem różnego rodzaju błędów.

2.1. Plik `parcook.cfg`

Plik `parcook.cfg` zawiera informację o konfiguracji demonów linii odpowiedzialnych za komunikację ze sterownikami, oraz opis parametrów definiowalnych, wyliczanych przez program `parcook`. Plik jest wymagany przez każdą instalację SZARP w wersji 2.1, czytany jest przez program `parcook`. Wykorzystywany może być także do tworzenia identyfikatorów dla komunikacji międzyprocesowej między aplikacjami systemu.

Format pliku jest następujący:

```
[demony] [okres] [ekstra]
[numer] [ilość parametrów] [demon] [port komunikacyjny] ...
...
[formuły]
[formuła]
...
```

Opis poszczególnych pól:

- *[demony]* - liczba określająca ilość linii z opisami demonów komunikacyjnych (linii komunikacyjnych), od 1 do 64.
- *[okres]* - długość w sekundach okresu co jaki odpytywane będą demony linii, nigdy nie używana była liczba inna niż 10, skutki wpisania innej wartości pozostają nieznanne.
- *[ekstra]* - ilość zdefiniowanych w pliku parametrów definiowalnych, może być 0.
- *[numer]* - unikalny numer linii komunikacyjnej, z zakresu od 1 do 64. Zwykle numery nadawane są kolejno. Na każdą linię komunikacyjną przypada 1 linia w pliku.
- *[ilość parametrów]* - ilość parametrów odczytywana ze sterownika, odpowiada wielkości segmentu pamięci dzielonej tworzonej dla demona linii, może być równa 0 lub większa. Liczba 0 oznacza sterownik pusty.
- *[demon]* - ścieżka do demona linii - programu odpowiadającego za komunikację w obrębie jednej linii komunikacyjnej (czyli fizycznie jednego kabla podpiętego do komputera). W starych konfiguracjach bywała pusta - parcook przyjmował domyślną wartość /opt/szarp/bin/linedmn. Zobacz też komentarze do następnego pola.
- *[port komunikacyjny]* - ścieżka do portu komunikacyjnego używanego przez demona linii. W rzeczywistości parcook wywołuje demona linii przekazując mu jako pierwszy parametr identyfikator (numer linii) - czyli liczbę z początku linii, a następnie wszystko co napotka po nazwie demona. Tradycyjnie drugim argumentem do demona linii jest właśnie ścieżka do portu. Kolejne argumenty zależą od konkretnego demona, dla najpopularniejszego - rsdmn mogą to być także prędkość komunikacji (w bodach), ilość bitów stopu, wersja protokołu (liczba). W niektórych starych komunikacjach, gdy nie była podawana ścieżka do demona, parcook tworzył także ścieżkę do portu komunikacyjnego karty Specialix na podstawie numeru linii (numer linii odpowiadał numerowi portu).
- *[formuły]* - ilość linii z formułami opisującymi parametry definiowane. Liczba ta nie musi być tożsama z ilością zdefiniowanych parametrów (pole *[ekstra]*), gdyż dopuszczalne jest wielokrotne przedefiniowanie parametrów. Liczba linii może być oczywiście równa 0. Następnie powinna wystąpić podana liczba linii z formułami.
- *[formuła]* - formuła programu parcook opisująca parametr. Postać formuły jest następująca:

```
formuła := wyrażenie '#' komentarz
wyrażenie := element ' ' | wyrażenie wyrażenie | 'null'
element := (stała|adres_parametru|kod_operacji) ' '
stała := #cyfra[cyfra]
adres_parametru := cyfra[cyfra]
cyfra := '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'
spacja := ' '
kod_operacji := '&'|'!'|'$'|'#'|'+'|'-'|'*'|'/'
komentarz := nie# [nie#]
nie# := !'#' (dowolny znak nie będący '#')
```

Jak wynika z powyższego elementy formuły są oddzielone spacjami. Formuła zapisana jest w Odwrotnej Notacji Polskiej (ONP, RPN). Liczba oznacza wstawienie na stos wartości parametru o określonym liczbą indeksie IPC. Indeksy IPC parametrów to indeksy w tablicy parametrów tworzonej przez program parcook po kolei na podstawie danych o demonach linii i ilości parametrów definiowalnych (liczone od 0). Stała, oznaczana przez liczbę poprzedzoną znakiem # powoduje wstawienie na stos wartości stałej. Napis *null* oznacza formułę pustą. Formułę kończy także znak #, po którym następuje nieobowiązkowy komentarz. Znaczenie kodów operacji jest następujące:

- + - pobierz dwa parametry z wierzchołka stosu i odłóż na stos ich sumę
- - - jak wyżej, tylko różnica
- * - jak wyżej, tylko iloczyn
- / - jak wyżej, tylko iloraz
- & - zamień miejscami dwa parametry z wierzchołka stosu (swap)
- ! - kopiuje wartość z wierzchołka stosu (duplicate)
- \$ - wywołaj funkcję o identyfikatorze i parametrach pobranych kolejno z wierzchołka stosu
- = - pobierz adres IPC z wierzchołka stosu i wstaw pod niego kolejną wartość pobraną z wierzchołka stosu

Pojawienie się w wyniku obliczeń błędu (np. dzielenie przez 0) spowoduje że wstawiane będą pod parametry wartości NO_DATA (brak danych).

Przykładowa zawartość pliku:

```
2 10 8
1 32 /opt/szarp/bin/rsdmn /dev/ttyX0
2 58 /opt/szarp/bin/rsdmn /dev/ttyX1
8
31 #0 #2 #7 $ #90 = # wel (Kosmonautów:Węzeł:praca automatyczna w lecie) -
31 #1 #2 #7 $ #91 = # wel (Kosmonautów:Węzeł:praca automatyczna w zimie) -
31 #2 #2 #7 $ #1000 * #92 = # wel (Kosmonautów:Węzeł:zawór mieszania otwarty) Zawór u
31 #3 #2 #7 $ #1000 * #93 = # wel (Kosmonautów:Węzeł:zawór mieszania zamknięty) Zawór
89 #0 #2 #7 $ #94 = # wel (Kosmonautów:Węzeł:praca automatyczna w lecie) -
89 #1 #2 #7 $ #95 = # wel (Kosmonautów:Węzeł:praca automatyczna w zimie) -
89 #2 #2 #7 $ #1000 * #96 = # wel (Kosmonautów:Węzeł:zawór mieszania otwarty) Z
89 #3 #2 #7 $ #1000 * #97 = # wel (Kosmonautów:Węzeł:zawór mieszania zamknięty)
```

2.2. Pliki lineX.cfg

Pliki o nazwach *lineX.cfg*, gdzie zamiast *X* występuje numer (identyfikator) linii - taki jak w pliku *parcook.cfg* - opisują konfigurację demona linii. Są wykorzystywane przed demonami linii oraz do tworzenia identyfikatorów mechanizmów IPC do komunikacji między programem *parcook* a demonami linii.

Plik ma postać:

```
[jednostki]
[kod] [typ] [podtyp] [parametry] [nastawy] [bufor]
...
```

Pierwsza liczba zawiera liczbę jednostek komunikacyjnych wewnątrz linii. Najczęściej (w przypadku RS-232) obecna jest jedna linia komunikacyjna. W przypadku RS-485 linii może być więcej. Specjalna wartość 0 oznacza specjalny okrojony protokół RS-232, bez adresowania jednostki. Następnie występują linie, po jednej dla jednostki, z opisem parametrów dla jednostki. Znaczenia poszczególnych pól są następujące:

- *kod* - znak ASCII będący identyfikatorem linii
- *typ* - liczba oznaczająca typ raportu (parametr dla sterownika, z zakresu od 0 do 255)
- *podtyp* - liczba oznaczająca podtyp raportu (parametr dla sterownika, z zakresu od 0 do 255)
- *parametry* - ilość parametrów odczytywanych z jednostki
- *nastawy* - ilość parametrów wysyłanych do jednostki (przez program sender)
- *bufor* - wielkość bufora uśrednień dla demona linii

Typowa zawartość pliku `lineX.cfg`:

```
1
1 1 1 32 0
```

Nieco inaczej wygląda składnia pliku w przypadku użycia radiomodemów. W takiej sytuacji pierwsza linia zaczyna się od znaku *R*, po którym występuje natychmiast (bez separatorów) ilość linii radiowych obsługiwanych przez demona. W następnych liniach dla każdej linii radiowej znajduje się sekcja składająca się z linii z tekstowym identyfikatorem modemu oraz opisu konfiguracji dla linii zgodnie ze zwykłym formatem pliku. Przykładowy konfiguracja dla komunikacji przez radiomodem z dwoma identycznymi sterownikami:

```
R2
Toszek
1
1 1 0 21 1 3
Jeden
1
1 1 0 21 1 3
```

Oba sterowniki dostarczają w sumie 42 parametry. Ciekawostką w powyższym przykładzie (autentycznym! - konfiguracja z Pyskowic) jest fakt, że obie linie posiadają ten sam identyfikator. Do obu ma być wysyłany jeden parametr. Ponieważ jednak program sender używa tylko identyfikatorów linii (nie korzysta z identyfikatorów modemów), więc prawdopodobnie ewentualne nastawy parametrów wysyłane mogłyby być tylko do pierwszego z wymienionych sterowników (bo program najpierw jego by zidentyfikował jako posiadającego odpowiedni identyfikator).

2.3. Plik PTT.act

Plik `PTT.act` opisuje mapowanie parametrów na indeksy w bazie w formacie CodeBase, a także zawiera szereg informacji o parametrach, takich jak pełna nazwa, skrót, jednostka, czy precyzja reprezentacji danych. Plik jest wykorzystywany przez dużą część programów SZARP 2.1, w

szczegółności meaner, draw, dysp. Numer linii z opisem parametru w pliku jest często wykorzystywany przez starsze programy jako identyfikator parametru (zwykle liczony od 1, opis parametru pierwszego parametru znajduje się w linii 2). Jest on bezpośrednio przeliczalny na indeks w bazie (numer rekordu i numer pola) według zależności:

```
numer_rekordu = (indeks_PTT - 1) / 15 + 1
numer_pola = (indeks_PTT - 1) % 15
```

Pierwsza linia w pliku `PTT.act` zawiera 3 liczby, oddzielone spacjami. Pierwsza to teoretycznie numer wersji formatu pliku, w praktyce zawsze jest to 1. Druga liczba to ilość parametrów zapisywanych do bazy. Trzecia (nie mniejsza od drugiej) to ilość opisów parametrów w pliku (równa liczbie linii pliku minus 1). Opisów parametrów może być więcej niż parametrów zapisywanych do bazy, bo nie wszystkie parametry muszą być zapisywane do bazy. Każda linia z opisem parametru ma następujący format:

```
[indeks IPC] [precyzja] [skrót nazwy] [pełna nazwa] [jednostka];[nazwa wykresu]#koment
```

Znaczenie poszczególnych pól jest następujące:

- *indeks IPC* - indeks IPC parametru (od 0). Wartości większe od 65536 oznaczają parametr pusty - obecny w bazie ale niezberyany ze sterownika.
- *precyzja* - liczba od 0 do 7. Wartości od 0 do 4 oznaczają ilość miejsc po przecinku, z jaką jest reprezentowany parametr w bazie. Przykładowo, jeśli w bazie zapisana jest wartość 125, a precyzja wynosi 1, to rzeczywista wartość jaką otrzymamy to 12.5. Wartość 5 oznacza, że parametr może przyjmować wartości 0 (interpretowana jako "nie") lub 1 (interpretowana jako "tak"). Wartość 6 oznacza że możliwe wartości to 0 ("Pochmurno"), 1 ("Zmiennie"), 2 ("Słonecznie"). Wartość 7 oznacza możliwe wartości -1 ("Minus") i 1 ("Plus").
- *skrót nazwy* - skrótowa nazwa parametru - do 4 liter.
- *pełna nazwa* - pełna nazwa parametru, składa się z 3 pól oddzielonych dwukropkami.
- *jednostka* - nazwa jednostki dla wartości w nawiasach kwadratowych.
- *nazwa wykresu* - nazwa wykresu dla parametru w programie przeglądającym

Opisy parametrów niezapisywanych do bazy mogą być pozbawione części od średnika włącznie.

Przykładowy fragment zawartości pliku:

```
1 32 36
0 1 Tcie Armii Ludowej:Węzeł:temperatura wody z ciepłowni [°C];Temp. wej. w węźle# (1,
1 1 Tpow Armii Ludowej:Węzeł:temperatura wody powrotnej [°C];Temp. powrotna# (1, 1)
2 2 Gw Armii Ludowej:Węzeł:przepływ węzła [t/h];Przepływ węzła# (1, 2)
3 3 Qw Armii Ludowej:Węzeł:moc węzła [MW];Wydajność węzła# (1, 3)
4 1 Tzew Armii Ludowej:Węzeł:temperatura zewnętrzna [°C];Temp. zewnętrzna# (1, 4)
5 1 Tcox Armii Ludowej:Węzeł:temperatura CO w funkcji zewnętrznej [°C];Temp. zad. CO o
6 1 Tcok Armii Ludowej:Węzeł:zadana temperatura CO [°C];Temp. zadana CO# (1, 6)7 1 Tc
8 1 Tpn Armii Ludowej:Węzeł:temperatura powrotna z CO [°C];Temp. powrotu z CO# (1, 8)
9 3 DPx Armii Ludowej:Węzeł:zadane ciśnienie dyspozycyjne [MPa];Zadana dyspozycja# (1,
10 3 DP Armii Ludowej:Węzeł:ciśnienie dyspozycyjne [MPa];Ciśnienie dyspoz.# (1, 10)
11 1 Polz Armii Ludowej:Węzeł:położenie zaworu upustowego [%];Położenie upustu# (1, 1
12 1 Tre1 Armii Ludowej:Węzeł:temperatura rezerwowa 1 [°C];Temp. rezerwowa 1# (1, 12)
13 1 Tre2 Armii Ludowej:Węzeł:temperatura rezerwowa 2 [°C];Temp. rezerwowa 2# (1, 13)
```

```

15 1 Kenl Armii Ludowej:Węzeł:energia z licznika lsw [GJ];Energia lsw# (1, 14)
16 1 Kenm Armii Ludowej:Węzeł:energia z licznika msw [GJ];Energia msw# (2, 0)
17 0 Kwol Armii Ludowej:Węzeł:woda z licznika lsw [m3];Woda lsw# (2, 1)
18 0 Kwom Armii Ludowej:Węzeł:woda z licznika msw [m3];Woda msw# (2, 2)
19 0 Khl Armii Ludowej:Węzeł:czas pracy z licznika lsw [h];Czas pracy lsw# (2, 3)
20 0 Khm Armii Ludowej:Węzeł:czas pracy z licznika msw [h];Czas pracy msw# (2, 4)
21 2 Ktz Armii Ludowej:Węzeł:temperatura zasilania z licznika [°C];Temp. zasilania# (2, 5)
22 2 Ktp Armii Ludowej:Węzeł:temperatura powrotu z licznika [°C];Temp. powrotu# (2, 6)
23 1 Kml Armii Ludowej:Węzeł:moc z licznika lsw [kW];Moc lsw# (2, 7)
24 1 Kmm Armii Ludowej:Węzeł:moc z licznika msw [kW];Moc msw# (2, 8)
25 3 Kpl Armii Ludowej:Węzeł:przepływ z licznika lsw [t/h];Przepływ lsw# (2, 9)26 3 K
26 3 Kpm Armii Ludowej:Węzeł:przepływ z licznika msw [t/h];Przepływ msw# (2, 10)
27 1 Kmsl Armii Ludowej:Węzeł:moc szczytowa z licznika lsw [kW];Moc szczytowa lsw# (2, 11)
28 1 Kmsm Armii Ludowej:Węzeł:moc szczytowa z licznika msw [kW];Moc szczytowa msw# (2, 12)
29 0 Kinl Armii Ludowej:Węzeł:informacja z licznika lsw [-];Informacja lsw# (2, 13)
30 0 Kinm Armii Ludowej:Węzeł:informacja z licznika msw [-];Informacja msw# (2, 14)
34 5 Zupo Armii Ludowej:Węzeł:zawór mieszania otwarty [%];Zawór upustowy otw.# (3, 0)
35 5 Zupz Armii Ludowej:Węzeł:zawór mieszania zamknięty [%];Zawór upustowy zam.# (3, 1)
14 1 imco Armii Ludowej:Węzeł:czas trwania impulsu ruchu zaworem CO [s];-
31 0 wel Armii Ludowej:Węzeł:zakodowany stan wejść logicznych [-];-
32 5 Alat Armii Ludowej:Węzeł:praca automatyczna w lecie [-];-
33 5 Azim Armii Ludowej:Węzeł:praca automatyczna w zimie [-];-

```

2.4. Plik definable.cfg

Plik `definable.cfg` opisuje tak zwane parametry definiowalne przeglądającego (patrz Sekcja 6.1, czyli parametry których wartości mogą być przedstawiane na wykresach przez program przeglądający `draw`, a wyliczane są na podstawie innych parametrów zapisanych do bazy.

Plik jest wczytywany za pomocą biblioteki `libpar`, więc składa się z szeregu parametrów o zadanej nazwie i wartości, zgodnie opisem w rozdziale Sekcja 4.1. Linie puste i zaczynające się od znaku `#` są ignorowane.

Najpierw wczytywane są wartości parametrów o nazwach `def_offset` i `def_count`. Pierwszy z nich musi mieć wartość równą liczbie parametrów zdefiniowanych w pliku `PTT.act` (zobacz Sekcja 2.3) - czyli trzeciej liczbie w pierwszej linii tego pliku. Oznacza ona przesunięcie - indeks, od jakiego zaczynają się parametry definiowalne przeglądającego (indeks w sensie pliku `PTT.act`). Wartość parametru `def_count` oznacza ilość zdefiniowanych w pliku parametrów definiowalnych.

W kolejnych liniach dla każdego ze zdefiniowanych parametrów definiowalnych muszą znaleźć się dwie definicje. Pierwsza, o nazwie `def_par_formula(X)`, gdzie `X` należy zastąpić kolejnym numerem parametru (od 1 do wartości `def_count`) opisuje formułę definiującą parametr. Postać formuły jest bardzo podobna do tej z pliku `parcook.cfg` - zobacz Sekcja 2.1. Pełen opis znajduje się w rozdziale Sekcja 6.1.

Uwaga! Jeżeli baza wykorzystuje automatyczne indeksy (a więc nie istnieją indeksy bazy `CodeBase`), to zamiast indeksów bazowych wykorzystywane są wirtualne indeksy parametrów, równe ich indeksom `IPC`. Jest to rozwiązanie tymczasowe, mające pozwolić na współpracy programu `draw` z nowym formatem bazy danych. Zobacz też Sekcja 8.

Druga definicja dla parametru to `def_par_ptt(X)`, gdzie `X` również należy zastąpić przez numer parametru. Wartością musi być opis parametru zgodny z tym w pliku `PTT.act` (zobacz Sekcja 2.3), tyle że pozbawiony pierwszego pola - czyli indeksu `IPC`, który dla parametrów definiowalnych przeglądającego nie ma sensu.

Przykładowa zawartość pliku:

```
def_offset=378
def_count=7
def_par_formula(1)=117 #0 N 239 #0 N + 257 #0 N +
def_par_ptt(1)=2 Qsum Sieć: Sterownik: Sumaryczna moc ciepłowni [MW]; Sumaryczna
moc # (26, 3)
def_par_formula(2)=38 #0 N 94 #0 N + 218 #0 N +
def_par_ptt(2)=3 Msum Sieć: Sterownik: sumaryczna masa węgla do kotła
[t/h]; Sumaryczna masa # (26, 4)
def_par_formula(3)=11 #0 N 67 #0 N + 186 #0 N +
def_par_ptt(3)=3 Vsum Sieć: Sterownik: sumaryczna objętość węgla do kotła
[m3/h]; Sumaryczna objętość # (26, 5)
def_par_formula(4)=378 #10000 * 379 /
def_par_ptt(4)=3 Kmsm Sieć: Sterownik: sumaryczny stosunek energia / masa
[MWh/t]; Energia/masa sum. # (26, 6)
def_par_formula(5)=378 #10000 * 380 /
def_par_ptt(5)=3 Kosm Sieć: Sterownik: sumaryczny stosunek objętość / masa
[MWh/m3]; Energia/obj. sum. # (26, 7)
def_par_formula(6)=15 #750 / #1000 * #36 * #100 / #100 52 #72 * #1000 / #10 /
#2 + - #100 / /
def_par_ptt(6)=2 Wcs Kocioł 1: Sterownik: wyliczona wartość opałow
a [kJ/g]; Wyliczona wart. opał. # (26, 8)
def_par_formula(7)=71 #750 / #1000 * #36 * #100 / #100 108 #72 * #1000 / #10 /
#2 + - #100 / /
def_par_ptt(7)=2 Wcs Kocioł 2: Sterownik: wyliczona wartość opałow
a [kJ/g]; Wyliczona wart. opał. # (26, 9)
# A to jest komentarz.
```

2.5. Pliki konfiguracyjne programu draw

Program przeglądający draw korzysta z pliku `PTT.act` (Sekcja 2.3) oraz z własnych plików, opisujących dostępne zestawy wykresów (tzw. okna) i konkretne wykresy. Są dwa rodzaje tych plików, o bardzo zbliżonej składni. Pliki o nazwie `ekrnXXXX.cor`, gdzie `XXXX` należy zastąpić nazwą prefiksu konfiguracji zawierają spis okien i wykresów dla konfiguracji. Pliki o nazwie `ekrnXXXX.def` zawierają spis okien tzw. definiowalnych użytkownika, czyli zestawy wykresów samodzielnie tworzonych przez użytkownika (są one umieszczane w katalogu domowym użytkownika).

Plik `ekrnxxxx.cor` jest plikiem tekstowym. Wszystkie linie zaczynające się od znaku `#` (hash) traktowane są jako komentarz i mogą być pominięte, jednak ich kasowanie czy ręczna edycja nie ma większego sensu, ponieważ mogą być one zapisane przez program przeglądający wywołany z opcją:

```
/opt/szarp/bin/draw -drawupdateekrancor
```

Ponadto przy każdym zakończeniu program przeglądający zapisuje plik `ekrnxxxx.def` przypadku, gdy w danej sesji użytkownik przeglądał okna definiowalne i przynajmniej raz nacisnął w oknie tworzenia okna definiowalnego przycisk *Ze zmianami*.

Dobrze jest, jeśli dany wykres występuje tylko raz w pliku `ekrnxxxx.cor` (wszystkie wykresy w pliku `ekrnxxxx.def` pochodzą z pliku `ekrnxxxx.cor`), choć w praktyce zasada ta jest często łamana. Jest to o tyle niewygodne, że w oknie wyboru wykresów do okna definiowalnego umieszczone są nazwy

pierwszych okien w pliku `ekrnxxxx.cor`, w których występują te wykresy, co czasem może być mylące dla użytkownika (wybrał np. *Kocioł|Wydajność*, a pokazuje mu się potem *Wydajności ciepłowni|Wydajność kotła 1*).

W dalszej części tekstu jako nazwy pól przyjęto właśnie umieszczone przed nimi komentarze.

2.5.1. Struktura pliku `ekrnxxxx.cor`

```
nazwa_ciepłowni
ilość_okien
<opis_okna_1>
<opis_okna_2>
...
<opis_okna_n>
```

Pola:

- Nazwa: *nazwa_ciepłowni*

Typ: string, maksymalnie 80 znaków

Opis: Nazwa wyświetlana w tytułowym pasku okna programu przeglądającego.

- Nazwa: *ilość_okien*

Typ: int, bez ograniczeń

Opis: Ilość struktur *<opis_okna>*, które opisano dalej. Należy pamiętać, aby po dodaniu nowego okna do pliku bezwzględnie zwiększyć odpowiednio wartość pola *ilość_okien*, ponieważ program przeglądający nie zasygnalizuje błędu, jeśli w pliku jest więcej struktur *<opis_okna>* niż wynosi wartość *ilość_okien*. W takim przypadku okna o numerach powyżej *ilość_okien* zostaną zignorowane, a przy zapisie na żądanie pliku *ekrnxxxx.cor* pominięte. W przypadku, gdy struktur *<opis_okna>* jest mniej niż wartość *ilość_okien*, program nie będzie działał prawidłowo.

2.5.2. Struktura okna

```
WindowID
Title
NumberOfAxes_n
<opis_osi_0>
<opis_osi_1>
...
<opis_osi_n-1>
FirstDraw
NumberOfDraw_m
<opis_wykresu_1>
<opis_wykresu_2>
...
<opis_wykresu_m>
```

Każdy opis okna jest otoczony dodatkowymi liniami komentarza:

```
# okno x {
...
# koniec okna x }
```

które są automatycznie generowane przez program przy wymuszonym zapisie `ekrnxxxx.cor`. Nawiasy klamrowe ułatwiają przechodzenie od okna do okna w edytorze vi - klawiszem %.

Pola:

- Nazwa: *WindowID*

Typ: unsigned long, zapisany w hexie

Opis: Identyfikator okna, określający jego specjalne właściwości. Pole to jest najczęściej sumą bitową identyfikatora rodzaju okna (patrz niżej) i wyróżników wykresów, których dotyczą te właściwości. Najmłodszy bit dotyczy wykresu 1, starszy wykresu 2 itd. aż do 12 bitu, czyli ostatnie trzy cyfry w hexie mogą przybierać wartość od 000 (nie dotyczy żadnego wykresu) do 3ff (dotyczy 12 wykresów). Najstarsze 4 bity zarezerwowane są na identyfikatory rodzaju, i tak:

- 8000XXX - okno definiowalne, czyli okno, którego skład może użytkownik określić sam; standardowo bity wykresów są niewykorzystywane (ten identyfikator występuje tylko w `ekrnxxxx.def`, nigdy w `ekrnxxxx.cor`)
- 4000XXX - wydajności ciepłowni, rzadko używane okno; w oknie dodatkowym tworzony jest wykres tortowy z porcjami wyznaczonymi przez wykresy wyróżnione odpowiednimi bitami *WindowID*
- 2000XXX - sumowanie godzinowo, najczęściej używana funkcja, w wyniku której w oknie dodatkowym pojawiają się dodatkowe zestawienia dotyczące wykresów wyróżnionych odpowiednimi bitami; takich operacji dokonuje się na wydajnościach, przepływach i wszystkich pozostałych wielkościach liczonych jako pewna wielkość dzielona przez czas (zawsze - godziny); przykładowo: 2000005 oznacza, że sumowane mają być wykres nr 1 i wykres nr 3

Jest możliwe umieszczenie w oknie definiowalnym wydajności ciepłowni (wówczas identyfikator ma postać `c000XXX`) albo sumowania godzinowego (wówczas identyfikator ma postać `a000XXX`) i odpowiednich bitowych wyróżników wykresów. Generalnie jednak nie należy się tym przejmować, bo okna definiowalne są zwykle w pełni obsługiwane przez program. Należy natomiast pamiętać o tym, że nie są dozwolone identyfikatory mieszane, np. `6000XXX`.

Poza identyfikatorami będącymi sumami bitowymi występują jeszcze inne (oprócz 0 - rzadkie) identyfikatory:

- 0 - najczęstszy: zwykle okno bez żadnych szczególnych właściwości
- 1 - zawory: obecnie już nie stosowane, zachowane dla kompatybilności wstecz - wszystkie wykresy w oknie traktowane są jako binarne wskazania krańcówek zaworów, gdzie wartość 1 oznacza zawór otwarty, wartość -1 oznacza zawór zamknięty, natomiast pozycja pośrednia to brak danych; dla takiego okna program sam wyznacza osie i pola *DivBy* wykresów (patrz dalej)
- 2 - wykresy: rzadko używana prymitywna funkcja $y=f(x)$, bez żadnych filtrów czy uśrednień; takie okno tworzy tylko program przeglądający - w pliku `ekrnxxxx.cor` ta wartość jako identyfikator nie ma prawa się znaleźć
- 3 - stosunek: występuje tylko na Ciepłowni Głównej w Suwałkach - w oknie muszą być dokładnie dwa wykresy, których stosunek wartości pierwszego do drugiego umieszczony jest w oknie dodatkowym.

- 4 - różnica: występuje tylko na Ciepłowni Głównej w Suwałkach; w oknie dodatkowym podawana jest różnica wartości: ostatnia-pierwsza widocznego w oknie wyświetlanego wykresu, albo w przypadku rozdwojonego kursora wartość drugiego kursora minus wartość drugiego kursora.
- Nazwa: *Title*
Typ: char, maksymalna długość 50 znaków
Opis: Tytuł okna. Dozwolone są wszystkie znaki oprócz '*' (gwiazdka), który występuje tylko na początku tytułów okien definiowalnych. Ponieważ treść tytułu okna definiowalnego jest dowolny, '*' na początku nazwy w liście dostępnych okien pokazuje, że jest to okno definiowalne.
- Nazwa: *NumberOfAxes*
Typ: unsigned char
Opis: Ilość dostępnych osi, których może być nie więcej niż wykresów, czyli 12. Osie są indeksowane od zera.
- Nazwa: *FirstDraw*
Typ: unsigned char
Opis: Numer porządkowy pierwszego wykresu, zazwyczaj 1, czasami (np. Wydajności ciepłowni) 0.
- Nazwa: *NumberOfDraw*
Typ: unsigned char, maksymalnie 12
Opis: Ilość opisów wykresów. Wykresy indeksowane są od 1.

2.5.3. Struktura opisu osi

Maxval x
Minval x
Maxrozzsz x
Minrozzsz x
Procrozzsz x

Pola:

- Nazwa: *Maxval*
Typ: double
Opis: Maksymalna wartość osi Y.
- Nazwa: *Minval*
Typ: double
Opis: Minimalna wartość osi Y.
- Nazwa: *Maxrozzsz*
Typ: double

Opis: Maksymalna wartość rozszerzenia na osi Y (patrz opis *Procrozs*).

- Nazwa: *Minrozs*

Typ: double

Opis: Minimalna wartość rozszerzenia na osi Y (patrz opis *Procrozs*).

- Nazwa: *Procrozs*

Typ: unsigned char

Opis: Procent rozszerzenia odcinka *Maxrozs* - *Minrozs* na osi Y. Dla prawidłowego działania programu musi zachodzić zależność:

$$\text{Minval} \leq \text{Minrozs} \leq \text{Maxrozs} \leq \text{Maxval}$$

Najczęściej używane są osie liniowe, gdzie *Procrozs* ma wartość 0. Osie nieliniowe mają miejsce np. przy wydajności węzłów (w zimie wysokie, w lecie niskie). Osie nieliniowe należy stosować ostrożnie, ponieważ są one bardzo mylące - szczególnie należy uważnie dobierać rozszerzenie tak, aby wykresy jak najrzadziej przyjmowały wartości z pogranicza rozszerzonej i nierozszerzonej części osi. Rozszerzenie może być de facto zwężeniem, np. dla *Maxval*=100.0, *Minval*=0.0, *Maxrozs*=50.0, *Minrozs*=0.0 i *Procrozs*=25 [%].

2.5.4. Struktura opisu wykresu

```
Item y
Menuitem y
DivBy y
Color y
WhichRecord y
WhichField y / IPKName y
Drawunit y
Axisnr y
```

Pola:

- Nazwa: *Item*
- Typ: char, maksymalna długość 10 znaków
- Opis: Skrótowa nazwa wykresu, ta sama co PTT -> tab[y].sym, czyli 3 pozycja w odpowiednim wierszu *PTT.act* (patrz niżej opis *WhichRecord* i *WhichField*). Przy modyfikacji pliku *ekrnxxx.cor* na systemie z działającym parcook'iem tej pozycji można nie wpisywać (tzn. trzeba wpisać cokolwiek, ale nie musi to być zgodne z rzeczywistą nazwą parametru), gdyż wynika ona jednoznacznie z numeru rekordu i pola w rekordzie (patrz dalej).

- Nazwa: *Menuitem*

Typ: char, maksymalna długość 30 znaków, zalecane do 23

Opis: Pełna nazwa wykresu, ta sama co PTT -> tab[y].alt, czyli 7 - ostatnia (przed komentarzem) pozycja w odpowiednim wierszu *PTT.act* (patrz niżej opis *WhichRecord* i *WhichField*). Przy modyfikacji pliku *ekrnxxx.cor* na systemie z działającym parcook'iem tej pozycji można nie

wpisywać (tzn. trzeba wpisać cokolwiek, ale nie musi to być zgodne z rzeczywistą nazwą parametru), gdyż wynika ona jednoznacznie z numeru rekordu i pola w rekordzie (patrz dalej).

- Nazwa: *DivBy*

Typ: double

Opis: Przez ile należy dzielić wartość z bazy aby dostać wartość rzeczywistą, czyli 10 do potęgi miejsce przecinka, gdzie miejsce przecinka to PTT -> tab[y].dot , czyli 2 pozycja w odpowiednim wierszu *PTT.act* (patrz niżej opis *WhichRecord* i *WhichField*). Przy modyfikacji pliku *ekrnxxx.cor* na systemie z działającym parcook'iem tej pozycji można nie wpisywać (tzn. trzeba wpisać cokolwiek, ale nie musi to być zgodne z rzeczywistą nazwą parametru), gdyż wynika ona jednoznacznie z numeru rekordu i pola w rekordzie (patrz dalej).

- Nazwa: *Color*

Typ: unsigned char

Opis: Indeks koloru wykresu, od 1 do 12. Poszczególne kolory są zdefiniowane w */opt/szarp/resources/Motif/SzarpDraw/SzarpDraw.res*.

- Nazwa: *WhichRecord*

Typ: int

Opis: Tradycyjnie był to numer rekordu, w którym są dane z wykresu. Numery zaczynają się od 1, dla numeru 0 wykres jest niewyświetlany (tożsamy z brakiem danych). Numer rekordu jest w bazie wartością pola *ID*. *WhichRecord* można wyliczyć z numeru wiersza w *PTT.act* (pomijając pierwszy wiersz opisujący ilość linii pliku):

$$WhichRecord = \langle numer_wiersza \rangle \div 15 + 1$$

Nowa konwencja umożliwia nadanie temu polu wartości *-1*, która umożliwia identyfikację parametru przez nazwę a nie indeks w bazie.

Nazwa: *WhichField* lub *IPKName*

Typ: int, zakres 0 - 14, lub string

Opis: Tradycyjnie był to numer pola w rekordzie numer *WhichRecord*, zawierającego dane z wykresu. Numery zaczynają się od 0. Cały rekord ma 15 pól. Numer pola określa w bazie pole o nazwie *FIELDXX*, gdzie XX to *WhichField*. *WhichField* można wyliczyć z numeru wiersza w *PTT.act* (pomijając pierwszy wiersz opisujący ilość linii pliku):

$$WhichField = (\langle numer_wiersza \rangle - 1) \bmod 15$$

Odwrotnie: aby znaleźć w *PTT.act* wiersz opisujący dany parametr należy (pomijając pierwszy wiersz z informacjami ogólnymi) znaleźć wiersz o numerze:

$$\langle numer_wiersza \rangle = (WhichRecord - 1) * 15 + WhichField + 1$$

Jeżeli wartość *WhichRecord* była mniejsza od 0, to zamiast pola *WhichField* powinno wystąpić pole *IPKName*, zawierające pełną nazwę parametru. Decyzja, czy spodziewać się tej nazwy, czy też numeru pola jest podejmowana na podstawie wartości *WhichRecord*.

- Nazwa: *Drawunit*

Typ: char, maksymalna długość 10 znaków

Opis: Nazwa jednostki wykresu, ta sama co zawartość nawiasów kwadratowych pola PTT -> tab[y].full, czyli 6 pozycji w odpowiednim wierszu *PTT.act* (patrz wyżej opis *WhichRecord* i *WhichField*). Każda linia musi więc mieć jednostkę w nawiasach kwadratowych, o czym należy pamiętać modyfikując *PTT.act*, zwłaszcza przy temperaturach i wartościach bez mian, np. korektach

(należy wówczas wpisać jednostkę "nic" czyli [-]). Przy modyfikacji pliku *ekrnxxx.cor* na systemie z działającym *parcook*'iem tej pozycji można nie wpisywać (tzn. trzeba wpisać cokolwiek, ale nie musi to być zgodne z rzeczywistą nazwą parametru), gdyż wynika ona jednoznacznie z numeru rekordu i pola w rekordzie (patrz wyżej).

- Nazwa: *Axisnr*

Typ: unsigned char

Opis: Indeks osi odpowiadającej danemu wykresowi (patrz opis struktury osi). Indeksy zaczynają się od 0.

3. Program sender

Program *sender* wykorzystywany jest do komunikacji z demonami linii, a dokładnie do wysyłania parametrów do sterowników i do odbierania ewentualnych odpowiedzi.

3.1. Konfiguracja programu sender

Aplikacja *sender* w swojej poprzedniej wersji wykorzystywała do konfiguracji plik *sender.cfg*. W aktualnej wersji program *sender* do konfiguracji wykorzystuje wyłącznie plik *params.xml* z katalogu */opt/szarp/{aktualna konfiguracja}/config/*. Do konfiguracji programu *sender* wykorzystujemy dwa zasadnicze elementy pliku *params.xml*:

- Pierwszy z nich, *send_freq*, dotyczy częstotliwości wysyłania parametrów do sterownika i powinien zostać podany zaraz na początku pliku konfiguracyjnego.

```
<params xmlns="http://www.praterm.com.pl/SZARP/ipk"
        xmlns:exec="http://www.praterm.com.pl/SZARP/ipk-extra"
        version="1.0"
        read_freq="10"
        send_freq="10"
        title="Pamieci">
```

- Drugi z nich, *send*, może zostać zdefiniowany dowolną ilość razy. Miejsce jego definicji określa jednoznacznie, do którego sterownika zostanie wysłany właśnie definiowany parametr. Dokładniejszy opis tego parametru został podany w sekcji Sekcja 5.2.5.

3.2. Uruchamianie programu sender

Uruchamianie programu *sender*:

```
/opt/szarp/bin/sender --help
Usage: sender [OPTION...]
SZARP sender application daemon.
```

<code>--D<param>=val</code>	Set initial value of libpar variable <code><param></code> to <code>val</code> .
<code>-n, --no-daemon</code>	Do not fork and go into background, useful for debug.
<code>-, --help</code>	Give this help list
<code>--usage</code>	Give a short usage message
<code>-V, --version</code>	Print program version

Config file `szarp.cfg`:

These parameters are read from section 'sender' and are optional:

<code>log</code>	path to log file, default is <code>/opt/szarp/log/sender.log</code>
<code>log_level</code>	log level, from 0 to 10, default is from command line (<code>-Ddebug=needed_log_level</code> parameter) or 2.

Report bugs to `coders@praterm.com.pl`.

W aktualnej wersji, aplikacja `sender` może pracować w trybie "debug" lub w trybie normalnym (bez debugowania). Uruchomienie w trybie "debug" może nastąpić w kilku przypadkach:

-

```
/opt/szarp/bin/sender
```

Przykładowa zawartość sekcji `sender` pliku `szarp.cfg`:

```
:sender
log_level=3
log=/opt/szarp/logs/senderek.log
```

Jeżeli uruchomimy program bez podania żadnych parametrów w linii argumentów, ale w pliku konfiguracyjnym `szarp.cfg` w sekcji `sender` został zdefiniowany parametr `log_level` o wartości większej od 0, lub parametr `log_level` nie został zdefiniowany, przez co aplikacja `sender` przyjmuje jego domyślną wartość na 2. Dodatkowo, w pliku `szarp.cfg` w sekcji `sender`, można zdefiniować parametr `log`, który opisuje ścieżkę do pliku, do którego zapisywane będą wszystkie informacje diagnostyczne generowane przez program `sender`. Tak uruchomiona aplikacja będzie działać w tle (uruchomiona zostanie jako demon).

-

```
/opt/szarp/bin/sender -n
```

Podobnie jak w poprzednim przypadku, wszelkie informacje o sposobie zapisywania informacji diagnostycznych zostaną pobrane z pliku `szarp.cfg` lub zostaną przyjęte wartości domyślne. Aplikacja uruchomiona z parametrem `-n` (lub `--no-daemon`) nie będzie działać w tle.

-

```
/opt/szarp/bin/sender -n -Ddebug=5
```

Jeżeli w pliku `szarp.cfg` nie ma informacji o wartości parametru `log_level`, wówczas możemy ją zdefiniować z linii argumentów z wykorzystaniem parametru `debug` i w ten sposób określić poziom diagnostyki.

Uruchomienie w trybie normalnym następuje zawsze wtedy kiedy poziom diagnostyki został ustawiony na 0.

Po uruchomieniu aplikacji `sender` w trybie diagnostycznym, ważna jest obserwacja w dwóch miejscach:

- po uruchomieniu wszystkie parametry typu `send` zostaną odczytane przez aplikację i zapisane do pliku konfiguracyjnego, przez co będzie możliwa kontrola ich poprawności np.:

```

NumberOfPars=20 BasePeriod=10
Par 000 src=    12 dst=    48, 4 retry=1 rtype=13107200 Probe  skipped
Par 001 src=    15 dst=   305, 5 retry=1 rtype=13107201 Minute skipped
Par 002 src=   123 dst=  3890, 4 retry=1 rtype=13107202 Min10  SENT
Par 003 src=   112 dst=  4146, 1 retry=1 rtype=13107203 Hour    SENT
Par 004 src=   112 dst=  4146, 2 retry=1 rtype=13107204 Hour    SENT
Par 005 src=   112 dst=  4146, 3 retry=1 rtype=13107205 Hour    SENT
Par 006 src=   112 dst=  4146, 4 retry=1 rtype=13107206 Hour    SENT
Par 007 src=   112 dst=  4146, 5 retry=1 rtype=13107207 Hour    skipped
Par 008 src=   112 dst=  4146, 6 retry=1 rtype=13107208 Hour    skipped
Par 009 src=   112 dst=  4146, 7 retry=1 rtype=13107209 Hour    skipped
Par 010 src=   112 dst=  4146, 8 retry=1 rtype=13107210 Hour    skipped
Par 011 src=   112 dst=  4146, 9 retry=1 rtype=13107211 Hour    SENT
Par 012 src=   112 dst=  4146, 0 retry=1 rtype=13107212 Hour    SENT
Par 013 src=    21 dst=  5681, 1 retry=1 rtype=13107213 Day     SENT
Par 014 src=    22 dst=  5681, 2 retry=1 rtype=13107214 Min10  skipped
Par 015 src=     0 dst=   562, 0 retry=1 rtype=13107215 Const   SENT
Par 016 src=     0 dst=   562, 1 retry=1 rtype=13107216 Const   SENT
Par 017 src=     0 dst=   562, 2 retry=1 rtype=13107217 Const   SENT
Par 018 src=     0 dst=   562, 3 retry=1 rtype=13107218 Const   SENT
Par 019 src=     0 dst=   562, 4 retry=1 rtype=13107219 Const   SENT

```

Gdzie: Par - numer kolejny pozycji na liście, od 0

src - adres w pamięci dzielonej lub 0, jeżeli wysyłana jest stała

dst - (numer linii - 1) * 256 + numer jednostki jako znak (np. '0')

, numer parametru wejściowego jednostki

retry - liczba powtórzeń

rtype - nieważne (jest to identyfikator nadawcy)

Probe, Minute ... Const - typ pamięci dzielonej lub stała

SENT, skipped - postępowanie w wypadku, gdy nastąpi brak danych (SZARP_NO_DATA) (wysyłać, nie wysyłać)

- oraz podczas wysyłania komunikatu, co pozwoli nam zorientować się czy konfiguracja jest poprawna np.:

```

Message 00048 with param: 4 value:  -32768 rtype: 13107200 bad value, skipped
Message 00305 with param: 5 value:         0 rtype: 13107201 rejected
Message 04146 with param: 1 value:  -32768 rtype: 13107203 not confirmed
Message 04146 with param: 5 value:  -32768 rtype: 13107207 bad value, skipped
Message 04146 with param: 8 value:  -32768 rtype: 13107210 bad value, skipped
Message 04146 with param: 9 value:  -32768 rtype: 13107211 not confirmed
Message 04146 with param: 0 value:  -32768 rtype: 13107212 rejected
Message 05681 with param: 1 value:         0 rtype: 13107213 not confirmed
Message 05681 with param: 2 value:         0 rtype: 13107214 not confirmed
Message 00562 with param: 0 value:  -32768 rtype: 13107215 rejected
Message 00562 with param: 4 value:     9999 rtype: 13107219 status OK

```

Message, param - to co dst powyżej

value - pobrana wartość

rtype - jak wyżej

Najważniejszy jest komunikat na końcu:

- bad value, skipped - value jest równa SZARP_NO_DATA i nie należy wysyłać (ewentualnie adres w pamięci jest równy NO_PARAM), nic nie zostało wysłane do demona.
- - not confirmed - demon nie potwierdził odbioru komunikatu.
- - rejected - komunikat doszedł do demona, ale sterownik nie odpowiedział.
- - status OK - sterownik odpowiedział raportem na polecenie wysłania parametrów.

3.3. Algorytm działania programu sender

Program sender przy starcie inicjalizuje tablicę SterInfo, w której dla każdego wysyłanego parametru znajdują się następujące informacje:

```
typedef struct _SterData {
    tMsgSetParam msg; /* komunikat do wysłania */
    unsigned short srcaddr; /* adres (indeks IPC) wysyłanego parametru */
    unsigned char status; /* stan transmisji parametru:
MSG_SEND - do wysłania
MSG_NOSEND - nie będzie wysyłany (brak
danych)
MSG_CONF - czekamy na potwierdzenie
*/
    unsigned char avgkind; /* rodzaj średniej do wysłania */
    unsigned char sendnodata; /* czy wysyłać wartości 'NO_DATA' */
} tSterData;
```

Struktura komunikatu do wysłania jest następująca:

```
typedef struct _MsgSetParam {
    long type; /* typ wiadomości, wyliczany na podstawie
numeru linii i jednostki */
    struct {
        ushort param; /* numer parametru wyjściowego sterownika */
        short value; /* wartość parametru */
        long rtype; /* typ oczekiwanej odpowiedzi od demona,
dla każdego parametru sendera inny */
        unsigned char retry;
        /* ilość powtórzeń przy nieudanym wysłaniu,
niewykorzystywana w programie */
    } tSetParam;
} tMsgSetParam;
```

Wykorzystywane są dwie kolejki - jedna do wysyłania żądań ustawienia parametrów, druga dla odbierania odpowiedzi (potwierdzeń) od demonów.

W pętli program wykonuje następujące działania:

1. Łączy się z parcookiem i dla każdego parametru oddzielnie podłącza się do segmentu pamięci dzielonej zależnego od pola avgkind (rodzaju średniej) dla parametru. Kopiuje wartość do pola value komunikatu, przy czym jeżeli wartość jest równa NO_DATA a nie mamy wysyłać pustych wartości, to status jest ustawiany na MSG_NOSEND. Jeżeli mamy wysyłać stałą, to oczywiście zostaje stała (wpisana przy inicjalizacji do pola value komunikatu).
2. Dla każdego parametru który ma status MSG_SEND (do wysłania) próbuje wstawić do kolejki uprzednio przygotowany komunikat. Zapamiętuje jeżeli choć w jednym przypadku się to nie udało. Podobnie jak w innych miejscach w programie, operacje na kolejkach wykonywane są z flagą NO_WAIT - nieblokująco.
3. Jeżeli w poprzednim kroku nie udało się wysłać wszystkich komunikatów, program zakłada, że kolejka jest zapchana i próbuje wyjąć z kolejki tyle komunikatów ile jest parametrów (a więc na pewno nie mniej niż włożył). Wykorzystywany jest typ komunikatu 0, oznaczający dowolny komunikat. Zapchanie kolejki nie jest takie proste - kolejka ma standardowo wielkość 16 KB, pojedynczy komunikat to około 24 bajty (może zależeć od kompilatora - sposobu pakowania pól w strukturze), a więc kolejka mieści ponad 680 komunikatów. Jeżeli wkładamy komunikaty co 10 sekund (i nikt ich nie wyjmuje), kolejka zacznie się zapychać po prawie dwóch godzinach...
4. Jeżeli w kroku 1 nie udało się wysłać wszystkich komunikatów, powtórz kroki 2 i 3, a następnie przejdź do kroku 5.
5. Czekaj 10 sekund (lub tyle sekund ile wynosi wartość parametru send_freq zdefiniowanego w pliku params.xml).
6. Dla każdego parametru pobierz z kolejki odpowiedzi wszystkie komunikaty o oczekiwanym typie zwrotnym. Jeżeli odpowiedź dla danego parametru była oczekiwana (a więc pole status dla parametru miało wartość MSG_CONF), sprawdź czy pola param i value odebranego komunikatu są takie jak komunikatu wysyłanego (dla danego parametru). Jeżeli wszystko się zgadza, ustawiany jest status - MSG_NOREP jeżeli ilość pozostałych powtórzeń zgłoszona przez demona linii jest równa 0 lub MSG_OK w przeciwnym przypadku. Oznacza to także, że status MSG_NOREP (oznaczający błąd) jest ustawiany także w sytuacji, gdy po prostu w pliku konfiguracyjnym nie zażądaliśmy powtarzania. Status ten nie ma jednak żadnego wpływu na działanie sendera, wykorzystywany jest tylko do diagnostyki.
7. Z kolejki odpowiedzi usuwane są wszystkie pozostałe tam komunikaty.
8. Program przechodzi do kroku 1.

Należy zauważyć, że demony różnie podchodzą do współpracy z senderem. Najbardziej zaawansowany jest linedmn, który próbuje wysyłać potwierdzenia, co może skutkować poprawnym logowaniem przez sendera faktu dotarcia lub niedotarcia danych do sterownika (choć domyślnie logowanie jest wyłączone). Demon rsdmn wysyła potwierdzenia, ale z błędnym typem komunikatu, więc są one usuwane przez sendera z kolejki jako błędne. Demon mbtrudmn nie wysyła w ogóle potwierdzeń (co wydaje się najsensowniejszym rozwiązaniem). Pozostałe demony (w chwili pisanie tego rozdziału) nie współpracują w ogóle z senderem.

4. Biblioteka libpar

Rozdział opisuje format głównego pliku konfiguracyjnego systemu SZARP - `szarp.cfg` i bibliotekę `libpar` służącą do odczytywania wartości konfiguracyjnych z tego pliku. Unikalną własnością

stosowanego rozwiązania jest możliwość korzystania w pliku konfiguracyjnym z wartości opcji podanych przy uruchamianiu programu.

4.1. Format pliku `szarp.cfg`

4.1.1. Opis ogólny

Plik `szarp.cfg` zawiera parametry o zadanym nazwach i wartościach, podzielone na sekcje. Spośród sekcji (o dowolnych nazwach) jedna sekcja - sekcja globalna jest wyróżniona. Program pytający się (za pomocą biblioteki `libpar`) o wartość parametru może wskazać, w jakiej sekcji należy szukać parametru. Jeżeli parametr nie zostanie znaleziony we wskazanej sekcji, szukany jest w sekcji globalnej. Jeżeli program nie poda nazwy sekcji, parametr będzie szukany tylko w sekcji globalnej.

Poza deklaracjami sekcji i parametrów w pliku można znaleźć także *dyrektywy* i deklaracje *zmiennych*.

Wszędzie poza nazwami sekcji i parametrów oraz nazwą pliku w dyrektywie `$include` (a więc w komentarzach, dyrektywach, stałych znakowych i w wartościach parametrów) backslash na końcu linii powoduje jej kontynuację.

Edycję pliku `szarp.cfg` ułatwia wykorzystanie pliku do podświetlania składni edytora vim.

Wykorzystywany typ pliku to `libpar` (:set ft=libpar).

Linie puste są ignorowane. Podobnie linie zaczynające się od znaku `#` - traktowane są jako komentarze. Wszystkie istotne linie (poza komentarzami i pustymi) muszą kończyć się znakiem końca linii a nie np. końca pliku. W miejscach, gdzie dopuszczalny jest "biały znak", mogą wystąpić spacje i tabulacje w dowolnej ilości.

4.1.2. Sekcje

Sekcje oznacza się przez podanie nazwy sekcji poprzedzonej znakiem `:` (dwukropka) - przy czym dwukropek musi być pierwszym znakiem w linii, a między nim a nazwą sekcji nie może być żadnych znaków. Nazwa sekcji może zawierać znaki alfanumeryczne i podkreślenie. Sekcje o tej samej nazwie można deklarować wielokrotnie - będą one logicznie sklejane w jedną. Na początku pliku rozpoczyna się sekcja globalna - obowiązująca do napotkania początku innej sekcji. Sekcję globalną można także bezpośrednio zadeklarować przez podanie nazwy sekcji `global` (w tym i tylko w tym przypadku wielkość liter nie ma znaczenia).

```
# Teraz mamy sekcję globalną

:sekcja1

# A teraz sekcję o nazwie 'sekcja 1'

:global

# A teraz znowu sekcję globalną.
```

4.1.3. Parametry

Deklaracja parametru polega na podaniu nazwy parametru (znaki alfanumeryczne i podkreślenie), bezpośrednio po niej znaku = a potem wartości parametru:

```
nazwa=to jest wartość parametru
```

Jest to równoważne:

```
nazwa=to jest \
wartość parametru
```

Za wartość parametru podstawiany jest cały ciąg tekstowy, do końca linii (ale bez niego). Powtórne zadeklarowanie parametru nadpisuje starą wartość.

4.1.4. Dyrektywy

W pliku mogą występować tak zwane dyrektywy. Dyrektywy muszą zaczynać się na początku linii. Obecnie dostępne są następujące:

- *\$include "nazwa_pliku"* - powoduje tekstowe wklejenie w miejscu jej wystąpienia zawartości pliku o podanej nazwie. Nazwa pliku może zawierać dowolne znaki poza podwójnym cudzysłowem i końcem linii. Możliwe jest wklejanie rekurencyjne, do maksymalnej głębokości określonej przez stałą (obecnie 10).

-

```
$if warunek
...
$elseif warunek
...
$else
...
$end
```

Dyrektywa warunkowa. Gałęzie *\$elseif* i *\$else* są opcjonalne, przy czym pierwsza z nich może występować wiele razy. Warunki mają postać:

```
argument = argument
lub
```

```
argument <> argument
```

(odpowiednio test na równość i różność leksykograficzną argumentów). Argumentem może być pojedyncze słowo lub stała tekstowa w podwójnych cudzysłowach (np. "Ala ma kota"). Obsługiwane są poprawnie następujące sekwencje znaków:

```
\n
\t
\\
\"
```

Inną postacią argumentu może być wywołanie funkcji, postaci:

```
identyfikator ( parametr )
```

gdzie identyfikator zaczyna się od litery lub podkreślenia i może zawierać także cyfry, a parametr jest stałą znakową lub kolejnym wywołaniem funkcji. Wywołanie funkcji w czasie parsowania jest zastępowane zwracanym przez nią napisem. Dostępne funkcje opisane są w Sekcja 4.1.5.

Warunki są wyliczane podczas parsowania i parsowane są tylko gałęzie, dla których warunki były spełnione. Dyrektywy warunkowe można zagnieżdżać do głębokości 10.

4.1.5. Funkcje

Aktualnie zaimplementowane funkcje:

- *execute("komenda")* - zwraca to, co wypisze na stdout podana jako parametr komenda, zaimplementowane tylko pod Linuxem. Pod Windows zwraca pusty ciąg.
- *exec("komenda")* - jak wyżej, ale obcina znak końca linii, również dostępna tylko pod Linuxem. Pod Windows zwraca pusty ciąg.
- *platform()* - zwraca napis *linux* pod Linuxem i *windows* pod Windows.

4.1.6. Zmienne

W pliku konfiguracyjnym we wszystkich stałych tekstowych, nazwach sekcji i parametrów oraz zawartościach parametrów mogą występować wyrażenia postaci *\$nazwa\$*. Są one zastępowane aktualną wartością zmiennej o nazwie *nazwa*.

Zmiennym można przypisywać wartości w następujący sposób:

```
$nazwa$ := wartość
```

gdzie wartość jest albo stałą tekstową albo wywołaniem funkcji. Następujący fragment:

```
$command$ := "hostname -s"
$prefix$ := exec("$command$")
:Motif_$prefix$
```

na komputerze o nazwie *leg1* spowoduje powstanie sekcji o nazwie *Motif_leg1*.

Jeżeli zmiennej nie nadamy wartości początkowej, będzie ona zastępowana pustym tekstem. Zmienne nie mogą występować w parametrach dyrektywy *\$include*.

Zmienne można deklarować w linii komend za pomocą argumentu:

```
-D<nazwa>=<wartość>
```

Wymaga to odpowiedniej inicjalizacji biblioteki libpar w programie (zobacz Sekcja 4.2.1).

Poza tym biblioteka automatycznie udostępnia zmienne o nazwach *argc*, *argv0*, *argv1* itd., których wartością są odpowiednio ilość parametrów przekazanych do programu i kolejne parametry, począwszy od nazwy programu. Zmienne te nie uwzględniają argumentów programu służących do inicjalizacji wartości zmiennych libpar.

4.2. Interfejs biblioteki

Biblioteka korzysta z parsera pliku napisanego we fleksie i udostępnia proste API do pobierania wartości parametrów. Dane wewnętrzne biblioteki przechowywane są w statycznych, globalnych zmiennych. Nie zaleca się korzystania z biblioteki inaczej niż przez opisane API. Korzystanie z biblioteki wymaga włączenia do programu pliku nagłówkowego `libpar.h`.

4.2.1. Inicjalizacja biblioteki

Przed użyciem jakiejkolwiek funkcji z biblioteki konieczna jest jej inicjalizacja. Dokonuje się tego przez użycie jednej z dwóch funkcji:

- `libpar_init_with_filename(char *filename, int exit_on_error)` - inicjalizuje bibliotekę i wczytuje plik konfiguracyjny o podanej nazwie. Jeżeli wystąpił błąd podczas odczytu pliku i wartość zmiennej `exit_on_error` jest różna od 0, wywoływana jest funkcja `exit(1)` - program kończy działanie. Jeżeli zamiast nazwy pliku podamy `NULL`, wczytany zostanie plik konfiguracyjny z domyślnej lokalizacji - pod Linuxem jest to `/etc/szarp/szarp.cfg`. Pod Windows nie ma domyślnej lokalizacji i funkcja nie załaduje żadnego pliku (wystąpi błąd).
- `libpar_init()` - równoważne `libpar_init_with_filename(NULL, 1)`.

Aby możliwe było korzystanie w pliku konfiguracyjnym ze zmiennych zadeklarowanych w linii komend uruchamianego programu (parametry `-Dnazwa=wartość` - zobacz Sekcja 4.1.6), należy jeszcze wywołać funkcję `libpar_read_cmdline(int *argc, char *argv[])`. Argumentami funkcji są *wskazniki* do standardowych argumentów funkcji `main()`. Wartość tych argumentów jest modyfikowana - `argc` jest zmniejszany, a `argv` modyfikowany tak, że w efekcie argumenty obsługiwane przez bibliotekę są w dalszej części programu niewidoczne. Pozwala to na nieprzejmowanie się programu obsługą opcji `'-D'`.

Funkcja `libpar_reinit_with_filename(char *filename, int exit_on_error)` ponownie inicjalizuje bibliotekę plikiem konfiguracyjnym o podanej nazwie. Jeżeli zamiast nazwy zostanie podany `NULL`, biblioteka użyje pliku `/etc/szarp/szarp.cfg` (jest to dozwolone tylko w Linuksie). Parametry z pliku, którym poprzednio zainicjalizowano bibliotekę, przestają być dostępne. Jeżeli parametr `exit_on_error` ma wartość różną od 0 i podczas odczytu pliku wystąpi błąd, biblioteka wywoła funkcję `exit(1)`. Wywołania funkcji `libpar_reinit()` jest równoważne `libpar_reinit_with_filename(NULL, 1)`.

Ostatnim etapem korzystania z biblioteki powinno być wywołanie funkcji `libpar_done()`. Zwalnia ona zajmowaną przez bibliotekę pamięć i powoduje, że dostęp do wartości parametrów staje się niemożliwy.

4.2.2. Odczytywanie wartości parametrów

Do pobierania wartości parametrów zwykle używa się jednej z dwóch funkcji:

•

```
void libpar_readpar(char *section, char *par, char *buf,
                   int size, int exit_on_error);
```

Funkcja szuka w sekcji o nazwie `sections` (lub globalnej jeżeli podano `NULL`) parametru o nazwie `par`. Jeżeli znaleziono parametr, jego wartość jest kopiowana do bufora o adresie `buf` i wielkości `size` (jeżeli wartość parametru była dłuższa, to jest ona obcinana - w efekcie bufor może nie kończyć się znakiem `'\000'`). Parametr `exit_on_error` różny od 0 powoduje, że nieznanie parametru kończy program z

błędem. Wypisywany jest odpowiedni komunikat. Jeżeli nie znaleziono parametru, bufor nie jest modyfikowany.

•

```
char *libpar_getpar(char *section, char *par, int exit_on_error);
```

Funkcja działa podobnie do poprzedniej, ale wartość parametru jest kopiowana do nowo zaalokowanego bufora o odpowiedniej wielkości, którego adres jest zwracany. Zwolnienie bufora za pomocą funkcji *free()* należy do programu. Jeżeli nie znaleziono parametru i podano *exit_on_error* równe 0, funkcja zwraca *NULL*.

5. IPK - konfiguracja systemu SZARP

IPK jest opartym na XML formacie opisu konfiguracji systemu SZARP. Niniejszy rozdział opisuje założenia IPK, format samego pliku, tworzenie i modyfikację konfiguracji (w tym współpracę z programami korzystającymi z systemu konfiguracji SZARP 2.1), a także zawiera uwagi dla programistów piszących programy korzystające z IPK.

Dokumentacja ta nie jest przeznaczona dla użytkowników systemu, ale dla osób konfigurujących system oraz programistów rozwijających obecne i piszących nowe aplikacje wchodzące w jego skład. Zakłada się w szczególności znajomość ogólnej struktury systemu, w szczególności tradycyjnych plików konfiguracyjnych wersji 2.1.

5.1. Koncepcja

Notatka: Informacje w tym rozdziale mają obecnie charakter historyczny. W szczególności żadne programy SZARP nie używają już plików konfiguracyjnych w formacie 2.1, wszystkie nowo dodawane programy także korzystają wyłącznie z konfiguracji w formacie IPK.

5.1.1. Po co nowy format konfiguracji

Konfiguracja SZARP'a 2.1 składa się z wielu plików, opisujących różne aspekty systemu. Oddzielne pliki opisują sposób komunikacji ze sterownikami, przyporządkowanie danym ze sterowników konkretnych parametrów, parametry wyliczane na podstawie innych (dwóch rodzajów), strukturę bazy danych do zapisu parametrów, generowane raporty i wykresy. Dodatkowo każdy program (np. *analiza* czy *sender*) wprowadza zwykle własny plik konfiguracyjny.

Podejście takie jest z jednej strony dość elastyczne, w szczególności nie jest konieczne opisywanie elementów systemu, które nie są w danej instalacji wykorzystywane. Część zmian i modyfikacji może być wykonywana lokalnie, bez ingerencji w opis innych elementów.

Niestety, między opisywanymi w różnych miejscach elementami zachodzą często dość głębokie i skomplikowane zależności. W szczególności jako identyfikatory parametrów w systemie stosowane są albo indeksy IPC, albo numery linii w pliku *PTT.act* (które odpowiadają z grubsza indeksom w bazie).

Powoduje to jednak, że o ile dodawanie nowych elementów do systemu jest dość proste, to usuwanie czy też modyfikacja układu parametrów już istniejących jest dość skomplikowane.

Dodatkowo, pliki konfiguracyjne były konstruowane w taki sposób, aby łatwo mogły z nich korzystać programy. Na przykład większość plików zawiera na początku liczniki linii, co z dzisiejszego punktu widzenia jest anachronizmem, utrudniającym modyfikację konfiguracji przez człowieka.

Rozmiary poszczególnych instalacji także zapewne przerosły nieco początkowe zamierzenia twórców. W efekcie ręczna edycja konfiguracji stała się bardzo skomplikowana i pracochłonna, trudno się przy niej ustrzec błędów. Rozwiązaniem miał być program *parconf*, czyli edytor konfiguracji systemu SZARP.

Program parsował całą strukturę konfiguracji SZARP'a, a następnie pozwalał za pomocą języka poleceń na dokonywanie w niej modyfikacji, których wykonanie ręczne wymagałoby dokonania setek przeliczeń i zmian różnego rodzaju indeksów. Rewolucyjnym pomysłem było także wprowadzenie szablonów, które pozwalały na szybkie i proste dodawanie zestawu parametrów na podstawie stworzonego wcześniej opisu dla konkretnego sterownika.

Program *parconf* nie jest niestety pozbawiony wad. Dodanie nowego pliku konfiguracyjnego (dla nowego programu) wymagałoby dodania obsługi tego pliku do *parconfa*. Uciążliwe okazały się istniejące w programie błędy (które objawiają się tylko w niektórych przypadkach), skutecznie uniemożliwiające poprawne wygenerowanie niektórych konfiguracji. Dodatkowo, *parconf* jest napisany w języku skryptowym Tcl/Tk. Efektem jest nie tylko powolne działanie, ale, w połączeniu z dość enigmatycznym stylem kodowania autora, powoduje, że utrzymywanie i dalsze rozwijanie programu stało się bardzo trudne.

Wszystkie te czynniki przesądziły o rozpoczęciu prac nad nowym ujęciem problemu konfiguracji systemu SZARP.

5.1.2. Założenia

Robocza nazwa, jaka pojawiła się przy planowaniu nowego systemu konfiguracji SZARP'a to *Idealny Plik Konfiguracyjny*, stąd skrót *IPK*. Przy projektowaniu *IPK* przyjęto następujące wymagania:

- Nowy format musi być łatwy do edycji ręcznej (za pomocą edytora tekstowego).
- Wprowadzanie większości zmian powinno być lokalne (nie powodować konieczności zmian w innych miejscach).
- Musi być zachowana kompatybilność wstecz, w szczególności ze względu na nakład pracy nie jest możliwe dostosowywanie do nowego formatu istniejących programów.
- Powinien istnieć mechanizm typu szablon *parconfa*, ułatwiający tworzenie konfiguracji z gotowych elementów.
- Dodawanie nowych elementów (funkcjonalności, programów) powinno być proste i nie mieć wpływu na elementy już istniejące.

Przyjęte rozwiązanie polega na tworzeniu jednego pliku, obejmującego całość informacji zawartych w następujących plikach konfiguracyjnych SZARP 2.1:

- `parcook.cfg`
- `line*.cfg`
- `PTT.act`
- `definable.cfg`

- sender.cfg
- ekrn*.cor
- *.rap

Plik jest dokumentem XML o składni opisanej w rozdziale Sekcja 5.2. Centralnym pojęciem jest parametr, czyli obiekt o pewnych właściwościach, takich jak nazwa, precyzja itp., który może być prezentowany na wykresach i raportach oraz odpowiada mu wartość liczbowa uzyskiwana bądź ze sterownika, bądź na podstawie wyliczeń (być może wykorzystujących inne parametry).

Identyfikatorem parametru jest tylko i wyłącznie jego pełna nazwa (np. *Kocioł 1: Sterownik: temperatura zadana*), która w związku z tym *musi być unikalna w ramach danej konfiguracji*.

Informacje o wykresach, raportach, formułach itp. są przypisane do danego parametru, tzn. jeśli parametr występuje w jakimś raporcie, to informacja o tym znajduje się przy opisie parametru. Dzięki temu w jednym miejscu mamy zgrupowane wszystkie informacje dotyczące parametru. Oznacza to także, że jeśli chcemy np. zmienić nazwę raportu, to należy tego dokonać dla wszystkich występujących w nim parametrów. Zwykle nie powinno to stanowić jednak większego problemu, np. można wykorzystać mechanizm typu "znajdź i zastąp" edytora tekstowego.

Zaprojektowany format jest dość odporny na błędy. Błędy typu składniowego oczywiście łatwo jest usunąć, natomiast trudno jest popełnić błąd innego rodzaju, gdyż uniemożliwia to struktura samego pliku. Jedyną rzeczą nie ujętą w strukturze jest sprawa indeksów w bazie, która wymaga pewnej uwagi edytującego konfigurację.

Wadą użycia XML-a i zgrupowania wszystkich informacji w jednym dokumencie jest duża objętość pliku konfiguracyjnego i w związku z tym możliwe trudności w wyszukiwaniu danego fragmentu. Zaleca się w związku z tym korzystanie z mechanizmów wyszukiwania używanego edytora. Zaletą jest natomiast łatwa rozszerzalność (dodawanie elementów i atrybutów nie wpływa na działanie dotychczasowych aplikacji) a także możliwość korzystania z różnego rodzaju narzędzi (edytorów, przeglądarek) do XML-a i łatwa konwersja na inne formaty. W szczególności wiele operacji na konfiguracji może być wykonanych z wykorzystaniem szablonów XSLT. Szczegóły znajdują się w rozdziale Sekcja 5.3. Poza tym parsery XML są dostępne w praktycznie wszystkich obecnie używanych językach programowania.

Zachowanie zgodności wstecz osiągnięto przez stworzenie narzędzi, umożliwiających obustronną konwersję konfiguracji ze starego na nowy format i odwrotnie. Dzięki temu można dokonywać zmian w konfiguracji korzystając z IPK, a następnie wygenerować konfigurację w starym formacie, używanym przez programy nie korzystające z IPK. Należy zaznaczyć, że dla niektórych konfiguracji automatyczna konwersja na IPK nie jest możliwa, wymaga to nieco pracy ze strony edytującego konfigurację.

Korzystanie przez programy bezpośrednio z dokumentu XML mogłoby być czasami uciążliwe. Stąd częścią projektu jest biblioteka, pozwalająca odczytywać i edytować informacje o konfiguracji. Biblioteka potrafi też odczytać konfigurację w starym (tradycyjnym) formacie, choć z biegiem czasu będą się pojawiać informacje, które w starym formacie nie będą mogły być zapisane (np. obsługa nowego formatu bazy z indeksami automatycznymi).

5.2. Format pliku IPK

Plik konfiguracyjny IPK jest pojedynczym dokumentem XML 1.0, zgodnym ze schematem RelaxNG umieszczonym w pliku `resources/dtd/ipk-params.rng` (lokalizacja pliku względem głównego katalogu SZARP), używającym przestrzeni nazw o identyfikatorze

<http://www.praterm.com.pl/SZARP/ipk>. Więcej informacji można znaleźć:

- www.w3.org/XML (<http://www.w3.org/XML>) - specyfikacja XML 1.0 i DTD
- www.w3.org/TR/REC-xml-names (<http://www.w3.org/TR/REC-xml-names>) - specyfikacja XML Namespaces
- www.relaxng.org (<http://www.relaxng.org>) - specyfikacja RelaxNG

Wspomniany schemat RelaxNG zawiera szczegółowy opis składni IPK, w tym poszczególnych elementów i atrybutów, ich dopuszczalnych wartości. Plik ten jest źródłem informacji nadrzędnym w stosunku do tej dokumentacji i odzwierciedla stan aktualny, który może być różny od opisanego w tej dokumentacji w przypadku opóźnień w uzupełnianiu dokumentacji.

5.2.1. Struktura pliku

Struktura pliku odpowiada fizycznej strukturze systemu, w szczególności zawiera dane o parametrach w takiej kolejności, w jakiej przychodzą one ze sterowników (wynika to właściwie ze sposobu działania programu parcook). Dane z poszczególnych sterowników są pogrupowane. Oddzielnie zgrupowane są parametry definiowalne przeglądającego oraz parametry definiowalne parcooka, które nie są redefinicjami istniejących fizycznie parametrów. Ogólna struktura wygląda więc następująco:

```
<params>
  <device>
    Opis parametrów ze sterownika 1
    ...
  </device>
  <device>
    Opis parametrów ze sterownika 2
    ...
  </device>
  ....
  <defined>
    Opis parametrów definiowalnych (parcooka)
  ...
  </defined>
  <drawdefinable>
    Opis parametrów definiowalnych (przeglądającego)
  ...
  </drawdefinable>
  <boilers>
    Opis konfiguracji kotłów dla programu analiza
  ...
  </boilers>
</params>
```

Na początku pliku znajduje się deklaracja XML, może ona także zawierać informację o kodowaniu. Nazwy, atrybuty i zawartości elementów, a także ewentualne komentarze muszą być zapisane w podanym kodowaniu. Jeśli kodowanie nie jest podane, przyjmuje się Unicode (UTF-8 lub UTF-16). W dowolnym miejscu w pliku mogą znajdować się XML-owe komentarze.

5.2.2. Element params

Nadrzędnym elementem jest element *params*. Zawiera on informację o wersji IPK (aktualnie 1.0), tytuł konfiguracji i okresy (w sekundach), co ile odbywa się odpytywanie sterowników o dane (przez demony linii) i wysyłanie danych przez program sender. Tytuł konfiguracji będzie widoczny jako tytuł okna programu przeglądającego. Do prawidłowego działania szablonów XSLT powinna być także obecna deklaracja przestrzeni nazw (atrybut *xmlns*).

Przykładowy początek pliku może wyglądać tak:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<!--
  To jest konfiguracja dla Katowic.
-->
<params xmlns="http://www.praterm.com.pl/SZARP/ipk" version="1.0"
        read_freq="10" send_freq="10" title="ZEC Katowice Wydział V">
```

Atrybuty element *params*:

- *xmlns* - deklaracja przestrzeni nazw SZARP - <http://www.praterm.com.pl/SZARP/ipk>, atrybut obowiązkowy. W razie potrzeby można dodać deklaracje innych przestrzeni nazw, zgodnie ze standardem XML.
- *version* - wersja formatu pliku, musi być równa 1.0, atrybut obowiązkowy.
- *read_freq* - długość w sekundach trwania pojedynczego cyklu pracy programu parcook, czyli częstotliwość wymieniania danych z postaciami elementami systemu SZARP. Standardowo jest to 10, zmiana tej wartości jest niezalecana i może powodować niepożądane efekty, w szczególności zafalszowanie wartości wyliczanych przez program średnich. Atrybut obowiązkowy.
- *send_freq* - długość w sekundach trwania pojedynczego cyklu pracy programu sender, czyli częstotliwość wysyłania danych (zadawania parametrów) do sterowników podłączonych regulatorów czy innych urządzeń. Rzeczywista częstotliwość wysyłania danych do urządzenia jest ustalana przez sterownik urządzenia, parametr ten mówi tylko o częstotliwości wysyłania danych z sendera do sterownika. Standardowo jest to 10, zmiana tej wartości jest niezalecana. Atrybut obowiązkowy.
- *title* - tytuł konfiguracji - zwykle nazwa obiektu, z którego zbierane są dane. Tytuł ten wyświetlany jest przez programy użytkowe SZARP, np. w tytule okna programu przeglądającego. Atrybut obowiązkowy.
- *ps_address* - adres IP związanego z konfiguracją serwera do ustawiania wartości parametrów. SZARP umożliwia opcjonalnie zdalne zadawanie wartości parametrów do regulatorów, wymaga to konfiguracji serwera psetd, o podanym w tym atrybucie adresie. Atrybut opcjonalny.
- *ps_port* - numer portu serwera zadawania parametrów o adresie podanym w poprzednio opisanym atrybucie. Atrybut opcjonalny.
- *documentation_base_url* - adres strony WWW serwera wyświetlającego opcjonalną dokumentację parametrów SZARP. W przypadku braku atrybutu używana jest wartość domyślna, obecnie <http://www.szarp.org>, która jednak może się zmienić w przyszłej wersji oprogramowania. Atrybut jest opcjonalny, ale zaleca się jego ustawianie.

5.2.3. Element device

W elemencie *params* umieszcza się elementy *device*. Opisują one pojedynczą linię komunikacyjną, obsługiwaną przez konkretnego demona linii. Zawierają informacje z pliku `parcook.cfg`, takie jak ścieżka do urządzenia i demona linii, prędkość transmisji, ilość bitów stopu. Możliwe są następujące atrybuty:

- *daemon* - ścieżka do demona linii odpowiedzialnego za obsługę danej linii. Parametr jest przekazywany do programu `parcook`, więc jeśli nie jest obecny, zostanie użyta wartość domyślna wybrana przez ten program.
- *path* - ścieżka do urządzenia (portu komunikacyjnego), na którym ma działać demon linii. Jeśli nie zostanie podana, demon linii użyje domyślnego portu (zależnie od numeru linii, czyli kolejności, w jakiej dany element *device* wystąpił w pliku konfiguracyjnym).
- *speed* - prędkość komunikacji w bodach, domyślną wartość ustala demon linii.
- *stop* - ilość bitów stopu, parametr dla demona linii, możliwe wartości to 1 lub 2.
- *protocol* - wersja protokołu, parametr przyjmowany na przykład przez demona `rsdmn`, liczba całkowita, zwykle 0 lub 1.
- *special* - dla niektórych demonów w pierwszej linii pliku `lineX.cfg` należało podawać specjalną wartość, nie będącą licznikiem jednostek (a więc właściwie tylko 0). Atrybut ten przechowuje tą wartość.
- *options* - dodatkowe opcje do demona linii, dodawane na końcu linii w pliku `parcook.cfg`. Opcje te nie są interpretowane, może być tu podany dowolny napis, który zostanie przekopiowany do tego pliku. Są przydatne dla demonów linii, które wymagają dodatkowych, niestandardowych parametrów. *Uwaga! Opcje powinny zaczynać się od znaku '-' (minus). Każdy napis zaczynający się od minusa, występujący w pliku `parcook.cfg`, będzie interpretowany jako opcje przez program `szarp2ipk`.*

Przykładowy element *device* zwykle wygląda więc tak:

```
<device daemon="/opt/szarp/bin/linedmn" path="/dev/ttyA12">
```

W bardziej skomplikowanych przypadkach może to być np.:

```
<device daemon="/opt/szarp/bin/rsdmn" path="/dev/ttyA12"
  speed="4800" stop="1" special="0">
```

5.2.4. Elementy radio i unit

W ramach jednej linii może występować kilka (co najmniej jedna) tzw. jednostek komunikacji. Każda z tych jednostek ma odpowiadający sobie element *unit*. Struktura elementu *device* wygląda w takim przypadku następująco:

```
<device >
  <unit>
    Lista parametrów jednostki 1
    ...
  </unit>
  <unit>
    Lista parametrów jednostki 2
```

```

    ...
  </unit>
  ...
</device>

```

Jeżeli mamy do czynienia z komunikacją radiową, dochodzi nam jeszcze jeden element, pośredni między *device* a *unit* - identyfikator linii radiowej (modemu radiowego), czyli element *radio*. Wtedy struktura elementu *device* przedstawia się następująco:

```

<device >
  <radio>
    <unit>
      Lista parametrów jednostki 1 modemu 1
      ...
    </unit>
    <unit>
      Lista parametrów jednostki 2 modemu 1
      ...
    </unit>
    ...
  </radio>
  <radio>
  ...
  </radio>
  ...
</device>

```

Element *radio* ma tylko jeden atrybut *id* - tekstowy identyfikator linii radiowej, w postaci jednego znaku ASCII, nie może się powtarzać w ramach linii komunikacyjnej. Przykładowa deklaracja:

```

  <radio id="W">
  ...
  </radio>

```

Element *unit* zawiera listę wchodzących w skład danej jednostki parametrów, najpierw odczytywanych ze sterownika (elementy *param*) a następnie (opcjonalnie) wysyłanych do sterownika (elementy *send*). Właściwości konkretnej jednostki komunikacyjnej opisuje kilka atrybutów (wszystkie są wymagane):

- *id* - identyfikator jednostki, w postaci jednego znaku ASCII, nie może się powtarzać w ramach linii komunikacyjnej.
- *type* - typ raportu, wartość interpretowana przez sterownik (liczba, najczęściej 1).
- *subtype* - podtyp raportu, wartość interpretowana przez sterownik (liczba, najczęściej 1).
- *bufsize* - wielkość bufora uśredniania dla parametrów w bajtach (liczba).

Przykład deklaracji elementu *unit*:

```

  <unit id="5" type="1" subtype="2" bufsize="3">
  ...
  </unit>

```

5.2.5. Element send

Wchodzące w skład jednostki elementy *send* opisują elementy wysyłane do sterownika przez program sender (w SZARP 2.1 opis tych elementów znajdował się w pliku `sender.cfg`). Elementy *send* nie posiadają żadnych identyfikatorów, są rozpoznawane na podstawie kolejności wystąpienia wewnątrz danej jednostki komunikacyjnej (elementu *unit*). Elementy *send* posiadają następujące atrybuty:

- Jeden z atrybutów *param* albo *value* (wymagany jest jeden z nich). *param* opisuje nazwę parametru (zwykle definiowalnego), którego wartość ma zostać wysłana do sterownika. Musi istnieć parametr o takiej nazwie (a więc element *param* z atrybutem *name* o takiej zawartości). Zamiast *param* może wystąpić atrybut *value*, którego wartością jest liczba (stała) wysyłana do sterownika.
- *type* - rodzaj wysyłanej próbki, jeden z "probe", "min", "min10", "hour", "day" (odpowiednio: próbka, średnia minutowa, 10-minutowa, godzinna, dzienna). Domyślnie przyjmuje się "probe". Przy wysyłaniu stałej wartości tego atrybutu nie ma znaczenia.
- *repeat* - liczba, ilość powtórzeń, jeśli sterownik nie potwierdził przyjęcia parametru, domyślnie 1. Praktycznie wartość tego atrybutu nie ma żadnego znaczenia (zobacz Sekcja 3.1).
- *send_no_data* - jeśli atrybut jest obecny (niezależnie od wartości), to sender będzie wysyłał do sterownika wartość NO_DATA. Jeśli go nie ma, to w przypadku jeśli miałby wysyłać taką wartość, nie będzie wysłane.

Przykład:

```
<send param="Wiatromierz:LB-746:prędkość wiatru" type="hour"
      repeat="1" send_no_data="1"/>
```

Czasami zdarza się, że konfiguracja jednostki zawiera pewną ilość parametrów zadawanych, tymczasem nie są one skonfigurowane (a więc program sender nie będzie ich wysyłał). W takim przypadku w pliku IPK powinien pojawić się element *send* pusty, także bez żadnych atrybutów. Jest to istotne zwłaszcza w sytuacji, gdy po parametrach nieskonfigurowanych ma nastąpić jakiś skonfigurowany.

5.2.6. Element param

Najważniejszą częścią elementu *unit* są elementy *param*, opisujące parametry odczytywane ze sterownika. Identyfikatorem parametru jest nazwa, unikalna w obrębie całej konfiguracji. To, jakiemu fizycznemu parametrowi odpowiada dany opis, jest ustalane na podstawie kolejności parametrów w jednostce komunikacyjnej - kolejność opisów musi odzwierciedlać kolejność parametrów ze sterownika. Możliwe atrybuty to:

- *name* - unikalny identyfikator parametru, powinien mieć postać 3 pól tekstowych oddzielonych dwukropkami, np.: "Kocioł 1: Sterownik: temperatura zadana".
- *short_name* - wymagany, skrócona nazwa parametru, np. "Tza". Dobrze żeby była unikalna przynajmniej w ramach jednostki komunikacyjnej (choć samo IPK żadnych takich wymagań nie nakłada). Długość skróconej nazwy powinna wynosić od 1 do 4 znaków.
- *draw_name* - nazwa używana jako nazwa wykresu parametru w programie przeglądającym, np. "Temperatura zadana". Dobrze jeśli nie jest dłuższa niż 20 znaków. Jeśli nie jest podana, użyta będzie nazwa pełna.
- *unit* - nazwa jednostki w jakiej prezentowany jest parametr, np. "°C". Jeśli nie jest podana, oznacza to brak jednostki, czyli "-".

- *base_ind* - indeks parametru w bazie. Dla bazy w formacie SZARP 2.1 jest to liczba, równa numerowi linii w pliku PTT.act pomniejszonemu o 2 (a więc indeks pierwszego parametru to 0). Dla nowego formatu bazy SzarpBase może to być napis *auto*, oznaczający indeks automatyczny (nowa baza nie wymaga indeksów, posługuje się nazwami parametrów). Jeśli atrybut nie występuje, oznacza to że parametr nie jest zapisywany do bazy.

Uwaga! Wartość atrybutu "auto" nie jest obsługiwana przez stary format konfiguracji. Możliwe jest wygenerowanie konfiguracji w formacie SZARP 2.1, ale nie wolno na podstawie tak wygenerowanej konfiguracji ponownie tworzyć pliku IPK, gdyż może spowodować to utratę informacji i wygenerowanie błędnej konfiguracji. Nie powinno się także mieszać w jednym pliku indeksów automatycznych i tradycyjnych, gdyż wygenerowana z takiego pliku konfiguracja w formacie SZARP 2.1 nie będzie poprawnie obsługiwana np. przez program przeglądający.

- *prec* - ilość miejsc po przecinku. Parametry liczbowe przechowywane są w bazie (i w pamięci dzielonej) w postaci liczby stałoprzecinkowej - ten atrybut określa ilość miejsc po przecinku w reprezentacji parametru. Jeżeli mamy wartość w bazie 118 i precyzję 0, oznacza to, że rzeczywista wartość jest równa 118, dla precyzji 1 - 11.8, a dla precyzji 3 - 0.118. Dla parametrów o wartościach dyskretnych (np. "Tak" / "Nie") opis odpowiednich wartości zawiera element *value*

Opis parametru uzupełniają elementy wchodzące w skład elementu *param*. Są to elementy *value*, *define*, *raport*, *draw* i *analysis* (występują właśnie w tej kolejności).

Dodatkowo parametr może zawierać element *doc* z przestrzeni nazw <http://www.praterm.com.pl/SZARP/ipk-extra>, którego tekstowa zawartość jest używana jako dokumentacja parametru na serwerze parametrów w sytuacji, kiedy nie zadziałają inne sposoby uzyskania dokumentacji (czyli dokumentacja generowana na podstawie dokumentacji programu technologicznego lub na podstawie formuły).

5.2.7. Element value

Wspomniane już elementy *value* opisują możliwe wartości parametrów dla parametrów dyskretnych. Jeżeli np. parametr może przyjmować wartości "Tak" i "Nie", którym w bazie odpowiadają wartości 1 i 0, to element *param* będzie zawierał dwa elementy *value* wyglądające tak:

```
<value int="0" name="Nie"/>
<value int="1" name="Tak"/>
```

Przykład pokazuje dwa możliwe (i wymagane) atrybuty elementu *value*:

- *int* - wartość liczbową parametru (w bazie lub pamięci dzielonej).
- *name* - tekstowa reprezentacja wartości (np. w raportach).

5.2.8. Element define

W skład opisu parametru może też wchodzić element *define*. Jego obecność oznacza, że wartość parametru ma być wyliczana na podstawie podanej formuły. Jeżeli parametr, w którym element *define* wystąpił, występuje w elemencie *defined* (lista parametrów definiowalnych parcooka) lub *drawdefinable*

(parametry definiowalne przeglądającego), to wartość parametru jest po prostu wyliczana na podstawie podanej formuły. Jeżeli zaś parametr jest "normalnym" parametrem ze sterownika (występuje w elemencie *unit*), to obecność formuły oznacza, że parametr jest niejako przeddefiniowany - w bazie i na wykresach będzie obecna jego wartość wyliczona na podstawie formuły, zamiast tej otrzymanej ze sterownika. Konstrukcja taka jest używana najczęściej do wprowadzania różnego rodzaju poprawek do wartości parametrów.

Element *define* ma następujące atrybuty:

- *type* - typ formuły opisującej wartość parametru. Możliwe wartości to:
 - *RPN* - oznacza parametr wyliczany przez program parcook, zawarty albo w jednej z sekcji *device* albo *defined* (ale nie *drawdefinable*). Drugi atrybut - *formula* - powinien zawierać formułę zapisaną w odwrotnej notacji polskiej (ONP, RPN - reverse polish notation). Składnię dostępnych operatorów opisuje rozdział Sekcja 6.2.
Opcjonalnie atrybut *formula* może zawierać wyrażenie *null*, wtedy element może zawierać podelement *script*, zawierający tekstowo formułę w języku LUA.
 - *DRAWDEFINABLE* - oznacza parametr wyliczany przez program przeglądający, czyli zawarty w sekcji *drawdefinable*. Atrybut *formula* zawiera w takim przypadku formułę również w postaci RPN, ale z nieco innym zestawem operatorów - zobacz Sekcja 6.1.
 - *LUA* - oznacza również parametr z sekcji *drawdefinable*, obliczany przez program przeglądający, ale zapisany jako formuła (skrypt) w języku Lua. Formuła może być zawarta albo w atrybucie *formula*, albo jako tekstowa zawartość podelementu *script*. Składnię formuł w języku Lua opisuje rozdział Sekcja 6.4.
- *formula* - formuła opisująca sposób wyliczania wartości parametru. Jej dokładna postać zależy od atrybutu *type* (patrz Sekcja 6).

Kolejne atrybuty wykorzystywane są tylko jeśli atrybut *type* ma wartość *LUA* :

- *lua_formula* - wymagany, określa sposób liczenia średnich danego parametru, możliwe wartości to *av* lub *va*. Pierwsza wartość oznacza, że najpierw liczone są średnie z wartości parametrów składowych, a następnie tak wyliczone wartości podstawiane są do formuły. Druga wartość oznacza, że liczymy wartość formuły dla wszystkich próbek, a następnie wyciągamy z nich średnią.

Notatka: Wybór sposobu liczenia może zmieniać wyliczane wartości średnie. Przykładowo, założmy że mamy 2 parametry i w danym dniu każdy z nich był zbierany tylko przez część czasu, a formuła liczy sumę tych 2 parametrów. Przy ustawieniu sposobu obliczania *av* średnia dzienna będzie sumą średnich wartości tych parametrów w danym dniu. Ale jeżeli wybierzemy sposób *va*, to średnia dzienna będzie sumą średnich wartości parametrów, wybranych z momentów kiedy, oba parametry były zbierane naraz.

- *lua_start_date_time* - opcjonalny atrybut określający pierwszą datę i godzinę, dla której ma być wyliczany parametr. Dla parametrów LUA określenie zakresu czasowego w jakim parametr ma wartość jest w ogólnym przypadku niemożliwe. Podanie atrybutu może przyspieszyć obliczanie parametru, przez podpowiedzenie programowi, że przed daną datą nie należy go próbować obliczać. Wartość atrybutu musi być zgodna z formatem *RRR-MM-DD gg:mm*, godzinę podajemy w strefie czasowej GMT.

- *lua_start_offset* - opcjonalny atrybut jest alternatywnym sposobem określenia kiedy program ma zacząć obliczanie parametru. Wartość atrybutu to liczba sekund jaką należy dodać (lub odjąć jeśli liczba jest ujemna) od daty początkowej - albo wyliczonej automatycznie przez program pierwszej daty w bazie, albo podanej jako wartość atrybutu *lua_start_date_time*.
- *lua_end_offset* - opcjonalny atrybut podaje, kiedy mamy zakończyć obliczanie parametru, jako ilość sekund które należy dodać (odjąć jeśli liczba jest ujemna) od automatycznie wyznaczonej przez program daty końca danych w bazie.

Notatka: Przykładem zastosowania tych atrybutów jest np. określenie parametru, który jest zależny od danych sprzed 7 dni, czyli może być wyliczony na tydzień w przyszłość, ale nie ma sensu go liczyć z pierwszego tygodnia danych w bazie. W takiej sytuacji możemy ustawić zarówno *lua_start_offset* jak i *lua_end_offset* na 604800 (7 dni = 7 * 24 h = 7 * 24 * 3600 s = 604800 s).

Ewentualnie zawarty w elemencie *define* element potomny *script* nie ma żadnych atrybutów, a formuła zapisana jest jako tekstowa zawartość elementu. Aby uniknąć konieczności zastępowania encjami XML znaków o specjalnych znaczeniach (takich jak & czy <) całą zawartość można zawrzeć w sekcji *CDATA*, np.

```
<script><![CDATA[
if (p("gcie:Kocioł 4:Sterownik:Wydajność kotła", t, pt) < 0.25) then
v = nan()
else
v = 4.0
]]></script>
```

Element *define*, podobnie jak wiele innych elementów w konfiguracji IPK, dopuszcza dodawanie elementów i atrybutów z obcych przestrzeni nazw (zobacz Sekcja 5.2.18). W szczególności w elemencie można umieścić element *doc* z przestrzeni nazw <http://www.praterm.com.pl/SZARP/ipk-extra>. Jego zawartość będzie wyświetlana jako część dokumentacji parametru w przypadku korzystania z serwera dokumentacji parametrów. Przykład:

```
<param name="Sieć:Sterownik:zmodyfikowana temperatura zewnętrzna" short_name="Tzew" dr
unit="°C" prec="2" xmlns:doc="http://www.praterm.com.pl/SZARP/ipk-extra">
<define type="DRAWDEFINABLE" formula="(*:*:temperatura zewnętrzna) 150 -">
<doc:doc>Pomiar temperatury zewnętrznej skorygowany o 1,5 stopnia ze względu na złe
umiejscowienie czujnika temperatury.</doc:doc>
</define>
<draw name="Temperatury sieciowe" min="-30" max="50"/>
</param>
<define type="DRAWDEFINABLE"
```

5.2.9. Element raport

Kolejnym elementem mogącym wystąpić wewnątrz elementów *param* jest element *raport*. Jego obecność oznacza, że opisywany parametr ma wystąpić w raporcie o podanej nazwie. Jeżeli parametr ma wystąpić w kilku raportach, należy umieścić kilka elementów *raport*. Możliwe atrybuty to:

- *title* - tytuł raportu, wartość tekstowa, wymagany.
- *description* - opis parametru w raporcie, jeżeli nie występuje, to jako opis użyta będzie ostatnia część nazwy parametru (po drugim dwukropku).
- *filename* - nazwa pliku z opisem raportu w formacie SZARP 2.1. Jest używana, aby odtworzyć konfigurację z takimi samymi nazwami plików jak oryginalnie. W nowych konfiguracjach atrybut nie powinien być używany - przyjmowana jest wtedy wartość domyślna, równa wartości parametru *title*.
- *order* - liczba rzeczywista, pozwala na kontrolę kolejności występowania parametrów w raportach. Jest to priorytet z jakim parametr ma występować w raporcie, im mniejszy tym wyżej (wcześniej) w raporcie wystąpi parametr. Brak priorytetu lub wartość mniejsza niż 0 oznacza, że parametr ma występować po tych z dodatnim atrybutem. Parametry o tych samych atrybutach występują w kolejności jak w pliku IPK (czyli według indeksów IPC). Dotyczy to także w szczególności parametrów bez atrybutu.

Przykład:

```
<raport title="Kocioł WR 5" description="temperatura zadana"
filename="Kocioł_WR_5.rap" order="1"/>
```

Wszystkie elementy *raport* z danym tytułem tworzą opis jednego raportu.

5.2.10. Element draw

Element *draw* opisuje wykres, na którym ma być widoczna wartość parametru w programie przeglądającym. Podobnie jak w przypadku raportów, można umieścić kilka takich elementów, umieszczając w ten sposób parametr na kilku wykresach. Element *draw* ma następujące atrybuty:

- *title* - tytuł wykresu (okna z wykresami), wymagany. Wykresy o tym samym tytule tworzą opis okna programu przeglądającego.
- *max* - liczba rzeczywista - maksymalna wartość parametru, określająca górny zakres osi na wykresie parametru. Jest to wartość w takiej postaci, w jakiej będzie widziana na wykresie, a więc po wszelkich przeskalowaniach. Obecność tego atrybutu jest wymagana przez program przeglądający Szarp Draw 2.1.
- *min* - podobnie jak wyżej, tylko dolna granica zakresu wartości dla parametru. Uwaga - w IPK nie ma pojęcia "osi" (axis) występującego w plikach konfiguracyjnych eknrXXXX.cor. Osie są tworzone automatycznie na podstawie atrybutów *min* i *max*.
- *scale* - atrybut potrzebny do skalowania części wykresu. Program przeglądający pozwala przeskalować część wykresu, tak żeby pewien zakres wartości zajmował inną część osi, niż bez przeskalowania. Granice przedziału skalowania określają dwa następne atrybuty, zaś ten atrybut jest liczbą całkowitą, mówiącą o tym, ile procent osi powinien zajmować podany przedział wartości. Jeżeli atrybut jest nieobecny lub ma wartość 0, przeskalowanie nie jest dokonywane.
- *minscale* - liczba rzeczywista - dolny zakres wartości podlegających przeskalowaniu - patrz wyżej.

- *maxscale* - jak wyżej, tylko górny zakres wartości.
- *color* - kolor wykresu, teoretycznie w jednej ze standardowych postaci (a więc nazwa symboliczna, zapis szesnastkowy wartości RGB itp.). Dla programu SzarpDraw 2.1 jest wymagany, co więcej powinien mieć jedną z wartości: black, red, #FFC246, cyan, green, yellow, #B9F4BE, blue, magenta, #00B6FF, #A6A550, #FF825F, #AFAFFF (odpowiedniki kolorów 1 - 12 z pliku `ekrnXXXX.cor`). Jeśli nie zostanie podany, przy generacji pliku `ekrnXXXX.cor` kolory zostaną dobrane automatycznie.
- *prior* - priorytet wykresu - liczba rzeczywista dodatnia. W formacie konfiguracji SZARP 2.1 kolejność okien z wykresami była ustalana przez kolejność wpisów w pliku `ekrnXXXX.cor`. Aby umożliwić zachowanie tej kolejności, można podać priorytet - im mniejszy, tym wyżej będzie okno na liście. Priorytet dotyczy okna (a nie wykresu), uwzględniany jest najmniejszy priorytet spośród wszystkich wykresów należących do danego okna. Ponieważ jest to liczba rzeczywista nie trzeba przeliczać innych priorytetów przy przestawieniu jednego okna (zawsze da się coś wcisnąć). Zaleca się używanie tylko jednego atrybutu *prior* na okno (a więc tylko w jednym z wykresów należących do okna).
- *order* - podobnie jak *prior* jest to liczba rzeczywista dodatnia, ale dotyczy kolejności wykresu w oknie. Wykresy nie mające tego atrybutu będą umieszczane na końcu okna.
- *special* - atrybut specjalny dla program SzarpDraw 2.1, wartość tekstowa, jedna z:
 - "piedraw" - parametr jest liczony do wykresu kołowego
 - "hoursum" - parametr jest liczony do sumowania godzinowego
 - "valve" - zawory, atrybut dla kompatybilności wstecz
 - "rel" - jeśli w oknie są dwa wykresy, to w oknie dodatkowym pojawi się wyliczenie stosunku wartości parametrów z wykresów
 - "diff" - w oknie dodatkowym pojawia się różnica wartości końcowej i początkowej dla wykresu.
 - "none" - wartość domyślna, brak specjalnych właściwości.

Dokładne omówienie znaczenia tych atrybutów znajduje się w pliku `resources/documentation/draw.ekran.README.html` w repozytorium SZARP.

Przykładowy opis wykresu może wyglądać tak:

```
<draw title="Kocioł WR 5 - temperatury" prior="81" color="cyan" min="0"
max="150" order="2"/>
```

5.2.11. Element *treenode*

Element *draw* może zawierać jeden lub więcej elementów *treenode*. Elementy *treenode* służą do ustawiania zestawów wykresów w drzewko, którego liśćmi są zestawy wykresów, a węzłami elementy *treenode*. Każdy z elementów *treenode* może zawierać opcjonalnie jeden element *treenode* który określa ojca elementu.

Przykład:

```
<draw title="Kocioł WR 5 - temperatury" prior="81" color="cyan" min="0"
max="150" order="2">
<treenode name="Kocioł WR 5"/>
</draw>
```

W tym wypadku w programie przeglądającym jako dziecko korzenia drzewa zostanie dodany element "Kocioł WR 5", którego potomkiem będzie węzeł "Kocioł WR 5 - temperatury".

Przykład kolejny:

```
<draw title="Kocioł WR 5 - temperatury" prior="81" color="cyan" min="0"
max="150" order="2">
<treenode name="Kocioł WR 5">
<treenode name="Kotły"/>
</treenode>
</draw>
```

W tym wypadku w programie przeglądającym jako dziecko korzenia zostanie utworzony element "Kotły", którego potomkiem będzie węzeł "Kocioł WR 5", a ten węzeł będzie zawierał liść "Kocioł WR 5 - temperatury".

Element *treenode* może zawierać następujące atrybuty:

- *order* - określa on położenie elementu *treenode* wobec jako braci (elementów mających tego samego ojca) Przykład:

```
<draw title="Kocioł WR 5 - temperatury" ...>
<treenode order="1" name="Kocioł WR 5">
<treenode name="Kotły"/>
</treenode>
</draw>
```

...

```
<draw title="Kocioł WR 4 - temperatury" .. >
<treenode order="0" name="Kocioł WR 4">
<treenode name="Kotły"/>
</treenode>
</draw>
```

Taka konfiguracja oznacza że węzeł "Kocioł WR 4" będzie znajdował się przed węzłem "Kocioł WR 5".

- *draw_order* - określa on położenie liścia *draw* w danym elemencie *treenode*. Działa on analogicznie do elementu *order*. Przykład:

```
<draw title="Kocioł WR 5 - temperatury" ...>
<treenode draw_order="1" order="1" name="Kocioł WR 5"/>
</draw>
```

...

```
<draw title="Kocioł WR 5 - ciśnienia" .. >
<treenode draw_order="0" order="0" name="Kocioł WR 5"/>
</draw>
```

W tym wypadku liść "Kocioł WR 5 - temperatury" będzie umiejscowiony przed liściem "Kocioł WR 5 - ciśnienia".

5.2.12. Element analysis

Element *analysis* obecny w elemencie *param* oznacza, że wartości tego parametru stanowią dane, na których operuje program analiza.

Element *analysis* zawiera dwa atrybuty:

- *boiler_no* - dodatnia liczba całkowita - numer kotła
- *param_type* - opisuje typ danych przechowywany przez parametr

Atrybut *param_type* przyjmuje jedną z następujących wartości:

- *analysis_enable* - ustawione dwa najmniej znaczące bity wartości parametru oznaczają, że regulator kotła zezwala na wykonywanie analizy
- *regulator_work_time* - wartość parametru zawiera czas pracy regulatora kotła w pracy automatycznej
- *coal_volume* - aktualna objętość wchodzącego do kotła węgla
- *energy_coal_volume_ratio* - aktualna objętość stosunku energia/objętość węgla
- *air_stream* - względna ilość powietrza dla określonej (zależnej od typu kotła) objętości węgla
- *max_air_stream* - maksymalna dopuszczalna wartość strumienia powietrza podczas analizy
- *min_air_stream* - minimalna dopuszczalna wartość strumienia powietrza podczas analizy
- *left_grate_speed* - prędkość lewego rusztu
- *right_grate_speed* - prędkość prawego rusztu
- *left_coal_gate_height* - wysokość lewej warstwowicy
- *right_coal_gate_height* - wysokość prawej warstwowicy
- *analysis_status* - program umieszcza w parametrze wartość 1 gdy analiza jest wykonywana, 0 w przeciwnym wypadku
- *air_stream_result* - w tym parametrze analiza umieszcza wyliczoną wartość strumienia powietrza

5.2.13. Przykłady elementów param

Na zakończenie trzy przykłady opisów parametrów:

```
<param name="Kocioł WR 5: Sterownik: temperatura zadana" short_name="Tod"
      draw_name="Temp. zadana" unit="°C" prec="1" base_ind="0">
  <raport title="Kocioł WR 5" description="temperatura zadana"
    filename="Kocioł_WR_5.rap"/>
  <raport title="RAPORT TESTOWY" description="Kocioł WR 5
    Sterownik" filename="test.rap"/>
  <draw title="Kocioł WR 5 - temperatury"
    prior="81" color="cyan" min="0" max="150"/>
</param>

<param name="Kocioł; WR 5: Sterownik: stosunek energia / masa"
      short_name="Km" draw_name="Stos. energia/masa" unit="MWh/t" prec="3"
      base_ind="39">
  <define type="RPN"
formula="(*: *: stosunek energia / objętość) 1000 * (*: *: masa nasypowa węgla) / ">
  <raport title="Kocioł WR 5"
description="stosunek energia / masa"
filename="Kocioł_WR_5.rap"/>
  <draw title="Waga WR 5 - sprawność" prior="91"
```

```

color="cyan" min="0" max="8"/>
</param>

<param base_ind="auto" name="Kocioł 3:Sterownik:aktualna objętość węgla" short_name="Obj. węgla kotła 3" unit="m3/h" prec="3">
  <raport title="Waga 3" filename="Waga_3.rap" order="6"/>
  <draw title="Waga 3 - dostarczony węgiel" min="0" max="1.5" special="hoursum"/>
  <analysis boiler_no="3" param_type="coal_volume"/>
</param>

```

5.2.14. Elementy defined i drawdefinable

Poza opisanymi wcześniej elementami *device* w głównym elemencie *params* mogą występować jeszcze elementy *defined* i *drawdefinable*. Oba zawierają elementy *param* - listę parametrów definiowalnych. Różnica polega na tym, że w elemencie *defined* mamy parametry definiowalne wyliczane przez parcooka (a więc z typem formuły "RPN"), a w elemencie *drawdefinable* wyliczane przez program przeglądający, a więc z typem formuły "DRAWDEFINABLE", nie mogą też one wchodzić w skład raportów ani mieć indeksu w bazie.

5.2.15. Elementy boilers oraz boiler

W elemencie *params* może znajdować się element *boilers* zawierający konfiguracje kotłów, których praca podlega analizie. Element *boilers* zawiera co najmniej jeden element *boiler*, opisujący konfigurację pojedynczego kotła.

W ramach elementu *boiler* dopuszczalne i wymagane są następujące atrybuty:

- *boiler_no* - numer kotła, dodatnia liczba całkowita
- *grate_speed* - maksymalna procentowa zmiana prędkości rusztu podczas jednego cyklu analizy
- *coal_gate_height* - maksymalna procentowa zmiana wysokości warstwowownicy podczas jednego cyklu analizy
- *boiler_type* - typ kotła, parametr ten może przyjmować następujące wartości:
 - WR-25
 - WR-10
 - WR-5
 - WR-2,5
 - WR-1,25

Element *boiler* obowiązkowo musi zawierać trzy elementy *interval*.

5.2.16. Element interval

Element *interval* opisuje cykl pracy programu analiza. Stanowi obowiązkowy składnik elementu *boiler*. Zawiera następujące atrybuty, których wartości stanowią dodatnie liczby całkowite:

- *grate_speed_lower* - dolne ograniczenie prędkości rusztu podczas cyklu (mm/h)
- *grate_speed_upper* - górne ograniczenie prędkości rusztu podczas cyklu (mm/h)
- *duration* - długość cyklu (w sekundach)

Poniżej przedstawiono przykładową konfigurację kotłów:

```
<boilers>
  <boiler boiler_no="3" grate_speed="15.00" coal_gate_height="4.00" boiler_type="WR-
    <interval duration="10800" grate_speed_lower="1000" grate_speed_upper="5000"/>
    <interval duration="8040" grate_speed_lower="4000" grate_speed_upper="6000"/>
    <interval duration="7200" grate_speed_lower="5000" grate_speed_upper="9000"/>
  </boiler>
  <boiler boiler_no="4" grate_speed="15.00" coal_gate_height="4.00" boiler_type="WR-
    <interval duration="10800" grate_speed_lower="1000" grate_speed_upper="5000"/>
    <interval duration="8040" grate_speed_lower="4000" grate_speed_upper="6000"/>
    <interval duration="7200" grate_speed_lower="5000" grate_speed_upper="9000"/>
  </boiler>
</boilers>
```

W ramach elementu *boiler* dopuszczalne i wymagane są następujące atrybuty:

- *boiler_no* - numer kotła, dodatnia liczba całkowita
- *grate_speed* - maksymalna procentowa zmiana prędkości rusztu podczas jednego cyklu analizy
- *coal_gate_height* - maksymalna procentowa zmiana wysokości warstwownicy podczas jednego cyklu analizy
- *boiler_type* - typ kotła, parametr ten może przyjmować następujące wartości:
 - WR-25
 - WR-10
 - WR-5
 - WR-2,5
 - WR-1,25

Element *boiler* obowiązkowo musi zawierać trzy elementy *interval*.

5.2.17. Elementy seasons oraz season

W elemencie *params* może znajdować się element *seasons* opisujący sezony letnie. Element *seasons* może zawierać dowolną ilość elementów *season*. Element *season* musi zawierać następującą listę atrybutów:

- *year* - rok, dla którego zdefiniowana jest granica sezonu letniego
- *month_start* - miesiąc, w którym rozpoczyna się sezon letni (liczba z przedziału 1-12)
- *day_start* - dzień, w którym rozpoczyna się sezon letni (liczba z przedziału 1-31)
- *month_end* - miesiąc, w którym kończy się sezon letni (liczba z przedziału 1-12)
- *day_end* - dzień, w którym kończy się sezon letni (liczba z przedziału 1-31)

Element *seasons* musi zawierać atrybuty *month_start*, *day_start*, *month_end* oraz *day_end*, mają one identyczne znaczenie jak atrybuty elementu *season*, ale określają domyślne granice sezonu używane, gdy brak jest definicji sezonu dla danego roku.

Poniżej przedstawiono przykładową konfigurację sezonów:

```
<seasons month_start="4" day_start="4" month_end="10" day_end="5">
  <season year="2006" month_start="3" day_start="3" month_end="11" day_end="11"/>
  <season year="2004" month_start="5" day_start="3" month_end="10" day_end="9"/>
</seasons>
```

5.2.18. Elementy i atrybuty z innych przestrzeni nazw

Dla potrzeb konfigurowania niestandardowych demonów linii wprowadzono możliwość umieszczania w dokumencie IPK elementów i atrybutów z innych przestrzeni nazw. Elementy te mogą znaleźć się na początku elementu *device* oraz na końcu elementów *param* i *send*. Większość elementów (poza wymienionymi także *params* i *unit*) może zawierać też atrybuty z innych przestrzeni nazw. Szczegóły można znaleźć w kodzie konkretnych demonów, większość z nich używa przestrzeni nazw <http://www.praterm.com.pl/SZARP/ipk-extra>. Przykład:

```
<device daemon="/opt/szarp/bin/mbusdmn" path="/dev/ttyA11"
  xmlns:modbus="http://www.praterm.com.pl/SZARP/ipk-extra"
  modbus:mode="master">
  ....
```

Jednym z takich atrybutów, niezwiązanym z konkretnym urządzeniem, jest atrybut *docpath* elementu *device*, wskazujący adres internetowy dokumentacji parametrów danego urządzenia, np.

```
<device daemon="/opt/szarp/bin/mbrtudmn" path="/dev/ttyA11"
  xmlns:extra="http://www.praterm.com.pl/SZARP/ipk-extra"
  extra:docpath="http://www.szarp.org/docs/szarp-plc-howto/html/heatmeter-kamstrup-66c.h
">
  ....
```

5.3. Tworzenie i edycja konfiguracji

5.3.1. Używanie konfiguracji w formacie IPK

Częścią IPK jest biblioteka, pozwalająca programom na bezpośredni dostęp do danych konfiguracyjnych, bez potrzeby parsowania pliku konfiguracyjnego IPK. Niestety, używanie tej biblioteki (albo też bezpośrednio pliku IPK) wymagałoby zmian we wszystkich programach wchodzących w skład systemu SZARP, co jest rozwiązaniem nieakceptowalnym.

Dlatego też IPK został skonstruowany tak, aby możliwa była wzajemna konwersja konfiguracji z formatu SZARP 2.1 na IPK i z powrotem. Do konwersji istnieją narzędzia wykonujące ją automatycznie, opisane w Sekcja 5.3.3 i Sekcja 5.3.4. Narzędzie te, co dokładniej jest opisane w tych rozdziałach, mogą sobie jednak nie poradzić z niektórymi konfiguracjami - wymagana jest czasami ręczna wstępna obróbka konfiguracji.

Kiedy mamy już możliwość konwersji na oba używane formaty, należy ustalić, który z nich będzie formatem głównym (tym, w którym będą nanoszone zmiany), a który tym generowanym automatycznie. Sugerowane jest, aby formatem głównym był IPK i po każdej zmianie w nim była generowana konfiguracja w starym formacie. *Jest to niezbędne dla konfiguracji wykorzystujących funkcje IPK niedostępne w starym formacie konfiguracji, a więc np. dla obsługi nowego formatu bazy Szarpa.*

W ten sposób programy nowe będą korzystały z biblioteki IPK, zaś stare z wygenerowanej konfiguracji w formacie SZARP 2.1.

5.3.2. Narzędzia do plików XML

Biblioteka IPK używa bibliotek libxml2 i libxslt (www.xmlsoft.org (<http://www.xmlsoft.org>)). Wraz z nimi udostępniane są dwa narzędzia:

- xmllint - procesor XML, pozwalający między innymi weryfikować poprawność dokumentu XML i jego zgodność z podanym schematem, formatować dokument a także zmieniać jego kodowanie. Przykłady użycia znajdują się w Sekcja 5.3.3 i Sekcja 5.3.5.
- xsltproc - procesor XSLT, pozwalający na przekształcanie dokumentów XML za pomocą szablonów XSLT. Przykłady użycia w Sekcja 5.3.6.

Należy zwrócić uwagę, że stworzone do manipulacji na konfiguracji szablony XSLT wchodzące w skład dystrybucji SZARP'a były testowane tylko z przy użyciu jako procesora XSLT programu xsltproc. Nie jest wykluczone, że zawierają one błędy, które mogą się ujawnić przy użyciu innego procesora.

Poza wymienionymi istnieje wiele innych narzędzi, w szczególności przeglądarek i edytorów XML, umożliwiających łatwiejszą pracę, zwłaszcza z dużymi konfiguracjami, gdy ważna jest możliwość graficznej prezentacji struktury pliku. Warto wiedzieć, że edytor XML jest wbudowany np. w przeglądarkę Mozilla (menu Tools -> Web development -> DOM inspector).

5.3.3. Konwersja konfiguracji SZARP 2.1 na IPK.

Programem służącym do konwersji konfiguracji z formatu SZARP 2.1 na IPK jest program szarp2ipk, znajdujący się w dystrybucji SZARP'a. Opis korzystania z programu można uzyskać uruchamiając go z opcjami '-h' lub '--help'. W szczególności program może przyjmować do dwóch parametrów - katalog w

którym ma szukać plików konfiguracyjnych SZARP 2.1 oraz ewentualnie także plik wyjściowy. Domyślnie przyjmowany jest odpowiednio katalog bieżący i plik `params.xml` w katalogu bieżącym. Tak więc w najprostszym przypadku należy zmienić katalog na katalog z konfiguracją i wywołać program bez parametrów - powstanie plik `params.xml` zawierający skonwertowaną konfigurację.

Program wypisuje na standardowe wyjście błędów komunikaty o napotkanych błędach i ostrzeżenia. Część z nich może dotyczyć nieobecności pewnych plików (co niekoniecznie musi oznaczać błąd - np. plik `sender.cfg` nie musi istnieć we wszystkich konfiguracjach).

Często zdarza się, że konfiguracja, która jest poprawna z punktu widzenia SZARP 2.1 powoduje błędy przy próbie konwersji na IPK. Można wyróżnić kilka podstawowych rodzajów błędów:

- Błędy składniowe w plikach - parsery używane przez bibliotekę IPK są w pewnych przypadkach bardziej restrykcyjne niż te w programach SZARP'a. Ponieważ program podaje miejsce wystąpienia błędu, powinien on być dość prosty do zlokalizowania i poprawienia - najczęściej chodzi o brak pewnych elementów (np. nazwy jednostki parametru), czy też o niezgodność liczby linii pliku z zadeklarowaną na jego początku (np. w raportach).
- Istnieją pewne rodzaje błędów, które program wykrywa, ale stara się naprawić. Są wśród nich takie, jak brak nazwy parametru (program przydziela nazwę typu "Unknown:Unknown:Unknown 1"), czy odwołanie w pliku `PTT.act` do nieistniejącego indeksu IPC (wtedy program stara się dodać pusty parametr definiowalny).
- Błędy typu podwójne wystąpienie parametru o jakiejś nazwie czy też wielokrotne zapisywanie (pod różną nazwą) tego samego parametru do kilku rekordów w bazie nie są poprawiane automatycznie i wymagają ręcznych poprawek przed konwersją.
- Najtrudniejsze do poprawienia są błędy wynikające z rekurencyjnych odwołań w strukturze parametrów definiowalnych. Program `parcook` przetwarza formuły w takiej kolejności, w jakiej są one zapisane w pliku `parcook.cfg`. Umożliwia to wielokrotne przedefiniowanie parametrów definiowalnych, w tym korzystanie z wcześniej wyliczonej wartości definiowanego właśnie parametru. Tego typu zależności nie da się zapisać w strukturze pliku IPK, gdyż wymagałoby to przechowywania dodatkowej informacji o kolejności formuł, oraz używanie wielu formuł dla jednego parametru. W związku z tym, jeżeli program wykryje tego typu błąd, przerywa pracę. Konwersja takiej konfiguracji wymaga ręcznego przepisania parametrów definiowalnych, tak aby nie korzystać z wielokrotnego przedefiniowania parametrów i wzajemnych odwołań do parametrów. Formuły należy pisać tak, aby nie trzeba było przyjmować żadnych założeń co do kolejności obliczania formuł (w praktyce będą wyliczane w kolejności indeksów IPC).

Wynika to ze struktury IPK, ale warto może w tym miejscu przypomnieć, że używane w SZARP 2.1 indeksy IPC w IPK nie są nigdzie bezpośrednio obecne, ale wynikają one z kolejności parametrów. (Biblioteka IPK pozwala na znalezienie parametru korzystając z jego numeru IPC, oraz obliczenie numeru IPC dla parametru - dla zachowanie kompatybilności wstecz.)

Wygenerowany przez program plik wyjściowy jest w kodowaniu ISO-8859-2. Jeżeli SZARP był skompilowany z użyciem starszej wersji biblioteki `libxml2` może się zdarzyć, że polskie znaki diakrytyczne będą podane za pomocą encji z kodami tych znaków. Aby uzyskać polskie znaki w "normalnym" zapisie, można przepuścić wygenerowany plik przez program `xmllint`:

```
# xmllint params.xml > params2.xml
```

Zobacz też Sekcja 5.3.2.

Uwaga! Zaleca się uruchomienie programu szarp2ipk tylko raz, przy pierwszej konwersji konfiguracji na IPK a następnie dokonywanie wszelkich modyfikacji na pliku IPK. Jest to konieczne np. dla obsługi nowego formatu bazy SZARP'a czy użycia dodatkowych opcji do demonów linii.

Program szarp2ipk nie wykonuje konwersji plików konfiguracyjnych programu analiza (analiza.cfg). Zadaniem tym zajmuje się narzędzie analiza2ipk. Analiza2ipk dołącza do istniejącego pliku IPK konfigurację programu analiza. Program analiza2ipk operuje bezpośrednio na dokumencie XML, co powoduje, że program nie modyfikuje oraz nie traci dodatkowych informacji np. komentarzy. Opis opcji programu można uzyskać uruchamiając go z opcją -h lub --help. Do istotniejszych parametrów przyjmowanych przez program należy nazwa katalogu z konfiguracją (katalog musi zawierać pliki: params.xml, analiza.cfg, PTT.act) oraz nazwa pliku docelowego. Gdy katalog z konfiguracją nie został podany program będzie szukał w/w plików w katalogu bieżącym. Domyślną nazwą pliku docelowego jest params.xml. Gdy plik o takiej samej nazwie jak nazwa pliku docelowego istnieje program odmówi jego nadpisania. Można usunąć to ograniczenie uruchamiając program z przełącznikiem -f lub --force. *Uruchomienie programu w katalogu z konfiguracją z podaną jedynie opcją -f lub --force spowoduje nadpisanie istniejącego pliku params.xml.*

5.3.4. Konwersja konfiguracji IPK na SZARP 2.1

Do konwersji konfiguracji z IPK na stary format służy program ipk2szarp. Podobnie jak szarp2ipk po podaniu opcji '-h' lub '--help' wypisuje informacje o swoim użyciu. I podobnie jako parametr można podać plik wejściowy (domyślnie ./params.xml) i katalog wyjściowy (domyślnie bieżący). Program jest zabezpieczony przed przypadkowym nadpisaniem konfiguracji z katalogu bieżącego - nadpisanie jakiegokolwiek pliku wymaga podania opcji '-f' ('--force'). Bez tej opcji program nie zniszczy żadnego istniejącego pliku.

Program stara się w przejrzysty sposób sygnalizować występujące błędy (jednak konwersja w tą stronę jest prostsza niż odwrotnie i nie powinna sprawiać problemów). Jednym z występujących problemów może być brak ciągłości w nadanych indeksach w bazie.

Automatyczne indeksy w bazie, obecne w IPK, nie są możliwe do reprezentacji w starym formacie konfiguracji. W związku z tym parametry z takimi indeksami w pliku PTT.act występują jakby nie były zapisywane do bazy. Z takiej konfiguracji nie wolno generować IPK, bo straci się informacje o zapisywaniu parametrów do bazy. *Przejsie na nowy format bazy i korzystanie z automatycznych indeksów oznacza więc konieczność definitywnego przejścia na IPK i wykorzystywania konfiguracji w starym formacie tylko dla kompatybilności ze starymi programami.*

W konfiguracjach z automatycznymi indeksami nieco inna jest też postać pliku definable.cfg - w pliku tym jako identyfikatory parametrów używane są indeksy w bazie. Ponieważ nie są one dostępne, zamiast nich generowane są indeksy IPC (kolejne numery parametrów w pamięci). Program przeglądający potrafi poprawnie zinterpretować te indeksy, posługując się informacją z IPK. Natomiast taki plik nie zostanie poprawnie wczytany przez program szarp2ipk, gdyż nie ma sposobu stwierdzenia, jaki typ indeksów jest użyty w definable.cfg. Jest to dodatkowy powód, dla którego konfiguracja z automatycznymi indeksami oznacza konieczność korzystania z IPK jako głównego formatu konfiguracji.

5.3.5. Weryfikacja poprawności konfiguracji

Po wprowadzeniu poprawek do pliku IPK dobrze jest sprawdzić jego poprawność. Można w tym celu

wykonać kilka kroków:

- Sprawdzić, czy nasz plik jest w ogóle poprawnym plikiem XML - najłatwiej przy użyciu programu `xmllint`:

```
# xmllint params.xml
```

Jeżeli zrobiliśmy jakiś błąd, zostanie wypisany odpowiedni komunikat.

- Kolejnym krokiem może być sprawdzenie, czy nasz plik jest zgodny ze schematem IPK. Ponownie można użyć programu `xmllint`:

```
# xmllint --relaxng /opt/szarp/resources/dtd/ipk-params.rng params.xml
```

Ten sam efekt można wywołać wywołując obecny w Szarpie skrypt `ipk_check_dtd`.

- Jeżeli poprzednie kroki się powiodły, warto spróbować wygenerować konfigurację SZARP 2.1 (zobacz Sekcja 5.3.3).

Poprawne przejście przez konfigurację wszystkich tych kroków daje dużą szansę na brak błędów w pliku.

5.3.6. Tworzenie i używanie szablonów

W IPK nie ma oddzielnego formatu dla szablonów konfiguracji. Przez szablon będziemy rozumieć dowolny plik XML (IPK), który może zostać użyty do dodania grupy parametrów do konfiguracji. Struktura IPK powoduje, że tak naprawdę dodanie grupy parametrów sprowadza się do tekstowego wklejenia ich w odpowiednie miejsce (zwykle dodaje się odpowiedni element lub elementy *device*, oraz parametry definiowalne). Problemem jest jedynie nadanie nowym parametrom odpowiednich indeksów w bazie.

Aby jednak ułatwić i zautomatyzować dodawanie parametrów przyjęto następujące rozwiązanie.

Szablonem IPK nazywamy plik, który od poprawnego pliku IPK opisującego interesujące nas parametry różni się w następujący sposób:

- Części nazw parametrów (i innych atrybutów), które są wspólne dla wszystkich parametrów w szablonie i powinny być zastąpione innymi ciągiem znaków przy konkretnym użyciu szablonu zastąpione są ciągiem pięciu znaków 'X'. Np. jeśli nasz szablon ma zawierać kompletny zestaw parametrów dla kotła, to zapewne wszystkie wystąpienia ciągu znaków typu 'Kocioł 1' będziemy chcieli zastąpić ciągiem X-ów, tak aby potem można było użyć szablonu i zmienić nazwy parametrów na zaczynające się od np. 'Kocioł WR 2'.

Odpowiednio zamienione nazwy powinny też wystąpić w tytułach raportów i wykresów, a także w formułach definiujących parametry.

- Parametry, które mają być zapisywane do bazy nie mają atrybutu *base_ind*, ale zamiast niego mają atrybut *tobase* o wartości 1. Oznacza on, że przy użyciu szablonu należy nadać parametrowi odpowiedni indeks w bazie. Nie dotyczy to indeksów automatycznych - można do szablonu bezpośrednio wstawić po prostu *base_ind="auto"*.

Oczywiście szablon powinien zawierać tylko informacje o interesujących nas parametrach. Przykładowy opis parametru w szablonie może wyglądać tak:

```
<param name="XXXXX:Sterownik:temperatura zadana" short_name="Tod"
draw_name="Temp. zadana" unit="°C" prec="1" tobase="1">
  <raport title="XXXXX" description="temperatura zadana"
filename="XXXXX.rap"/>
  <raport title="RAPORT TESTOWY" description="XXXXX Sterownik"
```

```
filename="test.rap"/>
    <draw title="XXXXX - temperatury" prior="81" color="cyan"
min="0" max="150"/>
    </param>
```

W dystrybucji SZARP'a znajduje się program `ipk_create_template`, który pozwala na wybranie z podanej konfiguracji parametrów zawierających podany ciąg znaków (np. 'Kocioł 1') i utworzenie z nich szablonu. Pierwszym parametrem jest szukany podciąg, drugim plik z konfiguracją. Wynik działania jest wypisywany na standardowe wyjście. Tak więc z danej konfiguracji szablon zawierający opis kotła można uzyskać przez wydanie następującej komendy:

```
/opt/szarp/bin/ipk_create_template 'Kocioł 1' params.xml > template.xml
```

Następnym krokiem jest przejrzanie pliku `template.xml` i sprawdzenie, czy na pewno umieszczone zostały te parametry, które chcemy umieścić w szablonie i czy ogólnie nie trzeba wprowadzić jakiś poprawek. W szczególności uwagi na temat ustalania kolejności wykresów znajdują się w Sekcja 5.3.8.

Kolejny program ułatwiający korzystanie z szablonów, to `ipk_add_template`. Służy do dodania parametrów opisanych przez szablon do konfiguracji. Automatycznie zamienia nazwy na podane (z XXXXX), nadaje także parametrom odpowiednie indeksy w bazie. W wywołaniu należy podać ścieżkę do szablonu z dodawanymi parametrami, ciąg znaków do utworzenia prawidłowych nazw parametrów, oraz nazwę konfiguracji. Jeżeli nie podamy tego ostatniego parametru, z szablonu zostanie utworzona nowa konfiguracja. Wynik działania programu jest wypisywany na standardowe wyjście.

Uwaga! Z szablonu wklejane są do konfiguracji całe elementy 'device', tak więc nie jest możliwe bezpośrednie dodanie parametrów do istniejącego sterownika. Można oczywiście dodać sztuczny nowy sterownik, a ręcznie przenieść parametry do istniejącego.

Jeżeli więc do konfiguracji z poprzedniego przykładu chcielibyśmy dodać czwarty kocioł, identyczny z pierwszym, mając już utworzony szablon wydajemy komendę:

```
/opt/szarp/bin/ipk_add_template template.xml 'Kocioł 4:' params.xml > new.xml
```

Plik `new.xml` będzie zawierał nową konfigurację. Oczywiście możliwe, że będziemy chcieli poddać ją edycji, np. aby dodać parametry dotyczące nowego kotła do wyliczeń ogólnej sprawności czy mocy ciepłowni.

Wszystkie wymienione w tym rozdziale programy do operacji na konfiguracji wykorzystują do swego działania procesor XSLT i odpowiednie szablony. Szablony te można także wykorzystywać samodzielnie, znajdują się one w dystrybucji SZARP'a w katalogu `libSzarp2/xslt`. W kodzie każdego z szablonów znajduje się dokładny opis działania oraz sposób podawania parametrów (dla procesora `xsltproc`, dla innych programów sposób podawania parametrów może być inny).

5.3.7. Samodzielne tworzenie i edytowanie konfiguracji IPK

Niniejszy rozdział zawiera kilka ogólnych uwag dotyczących edycji konfiguracji w formacie IPK.

- Co było sygnalizowane już w innych miejscach dokumentacji, należy starać się ograniczać ilość informacji opisanych w konfiguracji - w szczególności nie podawać wartości atrybutów tam, gdzie wystarczą wartości domyślne. Takie podejście upraszcza konfigurację i ułatwia jej edycję. Podobnemu celowi może służyć zastępowanie części nazw parametrów w formułach znakami gwiazdki.

- Usuwanie parametrów sprowadza się tylko usunięcia odpowiedniego elementu *param*. Należy jednak sprawdzić, czy przypadkiem parametr nie pojawia się w którejś z formuł. Najlepiej przeszukać plik pod kątem występowania ostatniego członu nazwy parametru (bo w formułach może być używana skrócona nazwa parametru, z gwiazdkami zamiast powtarzającego się fragmentu nazwy). Inna sprawa, że struktura bazy w SZARP 2.1 nie umożliwia usuwania parametrów, wobec tego, jeśli parametr był zapisywany do bazy (i jest używana stary format bazy), można jedynie usunąć dotyczące go raporty i wykresy.
- Przy dodawaniu parametrów należy zwrócić uwagę na nadanie odpowiedniego (kolejnego) indeksu w bazie. Największy występujący w konfiguracji indeks wypisuje wchodzący w skład dystrybucji SZARP'a program `ipk_base_max`. Program wczytuje albo konfigurację podana jako parametr, albo ze standardowego wejścia. Jak większość narzędzi do operacji na IPK program jest prostym skryptem korzystającym z szablonów XSLT.

Prostszym rozwiązaniem może być wykorzystanie mechanizmów używanych także przy pracy z szablonami - zamiast atrybutów `'base_ind'` dajemy nowym parametrom atrybuty `'tobase'` o wartości `'1'` (a więc w obrębie elementu `'param'` dodajemy wpis `'tobase="1"'`). Następnie na tak przygotowanym pliku uruchamiamy skrypt `ipk_set_base`, który ustawi odpowiednie, kolejne indeksy w bazie.

Problem ten nie występuje przy korzystaniu z nowego formatu bazy - wystarczy podać jako wartość indeksu `"auto"`.

5.3.8. Określanie kolejności wykresów i raportów.

Jeżeli nie nadamy atrybutów *prior* wykresom, kolejność okien w programie przeglądającym będzie zależna od kolejności (indeksu IPC) pierwszego widocznego w oknie parametru. Za pomocą wspomnianego atrybutu możemy tą kolejność zmieniać.

Kolejność wystąpienia okna ostatecznie określane jest przez najmniejszy atrybut spośród wszystkich wykresów występujących w oknie. Może to łatwo spowodować pewien chaos i trudności z późniejszym układaniem okien, jeżeli będziemy mieli dużo atrybutów *prior*. Dlatego zaleca się nadawanie atrybutu tylko jednemu (np. pierwszemu) wykresowi w oknie. Dzięki temu zmiana kolejności będzie wymagała zmiany tylko jednego atrybutu.

Zagadnieniem mogącym sprawiać też problemy jest ustalanie kolejności wykresów przy korzystaniu z szablonów. Polecanym rozwiązaniem jest edycja szablonu tak, aby wszystkie wykresy miały priorytety ustalone jako liczby z zakresu od 1 do 2, czyli np. 1.10, 1.23 itp. Część ułamkowa określa kolejność wykresów wewnątrz szablonu. Następnie, przed dodaniem szablonu do konfiguracji należy zwiększyć wszystkie priorytety, tak aby ustalić ich położenie względem innych parametrów już obecnych w konfiguracji. Pomocny w tym może być szablon XSLT `move_draw.xsl`. Pozwala on wybrać z konfiguracji (lub szablonu) odpowiednie wykresy i zwiększyć lub zmniejszyć ich priorytet. Poniższy przykład zwiększa o 12 priorytet wszystkim wykresom, które w tytule (atrybut *title*) zawierają ciąg znaków `'Kocioł 1'`:

```
xsltproc --stringparam title 'Kocioł 1' --stringparam num 12 \
/opt/szarp/resources/xslt/move_draw.xsl params.xml > new.xml
```

Szablon ułatwia więc zmienianie priorytetu dla całych grup wykresów. Jako parametr `'num'` można też oczywiście podać liczbę ujemną, co powoduje zmniejszenie atrybutu *prior* wybranych wykresów. Jako parametr `'title'` można podać pusty string `"` - wówczas wszystkie wykresy będą miały zmieniony atrybut *prior*.

W dystrybucji SZARP'a znajduje się także program `ipk_move_draws`, wywołujący podany wyżej szablon, przez co taki sam efekt można uzyskać wywołując ten program: `/opt/szarp/bin/ipk_move_draws 'Kocioł 1' 12 params.xml > new.xml`

Innym rozwiązaniem pozwalającym na proste ustalanie kolejności zarówno okien programu przeglądającego jak i samych wykresów oraz raportów jest użycie programu `ipkedit`, opisanego w rozdziale Sekcja 5.3.9.

5.3.9. Program Edytor IPK

Program `ipkedit` ułatwia najbardziej żmudne prace przy edycji konfiguracyjnej SZARP w formacie IPK. Pozwala na zmianę kolejności parametrów w raportach, a także zestawów wykresów i kolejności wykresów w ramach zestawów. Pozwala także na edycję zakresów osi wykresów i ich kolorów.

Program nie wykorzystuje biblioteki IPK, ale operuje bezpośrednio na danych XML, modyfikując jedynie wybrane atrybuty odpowiednich elementów. Nie usuwa więc żadnych dodatkowych informacji z konfiguracji, takich jak np. elementy z innych przestrzeni nazw czy komentarze.

`ipkedit` jest aplikacją graficzną, korzystającą z biblioteki `wxWindows`. Interfejs programu jest bardzo prosty. Udostępnia podstawowe operacje plikowe oraz możliwość zmiany kolejności zestawów, wykresów i raportów oraz prosty edytor właściwości wykresów.

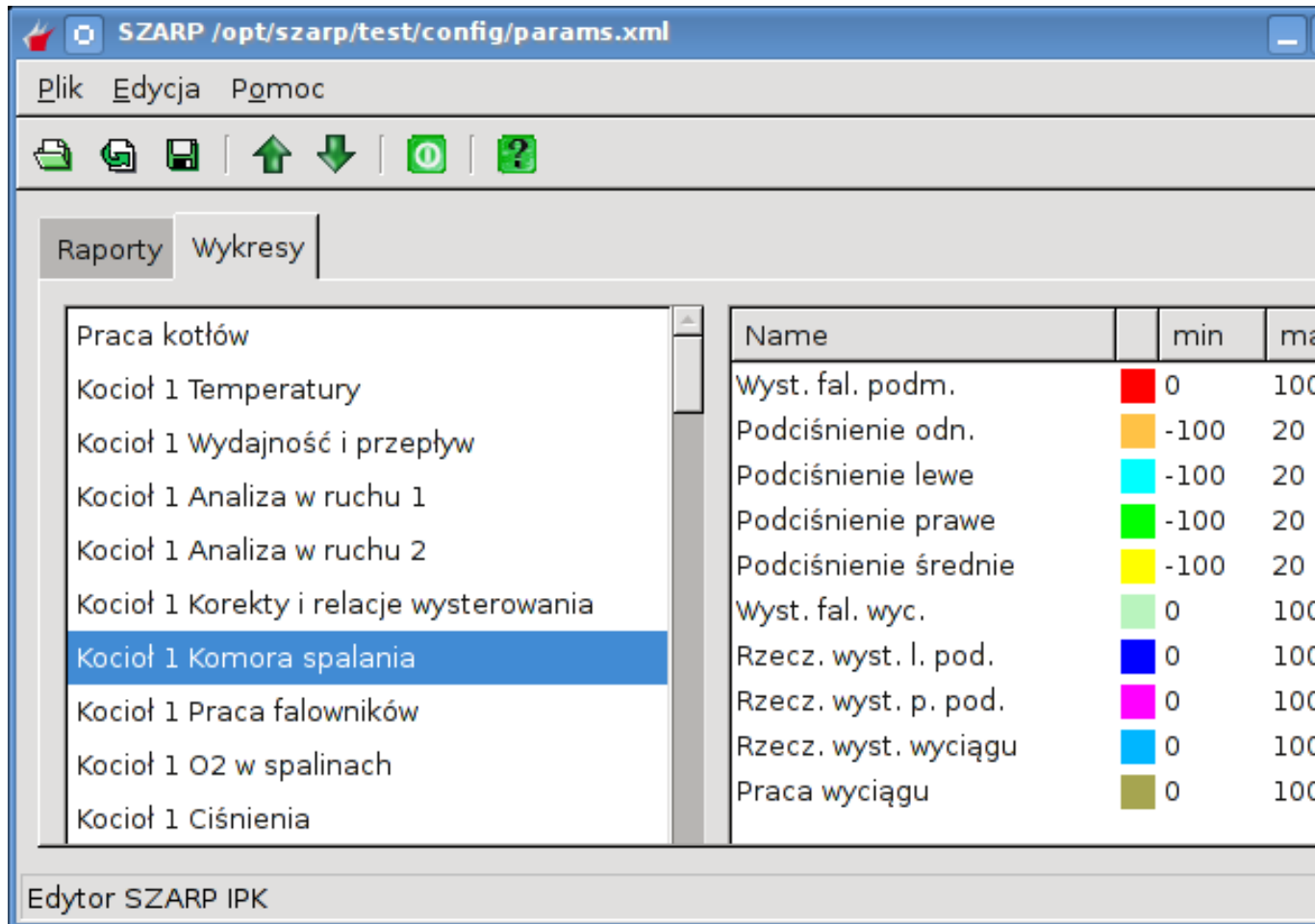
Centralną część okna programu zajmują dwie zakładki, zawierające zgodnie z nazwami odpowiednio listę raportów i ich elementów, oraz listę okien i wykresów. Większość funkcji programu dostępna jest z umieszczonej na górze okna paska ikon. Są to odpowiednio:

- Załadowanie pliku z dysku. Plik jest weryfikowany pod kątem poprawności, tzn. musi być dokumentem XML zgodnym ze schematem RelaxNG IPK. Plik do otwarcia można także podać jako parametr programu.
- Ponowne załadowanie pliku (przydatne gdy chcemy załadować konfigurację po zmianie wykonanej w edytorze).
- Zapisanie pliku na dysku.
- Przesunięcie obiektu w górę listy. Jeżeli mamy zaznaczony parametr w raporcie, przesuwanie dotyczy tego parametru, jeżeli wykres, to wykresu. Jeżeli nie zaznaczyliśmy wykresu, a mamy zaznaczone okno przeglądającego, przesuwane jest okno. Nie można przesuwać raportów. Przesuwanie powoduje jedynie zmianę atrybutów *prior* lub *order* obiektów, bez żadnych dodatkowych ingerencji w dokument.
- Przesunięcie obiektu w dół listy - patrz uwagi wyżej.
- Wyjście z programu.
- Wyświetlenie dokumentacji programu.

Wszystkie powyżej opisane funkcje dotyczące operacji na plikach mają swoje odpowiedniki w menu *Plik*, oraz towarzyszące pozycjom menu skróty klawiszowe. Dodatkowo w menu jest dostępna funkcja *'Zapisz jako'*, pozwalająca zapisać plik pod zmienioną nazwą.

Dodatkowo po wyświetleniu listy wykresów w zestawie i dwukrotnym kliknięciu wykresu (lub naciśnięciu **Enter**) otwiera się okienko pozwalające na zmianę zakresów osi oraz koloru dla wykresów. Wybranie koloru „automatycznego” powoduje przydzielenie pierwszego wolnego koloru na podstawie kolejności wykresów. Tak przydzielone kolory zaznaczone są na liście wykresów przez szarą obwódkę wokół kwadratu z kolorem.

W menu *Edycja* poza poleceniami do przesuwania obiektów w górę i w dół listy znajduje nie dodatkowe polecenie *Wyczyść atrybuty*. Usuwa ono z całego pliku wszystkie atrybuty *order* i *prior* wykresów i raportów. Po wydaniu polecenia kolejność obiektów jest zależna od kolejności wystąpienia w konfiguracji IPK.



Rysunek 1. Przykładowy wygląd okna programu

Jeżeli dana operacja (np. otwarcie pliku) może spowodować utratę wprowadzonych zmian, program wyświetla okienko pozwalające na zapisanie lub zignorowanie zmian, oraz na rezygnację z wykonywanej operacji.

5.3.10. Spis narzędzi do IPK

Poniżej znajduje się lista wchodzących w skład dystrybucji SZARP'a narzędzi pomocnych w pracy z IPK. Wszystkie te narzędzia obsługują opcję `'-h'` (lub `'--help'`), po podaniu której wyświetlają dokładny opis opcji i parametrów programu. Stąd też poniżej znajdują się tylko ogólne opisy przeznaczenia poszczególnych narzędzi:

- `ipk2szarp` - konwersja konfiguracji w formacie IPK na 'stary' format SZARP 2.1.

- *i2smo* - (*ipk2szarp* modified only) nakładka na *ipk2szarp* tworząca pliki wynikowe w tymczasowym katalogu i kopiująca do katalogu docelowego tylko pliki rzeczywiście zmienione. Dzięki temu ograniczona jest ilość zmian wymuszających restart SZARP'a. Program przyjmuje takie same opcje jak *ipk2szarp*, z tym że ignorowany jest brak opcji '-f' - zmienione pliki są zawsze nadpisywane.
- *ipk_add_template* - dodanie szablonu do konfiguracji lub stworzenie nowej konfiguracji na podstawie szablonu IPK.
- *ipk_base_max* - wypisuje największy występujący w konfiguracji indeks parametru w bazie.
- *ipk_check_dtd* - sprawdza zgodność poprawność konfiguracji pod kątem zgodności ze schematem RelaxNG (nazwa ze względów historycznych - wcześniej używano zamiast RelaxNG DTD).
- *ipk_create_template* - wybiera z konfiguracji parametry na podstawie fragmentu nazwy i tworzy z nich szablon IPK.
- *ipkedit* - program z interfejsem graficznym służący do zmiany kolejności okien programu przeglądającego, wykresów i parametrów w raportach.
- *ipk_get_base* - wypisuje indeks w bazie parametru o podanej nazwie.
- *ipkmax* - jak *ipk_base_max*, tylko nie jest to skrypt korzystający z szablonów XSLT, ale program napisany z wykorzystaniem biblioteki IPK - napisany był jako program przykładowy i testujący bibliotekę.
- *ipk_move_draws* - zwiększa lub zmniejsza o zadaną liczbę priorytet okien programu przeglądającego dla parametrów wybranych na podstawie fragmentu nazwy.
- *ipk_normal_draws* - "normalizuje" priorytety okien programu przeglądającego w pliku wejściowym, przekształcając je do postaci '1.X'.
- *ipk_remove_filenames* - usuwa atrybuty 'filename' z opisu raportu - zmniejsza to rozmiar i zwiększa czytelność konfiguracji. Używane są nazwy plików tworzone na podstawie tytułu raportu.
- *ipk_set_base* - wyszukuje w konfiguracji wszystkie atrybuty 'tobase' parametrów i zamienia je na odpowiednie (kolejne) indeksy w bazie. Dzięki temu przy ręcznym dodawaniu parametrów nie trzeba samodzielnie sprawdzać największego obecnego indeksu w bazie.
- *szarp2ipk* - tworzenie konfiguracji IPK na podstawie konfiguracji w formacie SZARP 2.1.

5.4. IPK dla programistów

5.4.1. Interfejs biblioteki IPK

Częścią projektu IPK jest biblioteka, ułatwiająca korzystanie z konfiguracji programom użytkowym. Dzięki temu nie jest konieczne samodzielne parsowanie konfiguracji czy dokonywanie różnych przeliczeń.

Ze względu na dość sporą złożoność, oraz aby umożliwić łatwiejszą rozbudowę i przyszłe modyfikacje, biblioteka została napisana nie w C, ale w C++.

Dokładny opis wszystkich klas można otrzymać przez wygenerowanie dokumentacji z kodu źródłowego. W tym celu w repozytorium SZARP'a w podkatalogu `libSzarp2` należy wydać komendę:

```
make doxy
```

W katalogu `doxy` zostanie wygenerowana dokumentacja, wersja HTML w podkatalogu `html`.

Struktura biblioteki odpowiada mniej więcej strukturze pliku IPK (zobacz Sekcja 5.2). Konfiguracja przechowywana jest w jednym głównym obiekcie typu *TSzarpConfig*. Obiekt ten zawiera listę elementów typu *TDevice*, opisujących poszczególne linie komunikacyjne, oraz listę parametrów definiowalnych. Obiekty klasy *TDevice* zawierają zaś wskaźniki do opisywanych linii radiowych, jednostek itd.

Przyjęto, zgodnie z metodologią obiektową, zasadę ukrywania implementacji poszczególnych elementów. Wszystkie atrybuty są chronione lub prywatne, jeżeli jest potrzeba dostępu do nich, definiowane są odpowiednie metody.

Należy zwrócić uwagę, że API biblioteki jest jeszcze na dość wstępnym etapie rozwoju. W miarę powstawania korzystających z biblioteki aplikacji będą się pojawiały nowe elementy. Mam nadzieję, że obiektowa struktura biblioteki pozwoli zachować kompatybilność wstecz.

W większości wypadków implementacja struktur danych wewnątrz biblioteki jest listowa. Dzięki temu możliwe jest łatwe dodawanie i usuwanie elementów (choć nie wszędzie są w tej chwili napisane odpowiednie interfejsy). Minusem jest liniowy czas dostępu do parametrów (np. wyszukiwanie parametrów według jakiś kryteriów). Niektóre algorytmy (np. tworzenie pliku `eknrCOR`) mają złożoność kwadratową względem liczby parametrów. Jeżeli w jakiś aplikacjach będzie to powodowało problemy wydajnościowe, trzeba będzie zaimplementować jakieś bardziej ambitne algorytmy.

Najprawdopodobniej jednak, przy typowych rozmiarach konfiguracji (maksymalnie tysiące parametrów) i obecnych mocach obliczeniowych aktualna prosta implementacja okaże się zupełnie wystarczająca.

Biblioteka nie przechowuje w tej chwili żadnych dodatkowych informacji pomocniczych, tak więc modyfikacje konfiguracji z poziomu biblioteki, podobnie jak operacje na pliku IPK, nie wymagają uaktualniania powiązanych informacji.

Do parsowania XML-a wykorzystywana jest biblioteka `libxml2`. Biblioteka ta przechowuje wszystkie dane tekstowe (także atrybuty) w kodowaniu UTF-8. Natomiast interfejs biblioteki IPK wykorzystuje kodowanie ISO-8859-2 (dokonując w razie potrzeby odpowiednich konwersji przy wywołaniach funkcji z biblioteki XML). Stąd też wszystkie parametry do funkcji biblioteki IPK powinny być podawane w tym kodowaniu.

Konstruktor obiektu przechowującego konfigurację jest bezparametrowy. Inicjalizacja następuje przez wczytanie informacji z pliku. Funkcja `TSzarpConfig::loadSzarpConfig(char* directory)` wczytuje konfigurację w formacie SZARP 2.1 z podanego katalogu. Należy zwrócić uwagę, że część z wykorzystywanych przez funkcję parserów plików konfiguracyjnych (napisanych we fleksie) nie jest 'reentrant' (jakiś polski odpowiednik?) i może być wykorzystywana tylko raz. Ogólnie więc funkcję tą można w programie wywołać tylko raz. Lepiej jest więc wczytywać konfigurację w formacie IPK, przy użyciu metody `TSzarpConfig::loadXML(char* path)`.

Oto przykładowy program korzystający z biblioteki. Wypisuje on indeks IPC parametru o nazwie podanej jako drugi parametr (pierwszym jest nazwa pliku konfiguracyjnego):

```
#include <iostream>
#include <string>
using namespace std;

#include "szarp_config.h"

int main(int argc, char* argv[])
{
    TSzarpConfig * config;
    TParam * param;

    if (argc < 2) {
```

```

cout << "Za mało parametrów\n" ;
return 1;
}

config = new TSzarpConfig();

if (config->loadXML(argv[1]) != 0) {
cout << "Błąd podczas wczytywania pliku " << argv[1] << "\n";
return 1;
}

param = config->getParamByName(argv[2]);

if (param == NULL) {
cout << "Nie znaleziono parametru '" << argv[2] << "'\n";
} else {
cout << "Indeks IPC parametru to " << param->GetIpcInd() << "\n";
}

delete config;
}

```

Kompilację i linkowanie programu można przeprowadzić za pomocą komend:

```

gcc -Iszarp/libSzarp2/include/ `xml2-config --cflags` -c -Wall getipc.cc
g++ -o getipc -L/usr/X11R6/lib -lXt szarp/libSzarp/libSzarp.so \
szarp/libSzarp2/libSzarp2.so -lxml2 -lz -lm getipc.o

```

Biblioteka była testowana za pomocą debuggerów pamięci. Polecanym przeze mnie programem jest valgrind, dostępny pod adresem <http://freshmeat.net/projects/valgrind>. Jest bardzo prosty w użyciu i pozwala na wykrycie błędów, których zlokalizowanie w inny sposób byłoby prawie niemożliwe.

5.4.2. Korzystanie z parserów konfiguracji SZARP 2.1

Biblioteka IPK musi umieć parsować wszystkie pliki konfiguracyjne SZARP 2.1. Z różnych względów nie chciałem korzystać z istniejących parserów i do wszystkich napisałem własne, korzystające z programu flex. Dzięki temu parsery mają większą szansę na brak błędów (bo kod jest generowany maszynowo przez fleksa), a poza tym są dużo łatwiejsze do modyfikacji.

Parsery można oczywiście wykorzystać niezależnie od reszty biblioteki, gdyby była potrzeba pisania programu, który z jakiś względów musi parsować któryś z plików konfiguracyjnych SZARP 2.1. Interfejsy do parserów są zawarte w plikach w katalogu `libSzarp2/include`, są to odpowiednio:

- `ekrncor.h` - parser plików z opisem okien dla programu przeglądającego (`ekrnXXXX.cor` i `ekrnXXXX.def`).
- `parcook_cfg.h` - parser plików `parcook.cfg`.
- `sender_cfg.h` - parser plików konfiguracyjnych programu `sender` (`sender.cfg`).
- `ptt_act.h` - parser plików `PTT.act`.
- `line_cfg.h` - parser plików z opisami linii dla demonów linii (`lineX.cfg`).

- `definable_parser.h` - parser plików z parametrami definiowalnymi (`definable.cfg`), napisany w C++.
- `raporter.h` - parser plików konfiguracyjnych raporta, także napisany w C++.

Wszystkie parsery napisane w C, poza `line_cfg.h` (a więc pierwsze cztery) mają tę wadę, że nie będą poprawnie działały jeżeli w czasie jednego przebiegu programu zostaną wywołane kolejny raz (flex zapamiętuje swój stan globalnie i potem nie chce to za bardzo działać). Jeżeli będzie to problemem, trzeba będzie zmienić interfejs parserów na C++. Docelowo jednak jedynym programem korzystającym z IPK i wczytującym konfigurację w formacie SZARP 2.1 powinien być `szarp2ipk`, więc raczej nie będzie to konieczne.

Kolejną wadą (choć można na to różnie patrzeć) parserów jest ich większa restrykcyjność niż parserów używanych w aplikacjach SZARP 2.1, które potrafią sparsować także pliki nie do końca zgodne z deklarowanym formatem. Jest to jednak także raczej problem okresu przejściowego, dopóki wszystkie konfiguracje nie zaczną używać IPK. Poza tym są to zwykle problemy, które łatwo poprawić.

5.4.3. Rozszerzanie IPK

W przyszłości zapewne IPK będzie obsługiwać więcej danych o konfiguracji niż obecnie. Nie każde jednak powstanie nowego programu musi się wiązać z rozszerzaniem IPK.

W niektórych przypadkach celowe wydaje się utrzymywanie oddzielnych plików konfiguracyjnych dla jakiś programów. Jest tak np. w przypadku dyspozytora. Plik konfiguracyjny tego programu nie opisuje właściwości parametrów samych w sobie, poza tym jest specyficzny dla użytkownika. Przystosowaniem programu do współpracy z IPK mogłoby być natomiast używanie jako identyfikatorów parametrów ich nazw, zamiast indeksów w bazie czy pamięci dzielonej. Uniezależnia to taki program od większości zmian w konfiguracji bazowej, trzymanej w IPK (najgorsze co może się stać to usunięcie jakiegoś parametru lub zmiana nazwy - program straci wtedy dostęp do tego parametru).

Inne programy (np. analiza) wykorzystują jednak raczej właściwości parametrów samych w sobie i w związku z tym być może korzystne byłoby umieszczenie informacji o roli parametru w analizie w IPK, np. przez dodanie elementu-dziecka do elementu *param*. W skrócie działania konieczne do dodania nowych danych do IPK przedstawiają się następująco:

- Zaprojektowanie nowego elementu i dodanie go do schematu RelaxNG. Oczywiście powinno to być zrobione tak, żeby zachować kompatybilność wstecz.
- Dodać strukturę opisującą nowe dane do odpowiedniej z klas opisujących strukturę IPK (tak aby zachować odpowiedniość ze strukturą pliku IPK). Potrzebne są podstawowe metody umożliwiające dostęp do tych danych, ich modyfikacje i tworzenie. Należy przestrzegać zasady nie wprowadzania atrybutów publicznych. Nowa klasa powinna zostać umieszczona w oddzielnym pliku źródłowym. Każda klasa, metoda i atrybut powinny być opisane zgodnie z wymaganiami Doxygena.
- Do funkcji wczytujących i zapisujących XML dodać parsowanie i tworzenie odpowiednich elementów drzewa XML.
- Jeżeli wprowadzane do IPK dane były już obecne w jakimś pliku konfiguracyjnym SZARP 2.1 należy napisać parser wczytujący te dane i metodę tworzącą ten plik, a następnie dodać wywołania tych funkcji do metod odczytujących i zapisujących konfigurację SZARP 2.1 (`TSzarpConfig::loadSzarpConfig()` i `TSzarpConfig::saveSzarpConfig()`).

5.5. Łączenie wielu konfiguracji w jedną

5.5.1. Koncepcja

Mechanizm łączenia wielu konfiguracji w jedną pozwala na prezentowanie w programie przeglądającym SzarpDraw wykresów z kilku oddzielnych baz na raz, a także na tworzenie parametrów definiowalnych, których formuły odwołują się do parametrów z różnych baz.

Notatka: Obecnie system SZARP umożliwia definiowanie parametrów odwołujących się do parametrów z innych baz oraz wyświetlanie w jednym zestawie wykresów z różnych baz. Stąd też konfiguracje agregowane nie są już niezbędne i nie mają takiego znaczenia.

Konfiguracje składowe muszą być w formacie IPK, a bazy muszą być w formacie SzarpBase (Sekcja 8). W wyniku złożenia powstaje nowa konfiguracja, której plik `params.xml` zawiera odpowiednio przekształcone wszystkie bądź wybrane parametry z konfiguracji składowych. Nowo powstała baza zawiera natomiast linki symboliczne do odpowiednich katalogów w bazach składowych. Dzięki temu w miarę pojawiania się danych w bazach składowych, będą one widoczne w konfiguracji łączonej.

Konfiguracja łączona może być normalnie przesyłana mechanizmem bodas, może też być tworzona bezpośrednio na komputerach, na których ma być oglądana.

5.5.2. Tworzenie i uaktualnianie konfiguracji

Konfiguracja jest tworzona przez program agregator instalowany domyślnie w katalogu `/opt/szarp/bin`. Program przyjmuje jeden parametr - ścieżkę do pliku konfiguracyjnego, którego format opisany jest w następnym rozdziale. Jeżeli program napotka błąd, wypisze o nim komunikat i zakończy działanie z kodem 1. Jeżeli wszystko pójdzie dobrze, program kończy działanie z kodem 0, nie wypisując żadnego komunikatu.

Plik konfiguracyjny opisuje konfiguracje które mają wejść w skład konfiguracji łączonej, przekształcenia jakim mają być poddane parametry z nich, oraz ewentualne parametry definiowalne, korzystające z danych z konfiguracji składowych. Plik zawiera też informację o prefiksie wynikowej konfiguracji. W wyniku działania programu powstaje plik `params.xml` w katalogu `/opt/szarp/<prefix>/config` (który musi istnieć) oraz odpowiedni katalog `/opt/szarp/<prefix>/szbase` zawierający strukturę bazy danych (`<prefix>` to prefiks konfiguracji ustalony w pliku konfiguracyjnym).

Notatka: Zalecaną konwencją jest wybieranie prefiksu czteroznakowego, kończącego się dużą literą 'X'. Np. w przypadku tworzenia konfiguracji obejmujących ciepłownię i węzły z Zamościa wybrany prefiks to `zamX`.

W związku z tym operacje potrzebne do stworzenia konfiguracji łączonej sprowadzają się do:

- Stworzenia katalogu `/opt/szarp/<prefix>/config`.
- Stworzenia i umieszczenia, najlepiej w utworzonym powyżej katalogu, pliku konfiguracyjnego

Notatka: Zalecaną konwencją jest nazywanie pliku konfiguracyjnego `aggr.xml`.

- Uruchomienia programu, podając jako parametr plik konfiguracyjny, np.:

```
/opt/szarp/bin/agregator /opt/szarp/zamX/config/aggr.xml
```

Notatka: Do uruchomienia programu potrzebne są prawa do zapisu do wynikowych katalogów - zwykle oznacza to uruchamianie programu po prostu przez użytkownika root lub korzystania z mechanizmu sudo.

Program agregator będzie musiał być ponownie uruchamiany po każdej istotniejszej zmianie w którejkolwiek z konfiguracji składowych - tak, aby mógł uaktualnić informację o parametrach i linki w bazie.

Notatka: Program tworzy tylko konfigurację w formacie IPK - jeżeli chcemy stworzyć lub uaktualnić konfigurację w formacie SZARP 2.1, musimy wywołać program ipk2szarp (zobacz Sekcja 5.3.4).

5.5.3. Plik konfiguracyjny

Plik konfiguracyjny musi być poprawnym dokumentem XML. Oto przykładowa zawartość:

```
<?xml version="1.0" encoding="ISO-8859-2"?>

<aggregate xmlns="http://www.praterm.com.pl/IPK/aggregate"
xmlns:aggr="http://www.praterm.com.pl/IPK/aggregate"
xmlns:ipk="http://www.praterm.com.pl/SZARP/ipk"
prefix="zamX"
title="Zamość">

<config prefix="zamo">
<remove xpath="//ipk:raport"/>
<regexp xpath="//ipk:draw/@prior">s#.*#10\0#</regexp>
<attribute xpath="//ipk:draw[not (@prior)]" name="prior"
value="099"/>
</config>

<config prefix="zmk1">
<remove xpath="//ipk:raport"/>
<regexp xpath="//ipk:draw/@prior">s#.*#20\0#</regexp>
</config>

<config prefix="zmw3">
<remove xpath="//ipk:raport"/>
<remove xpath="//ipk:draw/@prior"/>
<!--<regexp xpath="//ipk:draw/@prior">s#.*#30\0#</regexp>-->
</config>

<ipk:drawdefinable>
```

```

<ipk:param name="Test:Bez sensu: sumaryczna temp. zewnętrzna"
short_name="x" prec="1" unit="°C">
<ipk:define type="DRAWDEFINABLE"
formula="(Sieć:Sterownik:temperatura zewnętrzna) ({zmk1} Komora KM-1:Komora:temperatur
({zmk3} Komora KW-1:temperatura zewnętrzna) +" />
<ipk:draw title="TEST" min="0" max = "200" />
</ipk:param>
</ipk:drawdefinable>

</aggregate>

```

Główny element to *aggregate*. Powinien on zawierać deklaracje przestrzeni nazw

<http://www.praterm.com.pl/IPK/aggregate> (aggr) i

<http://www.praterm.com.pl/SZARP/ipk> (ipk). Zaleca się, aby ta pierwsza była przestrzenią domyślną, dzięki czemu nie trzeba będzie się do niej odwoływać w każdym elemencie.

Wymagane są także dwa atrybuty - *title* zawiera opisowy tytuł konfiguracji (wyświetlany w programie przeglądającym), natomiast *prefix* decyduje o nazwie i położeniu tworzonej konfiguracji wynikowej, w sposób opisany w poprzednim rozdziale.

Najważniejszą składową pliku są elementy *config*. Każdy z nich opisuje jedną konfigurację składową, o nazwie określanej przez atrybut *prefix*.

Pierwszy element *config* zawiera opis tak zwanej konfiguracji podstawowej. Parametry z tej konfiguracji nie będą podlegały przekształcaniu nazw. Pierwszy parametr z tej konfiguracji powinien być zapisywany do bazy (jest to wymaganie wynikające ze sposobu działania program SzarpDraw). Parametry statusu całej konfiguracji łączonej będą kopiowane także z niej. W związku z tym warto wybrać na konfigurację podstawową konfigurację największą, o największej pewności prawidłowej transmisji danych w ramach systemu BODAS.

Obecność elementu *config* z danym prefiksem oznacza włączenie do konfiguracji wszystkich parametrów z konfiguracji opisanej prefiksem. Nazwy parametrów (poza konfiguracją podstawową, pierwszą) są przekształcane do postaci *{prefix} nazwa*, np. parametr KM-1:Komora:Przepływ CO z konfiguracji o prefiksie zmk1 po konwersji będzie miał nazwę {zmk1} KM-1:Komora:Przepływ CO.

Jednak *przed* włączeniem i zmianą nazw parametry mogą być poddane przekształceniom opisanym przez elementy składowe elementu *config* - *remove*, *regexp* i *attribute*. Wszystkie te elementy mają wymagany atrybut *xpath*, zawierający wyrażenie XPath opisujące elementy, które mają być poddane przekształceniom. Wyrażenia XPath mają dostęp do przestrzeni nazw IPK za pomocą prefiksu *ipk*.

Notatka: XPath jest językiem do wybierania fragmentów dokumentu XML, mającym status standardu W3C. Oficjalna strona to <http://www.w3.org/TR/xpath>. W sieci można znaleźć wiele przykładów i samouczków. Oto kilka podstawowych przykładów:

- `//ipk:param` - wszystkie elementy param z dokumentu.
- `//ipk:drawdefinable/ipk:param` - wszystkie elementy param będące dziećmi elementu `drawdefinable`.
- `///ipk:device//ipk:param` - wszystkie elementy param mające jako przodka (niekoniecznie bezpośredniego) element `device`.
- `//ipk:param[not(/ipk:define)]` - wszystkie elementy param nie mające elementu potomnego `define`.
- `//ipk:param/ipk:draw/@title` - atrybuty `title` wszystkich elementów `draw` będących dziećmi elementów `param`.

- `//ipk:draw[@prior]` - elementy draw posiadające atrybut `prior`.
- `//ipk:draw[contains(@title,'Komora')]` - elementy draw, których atrybut `title` zawiera napis „Komora”.
- `//ipk:param/ipk:define[@type='DRAWDEFINABLE']/@formula` - atrybuty `formula` wszystkich elementów `define`, które mają atrybut `type` równy `DRAWDEFINABLE` i są dziećmi elementów `param`

Możliwe są trzy typy przekształceń:

- Element *remove* powoduje usunięcie elementów wybranych elementów XML. Przykładowo, zwykle będziemy chcieli usunąć elementy *raport*, których i tak nie będziemy wykorzystywać:

```
<remove xpath="//ipk:raport"/>
```

Możemy też usunąć wybrane atrybuty:

```
<remove xpath="//ipk:draw/@prior"/>
```

- Element *regexp* pozwala na przekształcenie zawartości elementu za pomocą wyrażenia regularnego (w praktyce odnosi się to tylko do zawartości atrybutów, gdyż elementy tekstowe i tak są usuwane i przeformatowywane w wynikowym dokumencie). Wyrażenie regularne podawane jest jako zawartość elementu i ma postać znaną z edytora vim lub programu sed - znak `'s'`, potem znak oddzielający, a następnie wzorce dopasowujący i zastępujący, rozdzielone i zakończone znakiem oddzielającym. We wzorcu zastępującym możemy odwoływać się do całego wyrażenia (za pomocą `\0`) lub poszczególnych podwyrażeń (np. `\1`). Przykładowo

```
<regexp xpath="//ipk:draw/@title">s#Komora(.*)#KM-1 \1#</regexp>
```

spowoduje zastąpienie w atrybutach *title* początkowego napisu 'Komora' napisem 'KM-1'. Identyczny efekt może być uzyskany przez nieco prostsze wyrażenie regularne:

```
<regexp xpath="//ipk:draw/@title">s#^Komora#KM-1#</regexp>
```

Notatka: Program używa biblioteki GNU Regex i rozszerzonych wyrażeń POSIX. W starszych wersjach programu (wersje SZARP'a do 2.2.1294 włącznie) używana była biblioteka POSIX Regex i aby dokonać podstawienia cały napis (np. wartość atrybutu) musiał być dopasowany do wyrażenia. Obecnie nie jest to wymagane - podstawienie jest wykonywane na pierwszym znalezionym najdłuższym możliwym podciągu pasującym do zadanego wyrażenia.

- Element *attribute* pozwala na dodanie do elementu atrybutu o danej nazwie. Jeżeli istniał już atrybut o takiej nazwie, to zostanie on usunięty. Zawartość atrybutu może być ustalona na sztywno przez podanie jej w atrybucie *value*:

```
<attribute xpath="//ipk:draw[not(@prior)]" name="prior" value="099"/>
```

W powyższym przykładzie elementy `draw`, które nie posiadają atrybutu `prior` otrzymają taki atrybut, o wartości "099".

Możliwe jest też ustalenie zawartości atrybutu przez skopiowanie innego atrybutu, podanego jako wyrażenie XPath w atrybucie *copy*, przy czym obliczanie wyrażenia zaczyna się od elementu docelowego (podanego jako atrybut *xpath*). Przykładowo:

```
<attribute xpath="//ipk:param[not(@draw_name)]" name="draw_name" copy="@name"/>
```

Elementy 'param' nie posiadające atrybutu 'draw_name' dostaną go z wartością skopiowaną z atrybutu 'name' tego samego elementu.

Przekształcenia są stosowane w takiej kolejności w jakiej wystąpią w pliku, można je więc łączyć aby uzyskać różnorodne efekty.

Ostatecznie, w pliku konfiguracyjnym może wystąpić element *ipk:drawdefinable*, zawierający dowolną ilość parametrów definiowalnych przeglądającego, w postaci dokładnie takiej, jak w dokumencie IPK. Formuły tych elementów mają dostęp do całej łączonej konfiguracji. Należy tylko pamiętać o zmianie nazw parametrów z konfiguracji składowych (poza pierwszą), oraz opatrywaniu wszystkich elementów prawidłowym prefiksem przestrzeni nazw IPK.

Następny rozdział (Sekcja 5.5.4) opisuje dodatkowy zaawansowany mechanizm, który może być wykorzystywany do tworzenia parametrów definiowalnych za pomocą szablonów.

5.5.4. Szablony parametrów definiowalnych

Szablony są mechanizmem pozwalającym na tworzenie parametrów definiowalnych na podstawie pozostałych obecnych w konfiguracji parametrów. Zarówno ilość parametrów jak i ich atrybuty mogą być zależne od danych obecnych w konfiguracji.

Szablony mogą być obecne w elemencie *ipk:drawdefinable* i są przetwarzane równolegle z kopiowaniem pozostałych parametrów definiowalnych, a więc po wszelkich innych przekształceniach na konfiguracji. Istotna jest kolejność wstawienia szablonu do parametrów definiowalnych - szablon ma dostęp do parametrów zdefiniowanych przed nim (także za pomocą innych szablonów), natomiast parametry zdefiniowane po nim mogą korzystać z parametrów powstałych w wyniku działania szablonu.

Szablon definiujemy przez element *template* o dwóch atrybutach:

- *xpath* - (atrybut obowiązkowy) określa zbiór elementów „bazowych” dla szablonu. Są one traktowane jako początek kontekstu przy dalszych wyrażeniach XPath, a poza tym, co ważniejsze, określają ilość utworzonych parametrów. Dla każdego elementu ze zbioru określanego przez atrybut zostanie iteracyjnie przetworzona zawartość elementu *template*.
- *name* - (opcjonalny) służy jako komentarz do szablonu, jego zawartość zostanie skopiowana do atrybutu *template* z przestrzeni nazw <http://www.praterm.com.pl/SZARP/extra>. Ma to na celu ułatwienie debugowania konfiguracji - wiadomo, że dany element w pliku wynikowym powstał jako efekt działania szablonu i wiadomo jaki to był szablon.

W elemencie *template* może być umieszczonych dowolna ilość elementów *variable*, które określają zmienne pomocnicze, używane do tworzenia parametrów wynikowych. Zmienne są globalne w całym dokumencie, mogą być wielokrotnie przedefiniowywane, za niezainicjowane zmienne podstawiane są puste napisy. Odwołania do zmiennych mogą występować w określonych, wymienionych w dokumentacji miejscach i mają postać $\backslash n$, gdzie *n* jest numerycznym identyfikatorem zmiennej (od 0 do 9). Element *variable* może mieć następujące atrybuty:

- *id* - obowiązkowy, numeryczny identyfikator zmiennej
- *xpath* - opcjonalny atrybut zawierający wyrażenie XPath typu string, które zostanie podstawione jako wartość zmiennej. Jeżeli atrybutu nie będzie, to zmienna będzie pustym napisem. Obliczanie wyrażenia XPath zaczyna się od węzła „bazowego”, czyli zdefiniowanego przez atrybut *xpath*

szablonu. Wartość wyrażenia musi być typu string, jeżeli potrzeba, możemy posłużyć się dodatkowo funkcją XPath *string()*. W zawartości atrybutu *xpath* są także dokonywane podstawienia za zmienne (czyli, w szczególności, zmienna może rekurencyjnie zależeć od siebie).

Zawartość elementu *variable* jeżeli nie jest pusta, traktowana jest jako wyrażenie regularne dokonujące podstawień na wartości zmiennej otrzymanej na podstawie atrybutu *xpath*. Sposób interpretacji wyrażenia i dokonywania podstawień jest identyczny, jak dla pozostałych elementów konfiguracji zagregowanej.

Wszystkie pozostałe elementy wchodzące w skład elementu *template* i nie pochodzące z przestrzeni nazw *aggr* są rekurencyjnie kopiowane do dokumentu wyjściowego. Na zawartości atrybutów i elementów dokonywane jest podstawienie zdefiniowanych wcześniej zmiennych za napisy postaci $\backslash n$ gdzie *n* odpowiada identyfikatorowi zmiennej, której wartość chcemy podstawić.

Notatka: W dokumencie mogą więc występować odwołania typu na przykład $\backslash 1$. Napis taki, zależnie od miejsca wystąpienia, może oznaczać odwołanie do zmiennej pomocniczej, albo podstawienie za fragment wyrażenia regularnego.

Pomocą w zrozumieniu działania mechanizmu może być następujący przykład. Załóżmy, że w konfiguracji zagregowanej (po zastosowaniu wcześniejszych przekształceń) mamy zestaw parametrów opisujących energię dostarczoną przez węzły cieplne, w MWh. Parametry te mają postać:

```
<ipk:param name="Węzły:[nazwa węzła]:energia sumaryczna" short_name="Esum"
  draw_name="Energia sumaryczna" unit="MWh" base_ind="auto" prec="2">
  <ipk:draw title="Węzeł [nazwa węzła]" min="0" max="2"/>
</ipk:param>
```

Chcemy dodać do konfiguracji parametry podające tą samą wielkość (energia z węzła) ale w GJ, a także parametr sumujący wszystkie te wartości. Możemy tego dokonać za pomocą następujących dwóch szablonów:

```
<ipk:drawdefinable>
  <template xpath="//ipk:param[starts-with(@name, '{chrw}') and
    ( contains(@name, ':Licznik energii CO') or
      ( contains(@name,':Licznik energii CO') )
    ) and not (
      contains(@name, 'Licznik energii CO ') or
      contains(@name, 'CO+CWU')
    ) ]"
    name="Energia z węzłów - przeliczanie GJ na MWh">
    <variable id="0" xpath="string(./@name)"/>
    <variable id="1" xpath="string(./@name)">s#^.*$#\0 GJ#</variable>
    <variable id="2" xpath="string(./ipk:draw/@title)"/>
    <variable id="9" xpath="'\9 (\1) 0 N +'"/>
    <ipk:param name="\1" short_name="Esum" draw_name="Energia sumaryczna MWh"
      unit="GJ" prec="2">
      <ipk:define type="DRAWDEFINABLE" formula="(\0) 0 N 3600 / "/>
      <ipk:draw title="\0" min="0" max="7200"/>
    </ipk:param>
  </template>
  <template xpath="//ipk:drawdefinable" name="Energia z węzłów - suma MWh">
    <ipk:param name="Węzły:Definiowalne:suma energii z węzłów"
      short_name="ESum" draw_name="Sumaryczna energia z węzłów"
```

```

    unit="GJ" prec="2">
    <ipk:define type="DRAWDEFINABLE" formula="0 \9"/>
    <ipk:draw title="Węzły - wielkości sumaryczne" min="0" max="20000"/>
  </ipk:param>
</template>
</ipk:drawdefinable>

```

Pierwszy szablon wybiera interesujące nas parametry (przykład jest rzeczywisty, więc zawiera kilka obejść dla błędów literowych, takich jak podwójne spacje w nazwach parametrów). Zmienna $\backslash 0$ zawiera nazwę parametru, zmienna $\backslash 1$ nazwę parametru po przekształceniu wyrażeniem regularnym (efekt dodania napisu „GJ” na końcu można oczywiście osiągnąć prościej, nie korzystając z wyrażen regularnych). Interesująca jest natomiast zmienna $\backslash 9$, która tworzy rekurencyjnie formułę zawierającą sumę wszystkich parametrów. Jest ona potem wykorzystywana do utworzenia parametru sumującego wszystkie poprzednie.

6. Składnia formuł parametrów definiowalnych

Składnia formuł parametrów definiowalnych przyjmuje nieco różną postać w zależności od typu parametru (*RPN*, *DRAWDEFINABLE* lub *LUA*), w którym dana formuła jest definiowana.

6.1. Składnia formuł typu DRAWDEFINABLE

Formuły typu *DRAWDEFINABLE* są zapisywane jako formuły matematyczne w odwrotnej notacji polskiej (ang. reverse polish notation - *RPN*). Oznacza to, iż przetwarzanie formuły odbywa się na stosie, na którym budowana jest jej reprezentacja. Wystąpienie liczby (czy to podanej jako stała, czy to jako parametr z bazy danych SZARP-a) powoduje odłożenie jej na szczyt stosu, zaś wystąpienie operatora powoduje zdjęcie ze stosu odpowiedniej liczby parametrów (zależnej od rodzaju operatora), wykonanie operacji odpowiadającej danemu operatorowi, a następnie odłożenie wyniku na szczyt stosu. Przykładowo sumę trzech liczb w *RPN* przedstawić można następująco:

```
1 2 + 3 +
```

W ten sposób wykonane zostanie dodanie liczby 1 do liczby 2, ich suma zostanie odłożona na szczyt stosu, a następnie do niej zostanie dodana liczba 3. Na kolejność wykonywania operacji można wpływać zmieniając kolejność argumentów w formule. Powyższy przykład można przepisać następująco:

```
1 2 3 + +
```

W tym momencie na stos zostaną w kolejności odłożone liczby 1, 2 i 3 (na szczycie stosu będzie znajdować się liczba 3), następnie zostanie wykonane dodawanie liczb 2 i 3, a do ich sumy zostanie dodana liczba 1.

W formułach typu *DRAWDEFINABLE* używać można nie tylko stałych liczbowych, ale także wartości parametrów z bazy danych SZARP-a. W tym celu należy podać nazwę danego parametru w nawiasach okrągłych. Jeśli któraś z części oddzielonych dwukropkami pokrywa się z nazwą parametru definiowalnego, w którym definiujemy daną formułę, można dla wygody zastąpić ją znakiem *. Na

przykład jeżeli definiujemy parametr o nazwie "Kocioł 1: Sterownik: stosunek energia/masa", można formułę zapisać następująco:

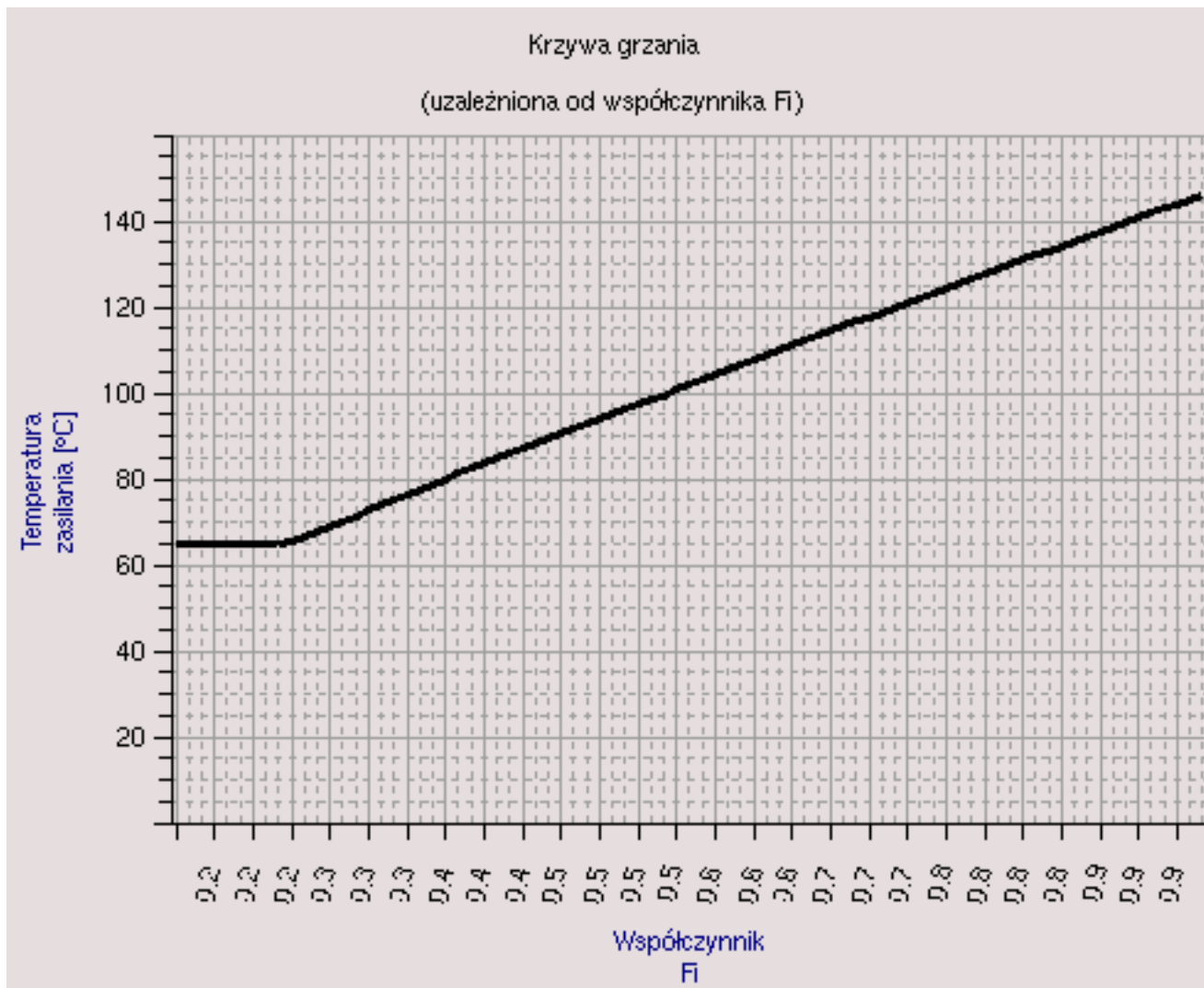
```
(Kocioł 1: Sterownik: stosunek energia/objętość) (Kocioł 1: Sterownik: masa nasypowa węgla
```

Jest to równoważne skróconemu zapisowi następującej postaci:

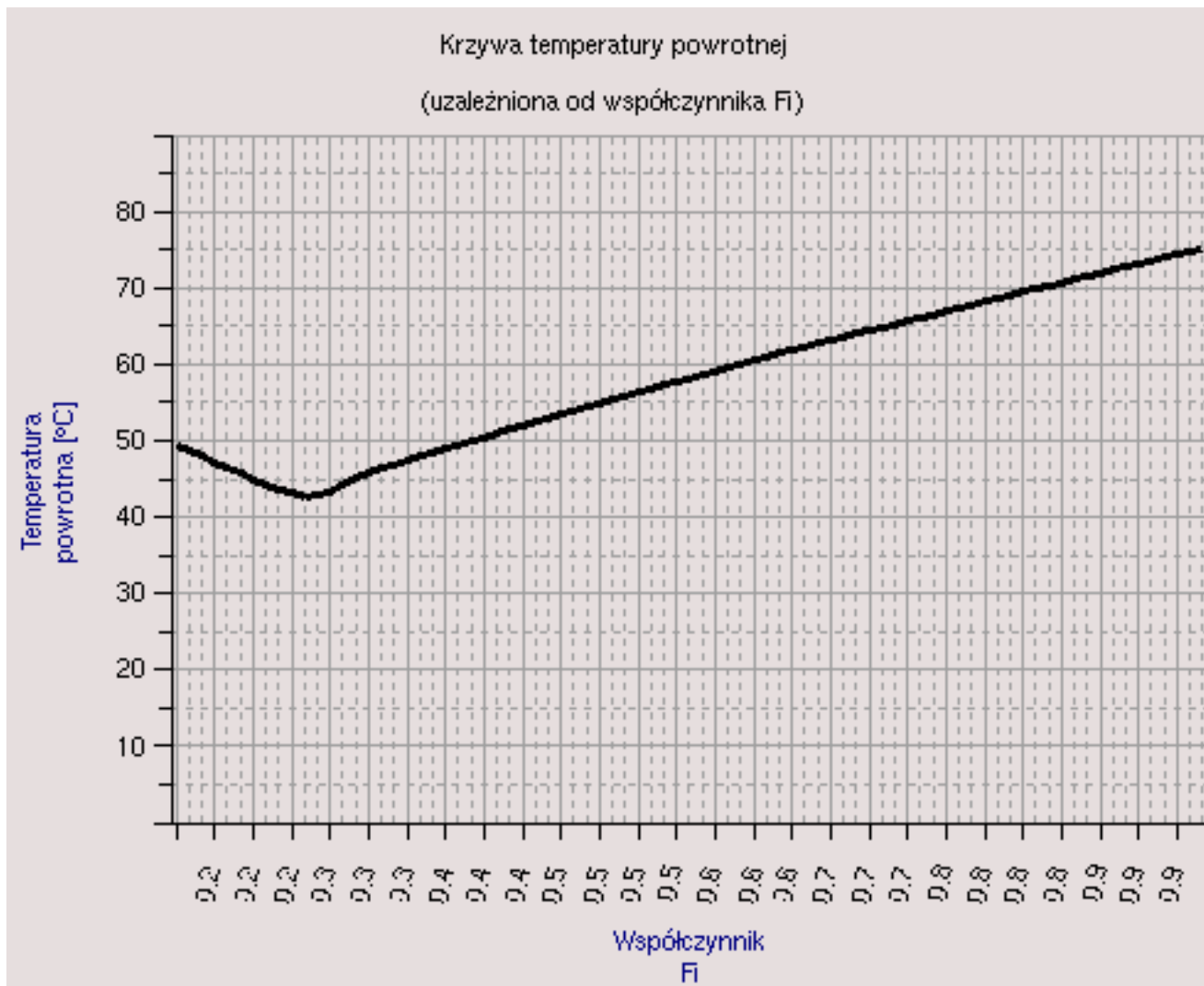
```
(*: *: stosunek energia/objętość) (*: *: masa nasypowa węgla) / 1000 *
```

Formuły DRAWDEFINABLE dają możliwość skorzystania z bardzo bogatego zestawu operatorów. Oprócz standardowych operatorów matematycznych (+, -, *, /) dostępne są następujące operatory mające specjalne znaczenie:

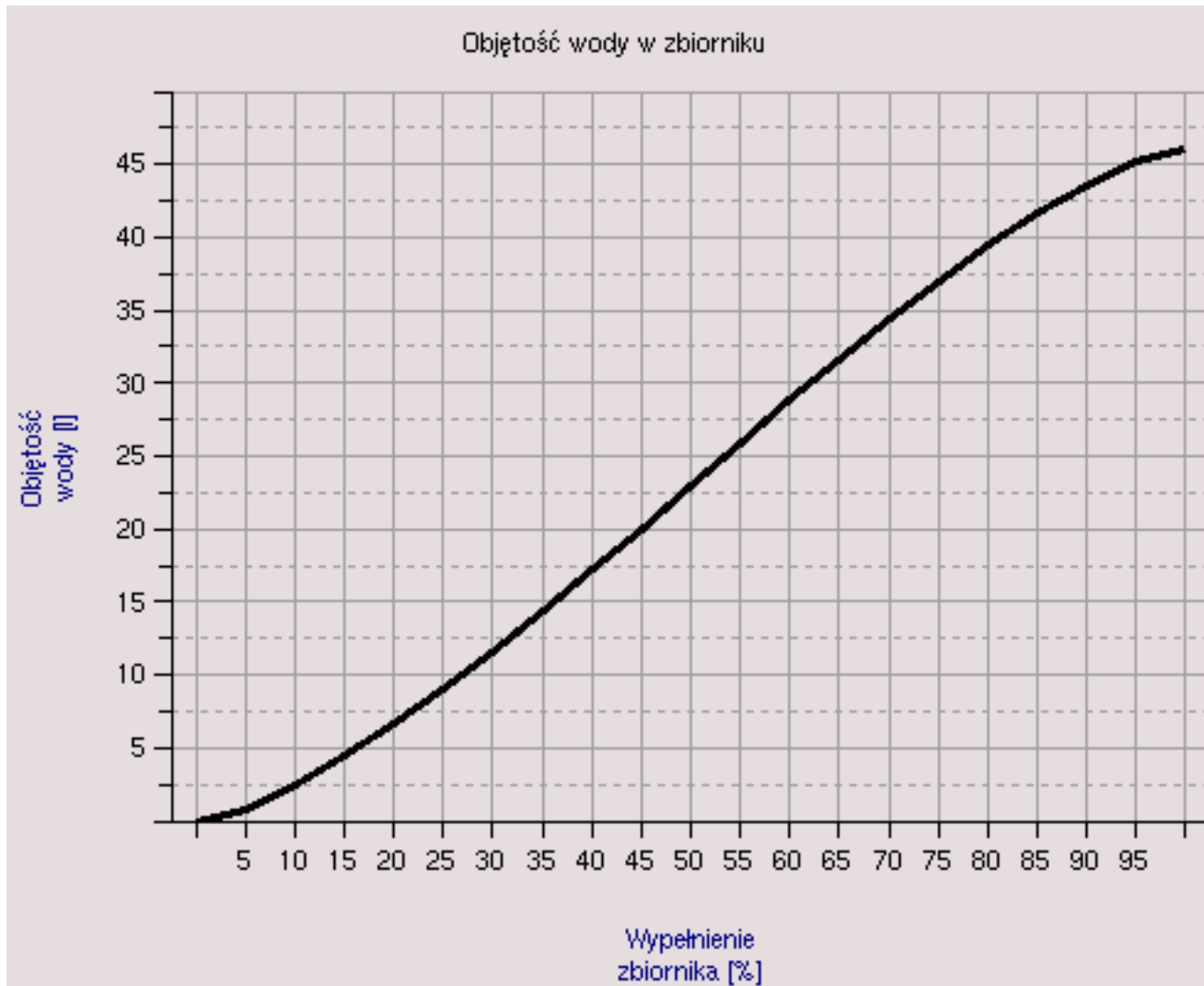
- & - zamienia miejscami dwa parametry na szczycie stosu,
- ! - powiela parametr ze szczytu stosu,
- \$ - oznacza wywołanie funkcji, której numer podany jest na szczycie stosu, liczba argumentów podawana jest jako drugi od góry element stosu, a argumenty są kolejnymi elementami na stosie, przetwarzanymi w kolejności od parametru najniżej na stosie do parametru najwyżej na stosie; dostępne funkcje to:
 - 0 - przyjmuje trzy argumenty; zwraca współczynnik F_i , przez który należy pomnożyć aktualną moc obliczeniową obiektu, aby otrzymać moc dla zadanych warunków atmosferycznych; warunki atmosferyczne podaje się w postaci trzech argumentów: aktualnej pogody (dozwolone wartości to: 0 - pochmurno, 1 - zmiennie, 2 - słonecznie), prędkości wiatru (w m/s) oraz temperatury zewnętrznej (czy to aktualnej czy średniej z jakiegoś okresu historycznego); funkcja ta uwzględnia jedynie potrzeby na CO - potrzeby CWU nie są uwzględnione, gdyż są one bardzo trudne do przewidzenia; współczynnik zwracany przez tą funkcję jest analogiczny do współczynników, które umieszczane były kiedyś w tabelach określających sposób prowadzenia systemu ciepłowniczego w zależności od warunków atmosferycznych (np. tabele opracowane przez Ministerstwo Gospodarki Materiałowej i Paliwowej w "Zasadach ustalania temperatur wody sieciowej w źródłach ciepła i sieciach ciepłowniczych" z 1987r.),
 - 1 - przyjmuje jeden argument; zwraca temperaturę zasilania dla podanego współczynnika F_i wg poniższej krzywej:



- 2 - przyjmuje jeden argument; zwraca temperaturę wody powrotnej dla podanego współczynnika Fi wg poniższej krzywej:



- 3 - przyjmuje jeden argument; dla argumentów dodatnich zwraca wartość argumentu, a dla ujemnych wartość 0,
- 4 - przyjmuje co najmniej dwa argumenty, przy czym pierwszy z nich oznacza rzeczywistą liczbę argumentów po nim występujących; zwraca sumę tych z podanych parametrów, które nie mają wartości NO_DATA,
- 5 - przyjmuje co najmniej dwa argumenty, przy czym pierwszy z nich oznacza rzeczywistą liczbę argumentów po nim występujących; zwraca liczbę parametrów, które nie mają wartości NO_DATA,
- 6 - przyjmuje jeden argument; dla procentowo podanego wypełnienia zbiornika o kształcie walca (taki kształt mają zbiorniki odgazowywacza, których wypełnienie może być liczone przy użyciu tej funkcji) funkcja zwraca objętość wody w nim się znajdującej wg poniższej zależności:



- 7 - przyjmuje dwa argumenty; z liczby podanej jako pierwszy argument zwraca bit o numerze podanym jako drugi argument.
- # - odkłada na szczyt stosu jednocyfrową wartość podaną bezpośrednio (bez spacji) za operatorem i ignoruje wszystkie znaki aż do wystąpienia następczej spacji,
- ?f - wyrażenie warunkowe: jeśli element ze szczytu stosu ma wartość 0, zwracana jest wartość występująca na drugiej pozycji od góry stosu, w przeciwnym wypadku zwracana jest wartość występująca na trzeciej pozycji od góry stosu,
- > - operator porównania większościowego: zwraca 1 jeśli druga od góry wartość na stosie jest większa niż wartość na szczycie stosu, a 0 w przeciwnym wypadku,
- < - operator porównania mniejszościowego: zwraca 1 jeśli druga od góry wartość na stosie jest mniejsza niż wartość na szczycie stosu, a 0 w przeciwnym wypadku,
- ~ - operator porównania równościowego: zwraca 1 jeśli wartość ze szczytu stosu jest równa drugiej od góry wartości na stosie, a 0 w przeciwnym wypadku,
- :- składa bitowo dwa górne elementy ze szczytu stosu w jeden o podwójnej długości: parametr ze szczytu stosu jest traktowany jako mniej znaczące słowo wartości wynikowej, a kolejny parametr jako jej bardziej znaczące słowo,

Notatka: Operator `:` definiuje tak naprawdę oddzielny typ parametru, tzn. składany (ang. combined), który może być zarówno wyświetlany przez raportera, jak i przez program przeglądający. *Nie wolno* używać operatora `:` w połączeniu z innymi operatorami, gdyż powoduje to niepoprawne wyliczanie wartości formuły definiowalnej. Jeżeli zachodzi konieczność wykonania jakichś operacji na wartości powstałej przez złożenie bitowe dwóch parametrów, należy to zrobić dwuetapowo: po pierwsze zdefiniować parametr, który będzie służył tylko do złożenia tych wartości, a następnie dopiero jego wartość wykorzystywać w dalszych obliczeniach.

- `^` - operator potęgowania: podstawą jest element ze szczytu stosu, a wykładnikiem drugi od góry element na stosie; wykładnik *musi* być nieujemny,
- `N` - operator porównania z wartością `NO_DATA`: jeśli element na szczycie stosu ma wartość `NO_DATA`, operator zwraca drugą od góry wartość na stosie, w przeciwnym wypadku zwracany jest element ze szczytu stosu,
- `X` - zwraca wartość `NO_DATA`,
- `S` - operator sezonu: zwraca wartość 1, jeśli aktualnie ciepłownia pracuje w trybie pracy sezonu letniego, a 0 w przeciwnym wypadku.

6.2. Składnia formuł typu RPN

Składnia formuł typu RPN jest bardzo podobna do składni formuł typu `DRAWDEFINABLE` (patrz Sekcja 6.1). Sposób ich obliczania jest identyczny, a jedyne różnice występują w dostępnym zestawie funkcji. Różnice te są następujące:

- odpowiednikiem operatora `?f` z formuł typu `DRAWDEFINABLE` jest operator `if`,
- występuje dodatkowy operator `n` oznaczający formułę pustą (tzn. bez żadnej wartości, nawet `NO_DATA`); dodatkowo operator ten powoduje pominięcie następnych trzech znaków (z tego powodu często jest on zapisywany jako `null`),
- występuje dodatkowy operator `=` powodujący zapisanie wartości ze szczytu stosu jako próbki do pamięci współdzielonej programu `parcook` pod adres podany jako drugi od góry element na stosie; zapis jest dokonywany tylko pod warunkiem, że wartość do zapisu nie była wartością pustą,
- występuje dodatkowy operator `m`, służący do szybkiego odfiltrowania błędnych (za małych) wartości; jeśli drugi od góry element na stosie jest mniejszy niż pierwszy od góry, to na stosie wstawiana jest wartość `NO_DATA`; wpp. zostaje wartość drugiego od góry elementu.

Przykładowo:

```
(Sieć:Sterownik:temperatura zasilania) 0 m
```

powyższa formuła spowoduje że jeśli wartość parametru będzie mniejsza od zera, to nie zostanie uwzględniona (pojawi się wartość brak danych).

- występuje symetryczny do poprzedniego operator `M` służący do odfiltrowania za dużych wartości; jeśli drugi od góry element na stosie jest większy niż pierwszy od góry, to na stosie wstawiana jest wartość `NO_DATA`; wpp. zostaje wartość drugiego od góry elementu.
- nie są dostępne operatory `^`, `X`, `S` oraz `:`,

- w związku z brakiem operatora X , wartość `NO_DATA` wyraża się poprzez podanie stałej liczbowej wynoszącej `-32768` - nie jest zatem możliwe używanie takiej liczby jako służącej do obliczeń.

6.3. Przykłady formuł definiowalnych DRAWDEFINABLE

Poniżej zaprezentowanych zostanie kilka przykładów formuł definiowalnych wraz z komentarzami. Prezentowana składnia jest składnią dla parametrów typu DRAWDEFINABLE, ale sposób formułowania wzorów jest identyczny dla formuł typu RPN. Dla czytelności formuły zostały podzielone na wiele linii, jednakże w IPK należy je wpisywać w jednej linii.

1. Składanie dwóch parametrów z bazy SZARP-a w jeden parametr o podwójnej długości:

```
(*::przepływ aktualny msw) (*::przepływ aktualny lsw) :
```

Jest to bardzo przydatna możliwość przy konieczności przechowywania w bazie wartości dłuższych niż 16 bitów - dzięki operatorowi `:` można na wykresie zaprezentować rzeczywistą, 32-bitową wartość mimo, iż w bazie trzymana jest ona jako dwa parametry o mniejszej długości.

2. Ustalenie wartości binarnej w zależności od wyniku porównania:

```
0 1 (Sieć:Sterownik:moc odpływu 1) (*::Moc odcięcia TSWL) < ?f
```

Przykładowy parametr przyjmuje wartość 0, jeśli dany odpływ pracuje z mocą mniejszą od mocy odcięcia, a 1 w przeciwnym wypadku. Łatwo zauważyć, że możliwy jest następujący zapis równoważny:

```
(Sieć:Sterownik:moc odpływu 1) (*::Moc odcięcia TSWL) >
```

3. Parametr przyjmujący różne wartości w lecie i w zimie:

```
200 100 S ?f
```

Powyższy parametr przyjmie wartość 200 w sezonie zimowym, a wartość 100 w sezonie letnim.

4. Zastępowanie wartości `NO_DATA` wartością 0:

```
(*::Finansowa strata na Twy TSWL) 0 N  
(*::Finansowa strata energii przesyłu TSWL) 0 N +
```

W przedstawionym przykładzie sumowane są dwa parametry, przy czym w przypadku każdego z nich w razie wystąpienia wartości `NO_DATA`, zastępowana jest ona wartością 0. Jeżeli liczba parametrów branych do sumowania byłaby znacząco większa, warto by było rozważyć skorzystanie z funkcji 4 operatora `$`.

5. Warunkowe nadawanie parametrowi wartości `NO_DATA`:

```
(*::Odchyłka Twy) (*::QS_70) * 4 * 100 / X (*::Praca ciepłowni) ?f
```

Przykładowy parametr przyjmuje rzeczywistą wartość tylko wtedy, gdy parametr "Praca ciepłowni" ma wartość 1, w przeciwnym wypadku przyjmuje wartość `NO_DATA`, co jest dobrym sposobem zasygnalizowania wystąpienia przypadku, w którym wyliczenie danego parametru nie miałyby sensu.

6. Sprawdzanie czy parametr przyjmuje konkretną, z góry ustaloną wartość:

```
(Sieć:Przepływy:praca pompy przewałowej nr 1) 0 >
(Sieć:Przepływy:praca pompy przewałowej nr 2) 0 >
+ 0 ~
```

Powyższy przykład pokazuje sposób na sprawdzenie czy obie pompy jednocześnie nie pracują (jest to rzeczywisty przykład z konfiguracji jednego z systemów ciepłowniczych i pozwala on na stwierdzenie, czy zawór obejścia pomp przewałowych jest otwarty).

7. Wartość parametru zależna od stanu konkretnego bitu w innym parametrze:

```
240 150 (*::zakodowany stan wejść logicznych) 2 2 7 $ ?f
```

W podanym przykładzie wywoływana jest funkcja numer 7 operatora \$ w celu wydobywania z zakodowanego stanu wejść logicznych stanu drugiego bitu. Jeśli bit ten ma wartość 1, to parametr przyjmuje wartość 240, w przeciwnym wypadku przyjmuje wartość 150. Powyższy przykład również jest rzeczywistym przykładem zaczerpniętym z działającej konfiguracji jednego z systemów ciepłowniczych - takie konstrukcje wykorzystywane są np. w celu interpretacji wartości będących mapami bitowymi przesyłanych przez regulatory.

8. Zliczanie parametrów nie mających wartości NO_DATA:

```
3 (Paca 6:WCW:Przepływ upustowy na węźle Paca 6)
(WP-6:WCW:Przepływ upustowy na węźle WP-6) (WP-3:WCW:Przepływ upustowy na węźle WP-
4 5 $
```

W podanym przykładzie zliczanie odbywa się na trzech parametrach dla większej jego czytelności, jednak w rzeczywistych przypadkach w konfiguracjach zazwyczaj zliczanie przebiega na co najmniej kilkunastu wartościach.

9. Sumowanie dużej ilości parametrów:

```
3 (WG-6:WCW:Przepływ recyrkulacji na węźle WG-6)
(WG-4:WCW:Przepływ recyrkulacji na węźle WG-4) (WG-9:WCW:Przepływ recyrkulacji na w
4 4 $
```

W podanym przykładzie sumowanie odbywa się na trzech parametrach dla większej jego czytelności, jednak w rzeczywistych przypadkach w konfiguracjach zazwyczaj sumowanie przebiega na co najmniej kilkunastu wartościach.

10. Ograniczanie wartości parametru tylko do wartości nieujemnych:

```
(*::Przepływ wody przez węzeł WG-9)
(*:WCO:Temp. wody powr. za obejściem WG-9) (*::Temp. wody powr. przed obejściem WG
(*::Temperatura wody z ciepłowni WG-9) (*:WCO:Temp. wody powr. za obejściem WG-9)
/ * 1 3 $
```

W powyższym przykładzie wyliczana jest wydajność, która z oczywistych względów nie może być ujemna, więc jej wartość została ograniczona tylko do wartości nieujemnych przy użyciu funkcji 3 operatora \$.

11. Wyznaczanie prognozowanej zmiany temperatury w nadchodzącym okresie:

```
(Klawiatura::Pogoda) (Klawiatura::Prędkość wiatru)
(Klawiatura::Średnia temperatura zewnętrzna przedostat doby)
(Klawiatura::Średnia temperatura zewnętrzna ostatniej doby)
(Klawiatura::Średnia temperatura zewnętrzna dla bieżącej doby)
+ + 30 /
3 0 $
1 1 $
```

```

10 *
(Klawiatura::Pogoda) (Klawiatura::Prędkość wiatru)
(Klawiatura::Średnia temperatura zewnętrzna przedost doby)
(Klawiatura::Średnia temperatura zewnętrzna ostatniej doby)
(Klawiatura::Średnia temperatura zewnętrzna dla bieżącej doby)
+ + 30 /
3 0 $
1 2 $
10 * -
(Klawiatura::Przepływ obliczeniowy) (Klawiatura::Przepływ rzeczywisty) / 1 - * 2 /

```

W powyższym przykładzie prognozowana zmiana temperatury w nadchodzącym okresie wyliczana jest na podstawie współczynnika F_i (funkcja 0 operatora \$), obliczonej temperatury zasilania (funkcja 1 operatora \$) oraz powrotnej (funkcja 2 operatora \$).

6.4. Parametry definiowalne LUA

Parametry definiowalne LUA (<http://www.lua.org/>) są odpowiednikiem parametrów definiowalnych pozwalającym na zapisanie formuł parametrów w postaci skryptów języka LUA. Skrypt może mieć dowolną zawartość, np. można używać zmiennych pomocniczych (co nie jest dostępne w 'zwykłych' definiowalnych), pętli, wyrażeń warunkowych etc. Celem skryptu jest przypisanie wartości zmiennej v , która jest interpretowana jako wartość wyliczonego parametru. Dokładne odbywa się to poprzez wywołanie poniższego kodu:

```

return function ()
    local p = szbase
    local PT_MIN10 = ProbeType.PT_MIN10
    local PT_HOUR = ProbeType.PT_HOUR
    local PT_HOUR8 = ProbeType.PT_HOUR8
    local PT_DAY = ProbeType.PT_DAY
    local PT_WEEK = ProbeType.PT_WEEK
    local PT_MONTH = ProbeType.PT_MONTH
    local PT_CUSTOM = ProbeType.PT_CUSTOM
    local szb_move_time = szb_move_time
    local state = {}
    return function (t,pt)
        local v = nil
        ....
        skrypt zawierający formułę
        ....
    return v
end
    end

```

Kolejną cechą parametrów definiowalnych LUA jest to, iż zawsze operują na liczbach zmiennoprzecinkowych w podwójnej precyzji, co jest udogodnieniem w stosunku do 'zwykłych' definiowalnych. Cecha ta występuje zarówno na poziomie parametrów definiowalnych programu przeglądającego, jak i programu parcook.

Parametry definiowalne LUA na poziomie programu przeglądającego pozwalają na znacznie większą swobodę niż 'zwykle' definiowalne. Między innymi dostępna jest zmienna z przypisanym czasem, dla którego wyliczana jest właśnie próbka, co umożliwia wykorzystanie jej do obliczeń. Dodatkowo dostępne są dwa tryby wyliczania wartości próbki. Próbka może być wyliczona jako średnia z zastosowania formuły do każdej próbki oraz jako wynik zastosowania formuły dla średnich. Przykładowo, jeżeli wyliczamy wartość parametru definiowalnego LUA dla średniej godzinowej, to w pierwszym przypadku zostanie wyliczona wartość każdej średniej z dziesięciu minut i wynik zostanie uśredniony, a w drugim będzie wyliczona będzie tylko wartość dla średniej godzinowej.

Poniżej zamieszczona jest przykładowa formuła parametru definiowalnego LUA:

```
local tmp = 2
v = p("swid:Sieć:Sterownik:temperatura z tabeli", t, pt, 0) * tmp
```

Formuła ta przypisuje zmiennej *v* (czyli wartości zwracanej przez formułę) wartość parametru *swid:Sieć:Sterownik:temperatura z tabeli* pomnożoną przez 2. Funkcja *p* służy do zwrócenia wartości zadanego parametru. Kolejne argumenty to: nazwa parametru, czas próbki (w tym przypadku równy czasowi próbki wyliczanej formułą), typ średniej oraz okres średniej w przypadku średniej typu *PT_CUSTOM* (obecnie nie jest wykorzystywany). Zmienne tymczasowe są poprzedzone słowem kluczowym *local*. Powoduje to, iż są one zmiennymi lokalnymi dla tej formuły. Można używać zmiennych globalnych, ale jest to niezalecane ze względów wydajnościowych.

6.4.1. Wykaz predefiniowanych zmiennych

W skryptach LUA są dostępne następujące predefiniowane zmienne:

- *v* - zmienna, pod którą ma być zapisana wyliczona wartość formuły.
- *t* - zmienna przechowuje czas, dla którego jest wyliczana wartość formuły. Niedostępne na poziomie *parcook*.
- *pt* - zmienna przechowuje typ próbki, dla której jest wyliczana wartość formuły. Niedostępne na poziomie *parcook*.
- *state* - jest to tablica, która zachowuje zawartość między kolejnymi wywołaniami skryptu.

Przykładowe użycie:

```
if state['prev'] == nil then
v = 1
state['prev'] = 1
elseif state['prev'] == 1 then
v = 2
state['prev'] = 2
else
v = nan()
state['prev'] = nil
end
```

Opis funkcji *nan* dostępny jest w Sekcja 6.4.2

- *PT_MIN10* - stała opisująca typ próbki.
- *PT_HOUR* - jak wyżej.
- *PT_HOUR8* - jak wyżej.
- *PT_DAY* - jak wyżej.

- *PT_WEEK* - jak wyżej.
- *PT_MONTH* - jak wyżej.
- *PT_CUSTOM* - jak wyżej.

6.4.2. Wykaz dostępnych funkcji

W skryptach LUA dostępne są następujące funkcje:

- *p* - funkcja zwraca wartość innego parametru z bazy danych. Kolejne parametry:
 - nazwa parametru,
 - czas próbki do pobrania,
 - typ średniej,
 - okres średniej w przypadku średniej typu *PT_CUSTOM* (obecnie nie jest wykorzystywany).

Przykład wyliczający średnią arytmetyczną z dwóch parametrów:

```
v = p("swid:Sieć:Sterownik:temperatura wody powrotnej z sieci c.o.", t, pt, 0)
v = v + p("swid:Sieć:Sterownik:temperatura wody wyjściowej do sieci c.o.", t, pt, 0)
v = v / 2
```

- *i* - funkcja zwraca wartość danego parametru podanego jako argument z pamięci współdzielonej. Dostępna tylko w parametrach definiowalnych na poziomie programu parcook. Przykład pobierający wartość parametru z pamięci współdzielonej:

```
v = i("Sieć:Sterownik:temperatura wody powrotnej z sieci c.o.")
```

- *szb_move_time* funkcja zwraca czas przesunięty o określoną liczbę okresów średniej. Kolejne parametry:
 - czas odniesienia,
 - liczba okresów, o które należy przesunąć czas odniesienia (dozwolone są wartości dodatnie i ujemne),
 - typ średniej,
 - okres średniej w przypadku średniej typu *PT_CUSTOM* (obecnie nie jest wykorzystywany).

Przykład wyliczający wartość danego parametru jako średnią arytmetyczną z 5 ostatnich wartości:

```
local j = 5
local i = j
local tmptime = t
v = 0
while i > 0 do
    v = v + p("swid:Sieć:Sterownik:temperatura wody powrotnej z sieci c.o.", tmptime)
    i = i - 1
    tmptime = szb_move_time(tmptime, -1, pt, 0)
end
v = v / j
```

- *szb_round_time* - funkcja zwraca czas zaokrąglony do odpowiedniego typu średniej. Kolejne parametry:

- czas do zaokrąglenia,
- typ średniej,
- okres średniej w przypadku średniej typu PT_CUSTOM (obecnie nie jest wykorzystywany).

Przykład wyznaczający parametr jako wartość z początku miesiąca:

```
local tmptime = szb_round_time(t, PT_MONTH, 0)
v = p("swid:Sieć:Sterownik:temperatura wody powrotnej z sieci c.o.", tmptime, PT_MON
```

- *szb_search_first* - funkcja zwraca czas wystąpienia pierwszej wartości podanego jako argument parametru.
- *szb_search_last* - funkcja zwraca czas wystąpienia ostatniej wartości podanego jako argument parametru.
- *isnan* - sprawdza, czy podana jako argument liczba, jest prawidłową wartością liczbową.
- *nan* - zwraca niepoprawną wartość liczbową. Przykład użycia *isnan* i *nan*:

```
local pv = i("swid:Sieć:Sterownik:temperatura wody powrotnej z sieci c.o.")
if isnan(pv) then
v = 0
elseif pv == 1 then
v = 1
else
v = nan()
end
```

6.4.3. Przykłady wpisów w params.xml

Poniżej zamieszczony jest przykład wpisu w params.xml (sekcja drawdefinable) definiujący parametr LUA na poziomie programu przeglądającego:

```
<param name="Sieć:Sterownik:Stopniodni" short_name="StD" draw_name="Stopniodni" unit="
  <define type="LUA" lua_formula="va" lua_start_date_time="2008-01-01 00:00"
    <script><![CDATA[
      local ct
      ct = p("prza:Sieć:Sterownik:Temperatura zewnętrzna", t, PT
      if not(in_season("prza", t)) and ct < 18 then
        v = 18 - ct
      else
        v = 0
      end
    ]]></script>
  </define>
  <draw title="Stopniodni" min="0" max="30" special="hoursum"/>
</param>
```

Atrybut *lua_formula* o wartości *va* mówi że średnie parametru obliczane są jako średnia z policzonych za pomocą formuły poszczególnych wartości w obejmowanym przez średnią okresie (zobacz Sekcja 5.2.8).

Przedział czasowy, dla którego wyliczane są wartości parametru jest zawężony przez podanie atrybutu *lua_start_date_time*

Natomiast poniżej podany jest przykład wpisu w params.xml (sekcja defined) dotyczącego parametru definiowalnego LUA na poziomie programu parcook:

```

<param name="Lua:Param:flipflop" short_name="FF" draw_name="Flip flop" unit="-" prec="
<define type="RPN" formula="null">
<script>
<![CDATA[
local pv = i("Lua:Param:flipflop")
if isnan(pv) then
v = 1
elseif pv == 1 then
v = 2
else
v = nan()
end
]]>
</script>
</define>
<draw title="Przykład formuły LUA w programie parcook" min="0" max="10" order="4"/>
</param>

```

Należy zwrócić uwagę na to, iż atrybut *type* ma wartość "RPN", a *formula* "null". Formuła definiuje parametr, który w poszczególnych przebiegach programu zmienia swoją wartość - kolejno 1, potem 2, potem brak danych i od nowa.

7. Obsługiwane urządzenia

Do komunikacji z urządzeniami SZARP wykorzystuje sterowniki, zwane też ze względów historycznych *demonami linii*. Rozdział zawiera spis aktualnie dostępnych w systemie SZARP sterowników. Opisy tworzone są automatycznie na podstawie kodu źródłowego, stąd też nie wszystkie muszą być w danej chwili kompletne, część opisów może być też dostępna jedynie w języku angielskim.

Specyfikacja w opisie sterownika oznacza zgodność z daną klasą specyfikacji, gdzie wyższy numer oznacza nowszą klasę. Poszczególne klasy to:

- 4 - sterownik wykorzystuje API DaemonConfig do konfiguracji i IPCHandler do komunikacji z systemem SZARP.
- 3 - sterownik nie spełnia wymagań klasy 4, ale czyta konfigurację wyłącznie z params.xml (lub linii komend podawanej przez parcooka).
- 2 - sterownik nie spełnia wymagań klasy 3 (np. wymaga plików konfiguracyjnych w formacie SZARP 2.1, ale korzysta do komunikacji z SZARP, logowania itp. z API libSzarp).
- 1 - sterownik nie spełnia wymagań klasy 2, np. we własnym zakresie łączy się do mechanizmów IPC SZARP lub ma własny mechanizm logowania.

7.1. Sterownik alstdmn

- Zgodność ze specyfikacją: 2 .
- Obsługiwane urządzenia: Licznik energii elektrycznej Alstom M301
- Protokół komunikacji: Modbus RTU

7.2. Sterownik aqtdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Ciepłomierze SuperCal Aquatherm 431/432
- Szczytywane parametry: Energia MSB, Energia LSB, Przepływ zliczony MSB, Przepływ zliczony LSB, Temperatura zasilania MSB, Temperatura zasilania LSB, Temperatura powrotu MSB, Temperatura powrotu LSB, Przepływ aktualny MSB, Przepływ aktualny LSB

7.3. Sterownik boruta_fp210

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Sterownik do demona borutadmn, obsługujący przepływomierz FP210 firmy Metronic.
- Protokół komunikacji: Protokół tekstowy FP210 na linii szeregowej RS232/485 lub konwerterze ethernet/RS232.
- Konfiguracja: Sterownik jest konfigurowany w pliku params.xml, w podelemencie unit elementu device. Opis dodatkowych atrybutów XML znajduje się w przykładzie poniżej.
- Przykładowa konfiguracja:

```
<device
xmlns:extra="http://www.praterm.com.pl/SZARP/ipk-extra"
daemon="/opt/szarp/bin/borutadmn"
path="/dev/null"
ignored, should be /dev/null
speed="9600"
ignored
>
```

```
<unit
id="1"
ignorowany
type="1"
ignorowany, powinno być "1"
    subtype="1"
ignorowany, powinno być "1"
```

```

bufsize="1"
wielkość bufora uśredniania, 1 lub więcej
extra:id="0x03"
identyfikator przelicznika, musi odpowiadać identyfikatorowi ustawionemu w urządzeniu
extra:proto="fp210"
nazwa protokołu, używana przez Borutę do ustalenia używanego sterownika, dla
tego sterownika musi być "fp210"
extra:mode="client"
tryb pracy jednostki, dla tego sterownika powinien być "client"
extra:medium="serial"
medium transmisyjne, "serial" dla RS232, "tcp" dla konwertera ethernet/RS232
extra:tcp-address="172.18.2.2"
extra:tcp-port="23"
adres i port IP do którego się łączymy, używany dla medium "tcp"
extra:path="/dev/ttyS0"
ścieżka do portu szeregowego dla medium "serial"
extra:speed="19200"
opcjonalna prędkość portu szeregowego w bps dla medium "serial", domyślna to 9600,
inne możliwe wartości to 300, 600, 1200, 2400, 4800, 19200, 38400; ustawienie innej
wartości spowoduje przyjęcie prędkości 9600
extra:parity="even"
opcjonalna parzystość portu dla medium "serial", możliwe wartości to "none" (domyślna),
"odd" i "even"
extra:stopbits="1"
opcjonalna liczba bitów stopu dla medium "serial", 1 (domyślnie) lub 2
>
<param name=".....:przepływ" unit="m3/h" prec="2" .../>
<param name=".....:licznik msw" unit="m3" prec="2" .../>
<param name=".....:licznik lsw" unit="m3" prec="2" .../>
<param name=".....:sumator 2 msw" unit="m3" prec="2" .../>
<param name=".....:sumator 2 lsw" unit="m3" prec="2" .../>
z urządzenia czytanych jest 5 parametrów, pierwszy to aktualny przepływ,
pozostałe to 2 sumatory, każdy reprezentowany przez 2 parametry 'kombinowane'
...
    </unit>
...
</device>

```

7.4. Sterownik boruta_lumel

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Sterownik do demona borutadmn, obsługujący protokół Lumel.

7.5. Sterownik boruta_modbus

- Zgodność ze specyfikacją: 4 .

- Obsługiwane urządzenia: Sterownik do demona borutadmn, obsługujący protokół Modbus, powszechnie używany przez różnego rodzaju urządzenia pomiarowe, sterowniki PLC czy systemy SCADA.
- Protokół komunikacji: Modbus RTU lub ASCII na łączu szeregowym (tryb master i slave), Modbus TCP przez połączenie sieciowe (tryb client i server)
- Konfiguracja: Sterownik jest konfigurowany w pliku params.xml, w podelemencie unit elementu device. Opis dodatkowych atrybutów XML znajduje się w przykładzie poniżej.
- Przykładowa konfiguracja:

```
<device
```

```
  xmlns:extra="http://www.praterm.com.pl/SZARP/ipk-extra"
  daemon="/opt/szarp/bin/borutadmn"
  path="/dev/null"
    ignorowany, zaleca się ustawienie /dev/null
  speed="9600"
    ignorowany
  >
```

```
<unit
```

```
  id="1"
```

identyfikator Modbus nasz lub strony przeciwnej (zależnie od trybu server/client); jeżeli jest literą lub pojedynczą cyfrą to interpretowany jest jako znak ASCII (czyli 'A' to '1' to 49), wpp. interpretowany jest jako liczba (czyli '01' to 1); zaleca się używać zamiast niego jednoznacznego atrybutu extra:od

```
extra:id="1"
```

identyfikator Modbus jednostki, nadpisuje atrybut 'id', zawsze interpretowany jako 1

```
  type="1"
```

```
    ignorowany, powinno być "1"
```

```
  subtype="1"
```

```
    ignorowany, powinno być "1"
```

```
  bufsize="1"
```

```
    wielkość bufora uśredniania, 1 lub więcej
```

```
  extra:proto="modbus"
```

```
    nazwa protokołu, używana przez Borutę do ustalenia używanego
    tego sterownika musi być "modbus"
```

```
  extra:mode="client"
```

```
    tryb pracy jednostki, "client" lub "server"; client oznacza
```

dla Modbus RTU, klient dla Modbus TCP); server oznacza stronę pasywną (slave dla Modbus RTU, serwer dla Modbus TCP)

```
  extra:medium="serial"
```

```
    medium transmisyjne, "serial" dla Modbus TCP, "tcp" dla Modbus
```

```
  extra:tcp-address="172.18.2.2"
```

adres IP do którego się podłączamy, wymagany dla trybu "client" i medium "tcp"

```
  extra:tcp-port="23"
```

port IP na który się łączymy (mode "client") lub na którym nasłuchujemy (mode "server"), wymagany dla medium "tcp",

```
  extra:path="/dev/ttyS0"
```

```
    ścieżka do portu szeregowego dla medium "serial"
```

```
  extra:speed="19200"
```

```
    opcjonalna prędkość portu szeregowego w bps dla medium "serial"
```

```
    inne możliwe wartości to 300, 600, 1200, 2400, 4800, 19200, 9600
```

```
    wartości spowoduje przyjęcie prędkości 9600
```

```
  extra:parity="even"
```

```

        opcjonalna parzystość portu dla medium "serial", możliwe wartości
        "odd" i "even"
        extra:stopbits="1"
        opcjonalna liczba bitów stopu dla medium "serial", 1 (domyślnie)
extra:FloatOrder="msblsb"
kolejność słów dla wartości zajmujących 2 rejestry (typu "long" lub "float"), domyślnie
to "msblsb" (najpierw bardziej znaczące słowo), inna możliwość to "lsbmsb" (najpierw
mniej znaczące słowo); może być nadpisana dla konkretnego parametru
extra:nodata-timeout="60"
czas w sekundach po którym wygasa wartość parametru w przypadku braku komunikacji, domyślnie
to 0 (brak wygasania)
extra:nodata-value="-1"
wartość wysyłana gdy wartość parametru SZARP nie jest dostępna, domyślnie 0
    >
</param>
elementy param oznaczają wartości przesyłane/odczytywane z urządzenia do systemu SZARP
name="Name:Of:Parameter"
...
extra:address="0x12"
wymagany, podany dziesiątkowo lub szesnastkowo bezpośredni adres Modbus pierwszego rejestru
zawierającego wartość parametru, od 0 do 65535
extra:register_type="holding_register"
typ rejestru Modbus - "holding_register" (domyślnie) lub "input_register", decyduje jaki tryb
Modbus jest wykorzystywana do czytania wartości rejestru - (0x03 - ReadHoldingRegister, 0x04 -
ReadInputRegister); dla trybu "server" nie ma to znaczenia - akceptowane są oba tryby
zapisujące - zarówno 0x06 (WriteSingleRegister) jak i 0x10 (WriteMultipleRegisters)
extra:val_type="integer"
wymagany, typ kodowania wartości - jeden z "integer", "bcd" (niezawieszany dla elementów
"long" lub "float"; wartości typu "integer" i "bcd" zajmują 1 rejestr, "long" i "float"
dwa kolejne rejestry
extra:FloatOrder="lsbmsb"
nadpisuje wartość atrybutu FloatOrder jednostki dla konkretnego parametru
extra:val_op="MSW"
opcjonalny operator pozwalający na konwersję wartości typu float i long na wartości
parametrów SZARP; domyślnie wartości te zamieniane są na 16-bitową reprezentację wartości
w systemie SZARP bezpośrednio, jedynie z uwzględnieniem precyzji parametru w SZARP; można
jest jednak przepisanie tych wartości do dwóch parametrów SZARP (tak zwane parametry
'kombinowane') co pozwala na nietracenie precyzji i/lub uwzględnienie większego zakresu
w tym celu należy skonfigurować 2 parametry SZARP z takimi samymi parametrami dotyczącymi
adresu i typu, przy czym jeden z nich powinien mieć val_op ustawiony na "MSW", a drugi na
"LSW" - przyjmą wartości odpowiednio bardziej i mniej znaczącego słowa wartości parametru
Modbus
...
/>
...
<send>
elementy send oznaczają wartości przesyłane/czytane z systemu SZARP do urządzenia
param="Name:Of:Parameter"
nazwa parametru, którego wartość mamy przesyłać
extra:address="0x12"
adres Modbus docelowego rejestru
extra:register_type="holding_register"
jak dla elementu "param", ale ma znaczenie tylko dla trybu "server", dla trybu
"klient" używana jest zawsze funkcja 0x10 (WriteMultipleRegisters)
extra:val_type="integer"
extra:FloatOrder="lsbmsb"

```

```

extra:val_op="MSW"
analogicznie jak dla elementu param
    ...
</unit>
</device>

```

7.6. Sterownik boruta_wmtp

- Zgodność ze specyfikacją: nieznana .

7.7. Sterownik boruta_zet

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Sterownik do demona borutadmn, obsługujący protokół ZET, używany do komunikacji z produkowanymi dla firmy Praterm regulatorami Z-Elektronik, SK2000 i SK4000. Protokół ZET jest prostym protokołem typu ASCII z adresowaniem jednostek i sprawdzaniem sumy kontrolnej CRC, przeznaczonym do transmisji po łączu szeregowym lub za pomocą transmisji TCP.
- Protokół komunikacji: Protokół ZET na linii szeregowej RS232/485 lub konwerterze ethernet/RS232.
- Konfiguracja: Sterownik jest konfigurowany w pliku params.xml, w podelemencie unit elementu device. Opis dodatkowych atrybutów XML znajduje się w przykładzie poniżej.
- Przykładowa konfiguracja:

```

<device
xmlns:extra="http://www.praterm.com.pl/SZARP/ipk-extra"
daemon="/opt/szarp/bin/borutadmn"
path="/dev/null"
ignorowany, zaleca się ustawienie /dev/null
speed="9600"
ignorowany
>

<unit
id="1"
identyfikator sterownika
type="1"
ignorowany, powinno być "1"
    subtype="1"
ignorowany, powinno być "1"
bufsize="1"
wielkość bufora uśredniania, 1 lub więcej
extra:proto="zet"
nazwa protokołu, używana przez Borutę do ustalenia używanego sterownika, dla
tego sterownika musi być "zet"
extra:mode="client"
tryb pracy jednostki, dla tego sterownika powinien być "client"

```

```

extra:plc="zet"
typ regulatora - "zet" dla Z-Elektronika lub "sk" dla SK2000/SK4000
extra:medium="serial"
medium transmisyjne, "serial" dla RS232, "tcp" dla konwertera ethernet/RS232
extra:tcp-address="172.18.2.2"
extra:tcp-port="23"
adres i port IP do którego się łączymy, używany dla medium "tcp"
extra:path="/dev/ttyS0"
ścieżka do portu szeregowego dla medium "serial"
extra:speed="19200"
opcjonalna prędkość portu szeregowego w bps dla medium "serial", domyślna to 9600,
inne możliwe wartości to 300, 600, 1200, 2400, 4800, 19200, 38400; ustawienie innej
wartości spowoduje przyjęcie prędkości 9600
extra:parity="even"
opcjonalna parzystość portu dla medium "serial", możliwe wartości to "none" (domyślna),
"odd" i "even"
extra:stopbits="1"
opcjonalna liczba bitów stopu dla medium "serial", 1 (domyślnie) lub 2
>
<param .../>
Ilość i kolejność parametrów odpowiada ilości i kolejności parametrów w sterowniku.
...
    </unit>
...
</device>

```

7.8. Sterownik borutadmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Boruta jest uniwersalnym demonem, który sam w sobie nie implementuje żadnego z protokołów. Obsługą właściwych protokołów zajmują się sterowniki będące modułami boruty. Obecnie dostępne są sterowniki dla protokołów Modbus, ZET i FP210.
- Protokół komunikacji: Modbus RTU/ASCII, Modbus TCP, ZET i FP210
- Konfiguracja: Sterownik jest konfigurowany w pliku params.xml. Każdy podelement unit zawiera konfigurację jednego sterownika. Szczegóły konfiguracji znajdziesz w opisach poszczególnych sterowników.
- Przykładowa konfiguracja:

```

<device
xmlns:extra="http://www.praterm.com.pl/SZARP/ipk-extra"
daemon="/opt/szarp/bin/borutadmn"
path="/dev/null"
>

<unit id="1"
extra:mode="client"
this unit working mode possible values are 'client' and 'server'
extra:medium="serial"

```

```
data transmission medium, may be either serial (line) or tcp
extra:proto="modbus"
protocol to use for this unit
in case of serial line, following self-explanatory attributes are also supported:
extra:path, extra:speed, extra:parity, extra:stopbits, extra:char_size (all but path
they have defaults which are 9600, N, 1, 8)
in case of tcp client mode following attributes are required:
extra:tcp-address, extra:tcp-port
in case of tcp server mode one need to specify extra:tcp-port attribute
>
...
    </unit>
</device>
```

7.9. Sterownik calcdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Demon wyliczający wartość chwilową parametru z wartości sumarycznej np. chwilowy przepływ z przepływu sumarycznego.
- Ten demon jest zdezaktualizowany - ten sam efekt może być uzyskany za pomocą parametrów w języku skryptowym Lua.

7.10. Sterownik calecdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Ciepłomierz Aquametro CALEC MCP-300.
- Protokół komunikacji: Calec protocol.

Serial port settings: 9600 8N1 Wiring: RX, TX, GND

Protocol:

Q: AM

R: OK

Q: R4

R: ACK

Q: [AD1,AD2] (register address in MCP format, see below)

R: [data] (4 bytes)

R: [checksum] (1 byte - lower byte of arithmetic sum of previous 6 bytes, starting from AD1)

MCP address format, R is register address:

$AD2 = R \& 0xFF$


```

xmlns:db="http://www.praterm.com.pl/SZARP/ipk-extra"
  daemon="/opt/szarp/bin/dbdmn"
  path="/opt/szarp/trgr"
path to szbase main directory
db:expire="600"
time (in seconds) of data expiration - if last available data
is older then given ammount of seconds, NO_DATA is send;
set 0 to turn expiration off
<unit id="1" ...>
<param name="..."
db:param="....."
name of parameter to read from database

```

7.12. Sterownik ddedmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Sterownik ten może służyć do odczytu danych z 2 programów:
 - Wonderware InTouch - popularne oprogramowanie SCADA pod Windows (powinno też działać z dowolnym innym serwerem DDE, który na żądanie Request zwraca tekstową reprezentację wartości parametru)
 - MBENET - sterownik protokołu Modbus do oprogramowania InTouch
- Protokół komunikacji: Sterownik wykorzystuje protokół XMLRPC do komunikacji z proxy DDE uruchomionym na serwerze Windows.
- Wymaga uruchomienia na serwerze z Windows skryptu utils/ddeproxy.py, służącego jako proxy do komunikacji z protokołem DDE.
- Przykładowa konfiguracja:

```

<device daemon="/opt/szarp/bin/dededmn" path="/dev/null" dde:uri="http://192.168.1.1
  xmlns:dde="http://www.praterm.com.pl/SZARP/ipk-extra" dde:appname="VIEW">
  <unit id="1" type="2" subtype="1" bufsize="1">
    <param name="Kocioł 1:DDE:Temperatura wody przed kotłem" short_name="Twe" unit:
      dde:topic="Tagname" dde:item="Batch%Conc" dde:type="integer" dde:prec="1" l
    ....

```

Where:

dde:uri is URI of DDE proxy (XMLRPC server)

dde:read_freq is polling frequency (in seconds), minimum and default value is 10 seconds
 dde:appname is DDE application name, default is 'MBENET' for MBENET driver, use 'VIEW' for InTouch
 installation

dde:topic is DDE topic, for InTouch it's always 'Tagname', for MBENET use correct value
 dde:item is DDE item name, for InTouch it's a Tagname of param, for MBENET it's Modbus
 register address

dde:prec optional precision of 'string' parameter, value is divided by 10 ^ dde:prec
 dde:type:

- for InTouch use "string" - numbers are passed from InTouch as string and parsed as float
 precision; if you need to split large value into 2 parameters, use same topic and item
 and additional dde:word="msw" and dde:word="lsw" attributes for most- and less- significant

- for MBENET use on of:
 - 'integer' - simple one register short integer value (precision dependant)
 - 'float' - double precision float, hold in two Modbus registers and saved as two parameters
 - 'short_float' - double precision float, hold in two Modbus registers, but copied as one parameter

7.13. Sterownik dlmsdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Liczniki energii używające protokołu DLMS (Device Language Message Specification).
- Protokół komunikacji: DLMS/COSEM/HDLC (IEC-62056)

7.14. Sterownik dpafdmn

- Zgodność ze specyfikacją: 2 .
- Obsługiwane urządzenia: Liczniki energii DataPAF 3.X

7.15. Sterownik dprdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Rejestrator Honeywell DPR100C/DPR100D.
- Przykładowa konfiguracja:

```
<device
  xmlns:dprdmn="http://www.praterm.com.pl/SZARP/ipk-extra"
  daemon="/opt/szarp/bin/dprdmn"
  path="/dev/ttyA11"
  <unit id="1" dprdmn:unit_address="0x01">
    <param
      name="..."
      ...
      dprdmn:address="0x00"
      dprdmn:param_type="alarm">
      ...
    </param>
    <param
      name="..."
      ...
      prec="0x01"
      dprdmn:address="0x02"
      dprdmn:param_type="input">
```

```

        </param>
        ...
    </unit>
</device>

```

7.16. Sterownik eapnetdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Licznik energii Pozyton EAP-IMS-B13DB76
- Konfiguracja: Demon wymaga dodatkowych parametrów w linii komend, zobacz opcję '--help'.

7.17. Sterownik endmn

- Zgodność ze specyfikacją: 2 .
- Obsługiwane urządzenia: Licznik energii Circutor CVMk.
- Protokół komunikacji: Modbus RTU

7.18. Sterownik execdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Sterownik czyta wyjście generowane przez zewnętrzny program.
- Protokół komunikacji: Sterownik parsuje wyjście uruchamianego programu, spodziewając się surowych (bez precyzji) wartości kolejnych parametrów w kolejnych liniach.
- Konfiguracja: Atrybut 'path' elementu device w pliku params.xml zawiera ścieżkę do uruchamianego programu. Opcjonalny atrybut 'frequency' z dodatkowej przestrzeni nazw określa co ile sekund uruchamiać program (domyślnie co 10). Zawartość atrybutu 'options' jest dzielona na słowa i przekazywana jako opcje do uruchamianego programu.
- Przykładowa konfiguracja:

```

<device
  xmlns:exec="http://www.praterm.com.pl/SZARP/ipk-extra"
  daemon="/opt/szarp/bin/execdmn"
  path="/opt/szarp/bin/some_script"
  exec:frequency="30"
  options="--some-option -f some-argument">
  ...

```

7.19. Sterownik fp2kdmn

- Zgodność ze specyfikacją: 2 .
- Obsługiwane urządzenia: Metronic FP-2001, Metronic FP-2001W.
- Protokół komunikacji: Modbus RTU.
- Ten demon jest przestarzały, zamiast niego należy używać ogólnego demona do obsługi protokołu Modbus, czyli mbdmn.
- Konfiguracja: Czyta konfigurację z pliku fp2001.cfg file.

7.20. Sterownik iecdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Liczniki energii używające protokołu opisanego w IEC 62056-21.
- Protokół komunikacji: IEC (IEC-62056-21)

7.21. Sterownik ifcl5dmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Przelicznik Danfoss Infocal 5
- Protokół komunikacji: M-Bus (EN 1434-3, EN 60870-5)

7.22. Sterownik k601dmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Ciepłomierz Kamstrup Multical 601.
- Protokół komunikacji: Kamstrup Heat Meter Protocol.
- Producent ciepłomierzy Kamstrup nie zezwolił na upublicznienie protokołu komunikacyjnego, wobec czego sterownik ten wymaga zamkniętej wtyczki, dostępnej w bibliotece szarp-prop-plugins.so.
- Konfiguracja: Przykładowa konfiguracja w pliku params.xml, atrybuty z przetrzelenie nazw 'm601' są obowiązkowe.
- Przykładowa konfiguracja:

```
<device
  xmlns:m601="http://www.praterm.com.pl/SZARP/ipk-extra"
  daemon="/opt/szarp/bin/k601dmn"
  path="/dev/ttyA11"
  m601:delay_between_chars="10000"
  m601:delay_between_requests="10">
```

```

<unit id="1">
  <param
    name="..."
    ...
    k601:register="0x80"
    k601:type="lsb"
    k601:multiplier="100">
    ...
  </param>
  <param
    name="..."
    ...
    k601:address="0x80"
    k601:type="msb"
k601:multiplier="100">
    ...
  </param>
  ...
  <param
    name="..."
    ...
    k601:address="0x81"
    k601:type="single"
k601:multiplier="1">
    ...
  </param>
</unit>
</device>

```

7.23. Sterownik kamsdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Ciepłomierz Kamstrup Multical IV, Kamstrup Multical 66CDE.
- Producent ciepłomierzy Kamstrup nie zezwolił na upublicznienie protokołu komunikacyjnego, wobec czego sterownik ten wymaga zamkniętej wtyczki, dostępnej w bibliotece szarp-prop-plugins.so.

7.24. Sterownik kwmsdmn

- Zgodność ze specyfikacją: 2 .
- Obsługiwane urządzenia: Liczniki energii Pozyton KWMS.

7.25. Sterownik Ibdmn

- Zgodność ze specyfikacją: 2 .
- Obsługiwane urządzenia: Wiatromierz Label LB-746.

7.26. Sterownik Iecdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Ciepłomierze Apator-Kfap LEC-4/LEC-5.

7.27. Sterownik Iecmdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Ciepłomierze LEC z interfejsem M-Bus.
- Protokół komunikacji: M-Bus.

7.28. Sterownik Ilinedmn

- Zgodność ze specyfikacją: 2 .
- Obsługiwane urządzenia: Sterowniki PLC Z-Elektronik po radioliniach.
- Protokół komunikacji: Z-Elektronik.
- Jest to przestarzały, nieużywany demon, przeznaczony do usunięcia.

7.29. Sterownik Imldmn

- Zgodność ze specyfikacją: 2 .
- Obsługiwane urządzenia: Regulator PID Lumel RE-31.
- Przykładowa konfiguracja:

```
Numery parametrów z sekcji "OPTIONS" params.xml:  
Czas wytrzymania #DataCode = 1 #Comma = 1  
Prędkość narostu wartości zadanej #DataCode = 2 #Comma = 1  
Kompensacja odchylenia sygnału wejściowego #DataCode = 3 #Comma = 2  
Przesunięcie wartości regulowanej #DataCode = 4 #Comma = 1  
Nastawa zakresu proporcjonalności PID-a #DataCode = 5 #Comma = 0  
Nastawa czasu całkowania PID-a #DataCode = 6 #Comma = 0  
Nastawa czasu różniczkowania PID-a #DataCode = 7 #Comma = 0  
Histereza alarmu 1 #DataCode = 8 #Comma = 1
```

Histereza dla regulacji dwustanowej #DataCode = 9 #Comma = 1
Adres urządzenia w sieci #DataCode = 10 #Comma = 0
Dolna granica zakresu #DataCode = 11 #Comma = 1
Górna granica zakresu #DataCode = 12 #Comma = 1
Ograniczenie mocy wyjścia 1 #DataCode = 13 #Comma = 0
Ograniczenie mocy wyjścia 2 #DataCode = 14 #Comma = 0
Rodzaj sygnału wejściowego #DataCode = 15 #Comma = 0
Typ jednostk #DataCode = 16 #Comma = 0
Rozdzielczość #DataCode = 17 #Comma = 0
Rodzaj regulacji #DataCode = 18 #Comma = 0
Typ alarmu #DataCode = 19 #Comma = 0
Funkcje alarmu 2 #DataCode = 20 #Comma = 0
Okres impulsowania dla wyjścia 1 #DataCode = 21 #Comma = 0
Okres impulsowania dla wyjścia chłodzeni #DataCode = 22 #Comma = 0
Zakres proporcjonalności dla wyjścia chłodzenia #DataCode = 23 #Comma = 1
Strefa nieczułości #DataCode = 24 #Comma = 1
Wartość mierzona #DataCode = 25 #Comma = 0
Wartość zadana #DataCode = 26 #Comma = 1
Wartość sygnału na wyjściu 1 # DataCode = 27 #Comma = 0
Wartość sygnału na wyjściu 2 #DataCode = 28 #Comma = 0
Wartość alarmu 2 lub czas wytrzymania #DataCode = 29 #Comma = 1
Histereza alarmu 2 #DataCode = 30 #Comma = 1
Typ alarmu 2 #DataCode = 31 #Comma = 0
Funkcje alarmu 2 #DataCode = 32 #Comma = 0

7.30. Sterownik logdmn

- Zgodność ze specyfikacją: nieznana .

7.31. Sterownik mbdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Wszystkie urządzenia i systemy SCADA (lub inne oprogramowanie) wykorzystujące protokół Modbus.
- Protokół komunikacji: Modbus RTU (połączenie szeregowe RS232/RS485) lub Modbus TCP/IP.
- Ten sterownik zastępuje starsze sterowniki mbtudmn i mbtcpdmn.
- Konfiguracja: Sterownik jest konfigurowany w pliku params.xml, wszystkie atrybuty z przestrzeni nazw 'modbus', nie opisane jako opcjonalne, są wymagane.
- Przykładowa konfiguracja:

```
<device
  xmlns:modbus="http://www.praterm.com.pl/SZARP/ipk-extra"
  daemon="/opt/szarp/bin/mbdmn"
  path="/dev/null"
  modbus:daemon-mode=
```

```

    allowed modes are 'tcp-server' and 'tcp-client' and 'serial-client' and 'serial-se
    modbud:tcp-port="502"
TCP port we are listening on/connecting to (server/client)
    modbus:tcp-allowed="192.9.200.201 192.9.200.202"
(optional) list of allowed clients IP addresses for server mode, if empty all
addresses are allowed
    modbus:tcp-address="192.9.200.201"
server IP address (required in client mode)
    modbus:tcp-keepalive="yes"
should we set TCP Keep-Alive options? "yes" or "no"
    modbus:tcp-timeout="32"
(optional) connection timeout in seconds, after timeout expires, connection
is closed; default empty value means no timeout
    modbus:nodata-timeout="15"
(optional) timeout (in seconds) to set data to 'NO_DATA' if data is not available,
default is 20 seconds
    modbus:nodata-value="-1"
(optional) value to send instead of 'NO_DATA', default is 0
    modbus:FloatOrder="msblsb"
(optional) registers order for 4 bytes (2 registers) float order - "msblsb"
(default) or "lsbmsb"; values names are a little misleading, it should be
msw/lsw (most/less significant word) not msb/lsb (most/less significant byte),
but it's left like this for compatibility with Modbus RTU driver configuration
>
<unit id="1" modbus:id="49">
modbus:id is optional Modbus unit identifier - number from 0 to 255; default is to
use IPK id attribute, but parsed as a character value and not a number; in this
example both definitions id="1" and modbud:id="49" give the same value, because ASCII
code of "1" is 49
    <param
Read value using ReadHoldingRegisters (0x03) Modbus function
        name="..."
        ...
        modbus:address="0x03"
        modbus register number, starting from 0
        modbus:val_type="integer">
        register value type, 'integer' (2 bytes, 1 register) or floo
        2 registers)
        ...
    </param>
    <param
        name="..."
        ...
        modbus:address="0x04"
        modbus:val_type="float"
modbus:val_op="msblsb"
modbus:val_op="LSW">
(optional) operator for converting data from float to 2 bytes integer;
default is 'NONE' (simple conversion to short int), other values
are 'LSW' and 'MSW' - converting to 4 bytes long and getting less/more
significant word; in this case there should be 2 parameters with the
same register address and different val_op attributes - LSW and MSW.
słowa
        ...
    </param>
    ...

```

```

        <send
          Sending value using WriteMultipleRegisters (0x10) Modbus function
            param="..."
            type="min"
            modbus:address="0x1f"
            modbus:val_type="float">
            ...
        </send>
        ...
    </unit>
</device>

```

7.32. Sterownik mbrtudmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Wszelkie urządzenia wykorzystujące protokół Modbus RTU.
- Protokół komunikacji: Modbus RTU.
- Demon przestarzały, zastępuje go uniwersalny demon protokołu Modbus - mbdmn.
- Przykładowa konfiguracja:

```

<device
  xmlns:modbus="http://www.praterm.com.pl/SZARP/ipk-extra"
  daemon="/opt/szarp/bin/mbrtudmn"
  path="/dev/ttyA11"
  modbus:id="0xA1"
  modbus:mode="master"
  'master' or 'slave'
  <unit id="1" modbus:id="0xA11">
    <param
      name="..."
      ...
      modbus:address="0x00"
    modbus:function="0x03"
    modbus function to use (only in master mode)
      modbus:type="integer">
    value type: integer, bcd or float
    ...
  </param>
  <param
    name="..."
    ...
    modbus:address="0x02"
    modbus:type="float"
  modbus:extra="lsb"
  lsb or msb for combined (2 registers) values
  >
  ...
  </param>
  ...

```

```

        <send
            param="..."
            type="min"
            modbus:address="0x1f"
            modbus:type="bcd">
            ...
        </send>
        ...
    </unit>
    <unit id="1" modbus:id="0xA12">
        <param
            name="..."
            ...
            modbus:address="0x00"
            modbus:type="integer">
            ...
        </param>
        <param
            name="..."
            ...
            modbus:address="0x02"
            modbus:type="float">
            ...
        </param>
        ...
        <send
            param="..."
            type="min"
            modbus:address="0x1f"
            modbus:type="float">
            ...
        </send>
        ...
    </unit>
</device>

```

7.33. Sterownik mbtcpdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Wszystkie urządzenia/oprogramowanie komunikujące się przez protokół Modbus TCP/IP.
- Protokół komunikacji: Modbus TCP/IP.
- Demon przestarzały, zastąpiony przez uniwersalny demon protokołu Modbus - mbdmn. Pozwala tylko na jedno jednoczesne połączenie.
- Konfiguracja: Konfiguracja w params.xml, wszystkie atrybuty z przestrzeni nazw 'modbus' nie oznaczone jako opcjonalne są wymagane.
- Przykładowa konfiguracja:

```

<device
    xmlns:modbus="http://www.praterm.com.pl/SZARP/ipk-extra"

```

```

    daemon="/opt/szarp/bin/mbtcpdmn"
    path="/dev/ttyA11"
    modbus:tcp-mode="server"
    allowe modes are 'server' and 'client'
    modbud:tcp-port="502"
TCP port we are listenning on/connecting to (server/client)
port TCP na którym mamy nasłuchiwać/do którego mamy się łączyć(zależnie od trybu)
    modbus:tcp-allowed="192.9.200.201 192.9.200.202"
(optional) list of allowed clients IP addresses for server mode, if empty all
addresses are allowed
    modbus:tcp-address="192.9.200.201"
server IP address (required in client mode)
    modbus:tcp-keepalive="yes"
should we set TCP Keep-Alive options? "yes" or "no"
modbus:tcp-timeout="30"
(optional) connection timeout in seconds, after timeout expires, connection
is closed; default empty value means no timeout
modbus:nodata-timeout="15"
(optional) timeout (in seconds) to set data to 'NO_DATA' if data is not available,
default is 20 seconds
modbus:nodata-value="-1"
(optional) float value to send instead of 'NO_DATA', default is 0
modbus:FloatOrder="msblsb"
(optional) registers order for 4 bytes (2 registers) float order - "msblsb"
(default) or "lsbmsb"; values names are a little misleading, it shoud be
msw/lsw (most/less significant word) not msb/lsb (most/less significant byte),
but it's left like this for compatibility with Modbus RTU driver configuration

    >
    <unit id="1">
        <param
Read value using ReadHoldingRegisters (0x03) Modbus function
            name="..."
            ...
            modbus:address="0x03"
                modbus register number, starting from 0
            modbus:val_type="integer">
                register value type, 'integer' (2 bytes, 1 register) or flo
                2 registers)
            ...
        </param>
        <param
            name="..."
            ...
            modbus:address="0x04"
            modbus:val_type="float"
modbus:val_op="LSW">
(optional) operator for converting data from float to 2 bytes integer;
default is 'NONE' (simple conversion to short int), other values
are 'LSW' and 'MSW' - converting to 4 bytes long and getting less/more
significant word; in this case there should be 2 parameters with the
same register address and different val_op attributes - LSW and MSW.
słowa
            ...
        </param>
        ...

```

```

    <send
      Sending value using WriteMultipleRegisters (0x10) Modbus function
      param="..."
      type="min"
      modbus:address="0x1f"
      modbus:val_type="float">
      ...
    </send>
    ...
  </unit>
</device>

```

7.34. Sterownik mbusdmn

- Zgodność ze specyfikacją: 3 .
- Obsługiwane urządzenia: Dowolne urządzenia korzystające z protokołu M-Bus.
- Protokół komunikacji: M-Bus
- Badanie konfiguracji nowo podłączanego urządzenia

Jednym z głównych powodów stworzenia uniwersalnego demona do komunikacji w protokole M-Bus było ułatwienie konfiguracji i podłączenia do systemu SZARP nowego urządzenia wykorzystującego ten protokół. W związku z powyższym, demona można wykorzystywać jako narzędzie służące do badania konfiguracji nowo podłączonego urządzenia. W tym celu należy zastosować następującą procedurę:

1. Należy uruchomić demona w trybie pracy służącym do testowania komunikacji z urządzeniami M-Bus komendą:

```
/opt/szarp/bin/mbusdmn --test --device <adres_urządzenia> [pozostałe parametry]
```

Parametry linii komend, które przyjmuje *mbusdmn* w trybie testowym mają na celu odwzorowanie możliwości konfiguracyjne, które dają atrybuty elementu *device* z IPK. Dostępne są następujące parametry:

- *device* - parametr obowiązkowy oznaczający nazwę urządzenia, do którego podłączony jest konwerter M-Bus <-> RS232
- *speed* - prędkość transmisji; wartość domyślna: 300 bodów
- *address* - adres protokołu M-Bus urządzenia, które należy odpytać; wartość domyślna: 254, co oznacza odpytanie wszystkich urządzeń, które słuchają na szynie
- *byte_interval* - czas oczekiwania między odczytaniem kolejnych bajtów; wartość ta jest zazwyczaj dobierana eksperymentalnie, indywidualnie dla każdego urządzenia; domyślna wartość to 10 000 mikrosekund
- *data_bits* - ilość bitów danych w ramce RS232; domyślna wartość to 8
- *stop_bits* - ilość bitów stopu w ramce RS232; domyślna wartość to 1
- *parity* - typ parzystości wykorzystywany w ramce RS232; dozwolone wartości to: "even" (parzystość parzysta), "odd" (parzystość nieparzysta), "none" (brak bitu parzystości), domyślna wartość to "none"

W tym trybie pracy daemon powinien wypisać na ekran wiele informacji, które pomogą stworzyć plik konfiguracyjny dla danego urządzenia. Przykładowo fragment z danych wypisanych przez *mbusdmm* w trybie testowania połączenia dla ciepłomierza Pollustat E (pokazano tu jedynie fragment, który dotyczy ramki danych w formacie M-Bus przysyłanej przez ciepłomierz, pominięto fragmenty dotyczące samej komunikacji w protokole M-Bus):

```
Address: 0
```

```
Frame contains data with LSB first
RSP_UD frame parsed as a variable data structure frame
Serial number: 61560223
Manufacturer ID: SPX
Version number: 96
Medium: Heat (at return temperature)
Access number: 10
Decoding status bits:
    Application status bits: 0
    Manufacturer status bits: 0
Signature: 0
```

```
Data:
```

```
    Function: instantaneous value
    Data type: 8 digit BCD
    Storage number: 0
    Value and unit information:
    Value information: Energy [J], * 1 000 000
Value: 870592
```

```
Data:
```

```
    Function: instantaneous value
    Data type: 8 digit BCD
    Storage number: 0
    Value and unit information:
    Value information: Volume [m3], * 0.001
Value: 5802265
```

```
Data:
```

```
    Function: instantaneous value
    Data type: 8 digit BCD
    Storage number: 0
    Value and unit information:
    Value information: Volume Flow [m3/h], * 0.001
Value: 181
```

```
Data:
```

```
    Function: instantaneous value
    Data type: 8 digit BCD
    Storage number: 0
    Value and unit information:
    Value information: Power [W], * 1
Value: 1114
```

```
Data:
```

```
    Function: instantaneous value
    Data type: 4 digit BCD
    Storage number: 0
```

```

Value and unit information:
Value information: Flow Temperature [°C], * 0.1
Value: 580

Data:
Function: instantaneous value
Data type: 4 digit BCD
Storage number: 0
Value and unit information:
Value information: Return Temperature [°C], * 0.1
Value: 526

Data:
Function: instantaneous value
Data type: 6 digit BCD
Storage number: 0
Value and unit information:
Value information: Temperature Difference [K], * 0.001
Value: 5360

Data:
Function: instantaneous value
Data type: 8 digit BCD
Storage number: 0
Value and unit information:
Value information: Fabrication Number
Value: 61560223

Data:
Function: instantaneous value
Data type: 8 digit BCD
Storage number: 0
Value and unit information:
Information from the extended VIF table:
Customer location
Value: 61560223

Data:
Function: maximum value
Data type: special: manufacturer specific data follows, more reco
Manufacturer specific data:

```

Jak widać na powyższym przykładzie, można w ten sposób uzyskać wiele interesujących informacji o danych przesyłanych przez urządzenie. Pierwszą ważną informacją jest adres urządzenia - należy go wykorzystać jako wartość atrybutu *address* w elemencie *unit* konfiguracji. Następnie dowiadujemy się różnych informacji ogólnych o danym urządzeniu - kod producenta, numer seryjny, numer wersji, medium mierzone przez dane urządzenie itp. Następnie podane są informacje o każdej z przesyłanych wartości: typ wartości (chwilowa, maksymalna, minimalna, błędna), sposób jej zakodowania w danych przesyłanych w protokole M-Bus (z tej informacji można prosto wywnioskować zakres wartości, a także transformacje, które należy na niej zastosować, aby była ona jak najbardziej użyteczna w systemie SZARP), kolejny numer wartości danego parametru w pamięci urządzenia (im mniejszy numer, tym nowsza wartość), znaczenie danej wartości (tj. nazwa wielkości, którą ona wyraża) jednostka, w jakiej jest ona wyrażona itp. Czasem mogą również występować informacje dodatkowe, np. o taryfie, w której dana wielkość

była mierzona. Na końcu widać, że dane specyficzne dla producenta danego urządzenia nie są przetwarzane (mogą jedynie zostać wypisane w postaci ciągu liczb szesnastkowych odpowiadających wartościom kolejnych bajtów odebranych od urządzenia).

Jeśli jednak po uruchomieniu demona w trybie testowym nie uzyskamy informacji na temat wartości przesyłanych przez urządzenie (pojawiają się informacje o błędach transmisji lub innego rodzaju komunikaty o błędzie), należy próbować eksperymentalnie dobrać wartości jego parametrów tak, aby udało się nawiązać połączenie z urządzeniem. Poleca się w pierwszej kolejności modyfikować prędkość transmisji oraz odstęp między kolejnymi bajtami, następnie ilość bitów danych, ilość bitów stopu, typ parzystości itp.

2. Na podstawie tak uzyskanych informacji należy stworzyć w pliku `params.xml` odpowiedni element `device`, przypisać dobrane podczas testowania wartości jego atrybutom, stworzyć element `unit` z żądanymi atrybutami oraz utworzyć odpowiednie elementy `param` dla każdej z wartości przesyłanych przez urządzenie. Przykładowy plik konfiguracyjny dla ciepłomierza Pollustat E obsługujący wartości podane powyżej wygląda następująco:

```
<device daemon="/opt/szarp/bin/mbusdmn" speed="2400" path="/dev/ttyA22"
  xmlns:mbus="http://www.praterm.com.pl/SZARP/ipk-extra" mbus:bittime="5000"
  mbus:stopbits="2" mbus:parity="even">
  <unit id="1" type="1" subtype="1" bufsize="1" mbus:address="0">
    <param name="Pollustat M-Bus:Obiegi wewnętrzne:Energia LSW" short_name="Els"
      unit="-" prec="3" mbus:transform="lsw" base_ind="auto">
      <raport title="Pollustat Obiegi wewnętrzne M-Bus" filename="pollustat-mbu"
    </param>
    <param name="Pollustat M-Bus:Obiegi wewnętrzne:Energia MSW" short_name="Ems"
      unit="-" prec="3" mbus:transform="msw" base_ind="auto" mbus:special="pr"
      <raport title="Pollustat Obiegi wewnętrzne M-Bus" filename="pollustat-mbu"
    </param>
    <param name="Pollustat M-Bus:Obiegi wewnętrzne:Objętość LSW" short_name="Vl"
      prec="3" mbus:transform="lsw">
      <raport title="Pollustat Obiegi wewnętrzne M-Bus" filename="pollustat-mbu"
    </param>
    <param name="Pollustat M-Bus:Obiegi wewnętrzne:Objętość MSW" short_name="Vm"
      prec="3" mbus:transform="msw" mbus:special="prev">
      <raport title="Pollustat Obiegi wewnętrzne M-Bus" filename="pollustat-mbu"
    </param>
    <param name="Pollustat M-Bus:Obiegi wewnętrzne:Przepływ" short_name="Pp" dr
      prec="3" base_ind="auto">
      <raport title="Pollustat Obiegi wewnętrzne M-Bus" filename="pollustat-mbu"
    </param>
    <param name="Pollustat M-Bus:Obiegi wewnętrzne:Moc" short_name="PQ" draw_na
      base_ind="auto" mbus:divisor="10">
      <raport title="Pollustat Obiegi wewnętrzne M-Bus" filename="pollustat-mbu"
    </param>
    <param name="Pollustat M-Bus:Obiegi wewnętrzne:Temperatura wejściowa" short
      draw_name="Temp. zasilania" unit="°C" prec="1" base_ind="auto">
      <raport title="Pollustat Obiegi wewnętrzne M-Bus" filename="pollustat-mbu"
    </param>
    <param name="Pollustat M-Bus:Obiegi wewnętrzne:Temperatura wyjściowa" short
      draw_name="Temp. powrotu" unit="°C" prec="1" base_ind="auto">
      <raport title="Pollustat Obiegi wewnętrzne M-Bus" filename="pollustat-mbu"
    </param>
    <param name="Pollustat M-Bus:Obiegi wewnętrzne:Różnica temperatur" short_na
      draw_name="Różn. temperatur" unit="°C" prec="2" base_ind="auto" mbus:di
      <raport title="Pollustat Obiegi wewnętrzne M-Bus" filename="pollustat-mbu"
    </param>
```

```

<param name="Pollustat M-Bus:Obiegi wewnętrzne:Numer fabryczny LSW" short_n
  prec="3" mbus:transform="lsw">
  <raport title="Pollustat Obiegi wewnętrzne M-Bus" filename="pollustat-mbu
</param>
<param name="Pollustat M-Bus:Obiegi wewnętrzne:Numer fabryczny MSW" short_n
  prec="3" mbus:transform="msw" mbus:special="prev">
  <raport title="Pollustat Obiegi wewnętrzne M-Bus" filename="pollustat-mbu
</param>
<param name="Pollustat M-Bus:Obiegi wewnętrzne:dummy" short_name="-" draw_n
</unit>
</device>

```

Po wykonaniu powyższych kroków daemon powinien być gotowy do współpracy z nowo podłączonym urządzeniem oraz z systemem SZARP.

- Konfiguracja: Demona służącego do komunikacji z urządzeniami w protokole M-Bus konfigurujemy, podobnie jak każdego innego daemon wchodzącego w skład systemu SZARP, poprzez plik `params.xml`. W tym celu tworzymy nową sekcję `device` podając w niej odpowiednią ścieżkę do Damona oraz, jako ścieżkę do urządzenia, ścieżkę do konwertera M-Bus <-> RS232, na przykład:

```
<device daemon="/opt/szarp/bin/mbusdmn" path="/dev/ttyS0">
```

Polecane jest również skonfigurowanie odpowiedniej prędkości transmisji poprzez podanie atrybutu `speed`, np.:

```
<device ... speed="300" ... >
```

Prędkość 300 bodów obsługiwana powinna być przez wszystkie urządzenia w protokole M-Bus, wyższe prędkości należy wykorzystywać tylko wtedy, kiedy mamy pewność, że wszystkie urządzenia podłączone do naszej szyny potrafią ją obsłużyć. Można również ustalić odstęp między kolejnymi odczytami danych z urządzenia - należy go podać w sekundach jako wartość parametru `askdelay`.

Wszystkie pozostałe atrybuty konfiguracyjne dotyczące tego demona powinny być umieszczone w przestrzeni nazw <http://www.praterm.com.pl/SZARP/ipk-extra>.

Do najważniejszych parametrów, które można skonfigurować w elemencie `device` należą:

- `byte_interval` - czas oczekiwania między odczytaniem kolejnych bajtów; wartość ta jest zazwyczaj dobierana eksperymentalnie, indywidualnie dla każdego urządzenia; domyślna wartość to 10 000 mikrosekund
- `databits` - ilość bitów danych w ramce RS232; domyślna wartość to 8
- `stopbits` - ilość bitów stopu w ramce RS232; domyślna wartość to 1
- `parity` - typ parzystości wykorzystywany w ramce RS232; dozwolone wartości to: "even" (parzystość parzysta), "odd" (parzystość nieparzysta), "none" (brak bitu parzystości), domyślna wartość to "none"
- `precision` - ilość miejsc po przecinku, które mają być brane pod uwagę przy konwersji liczby zmiennoprzecinkowej (jeśli taka zostanie odebrana od urządzenia) na liczbę stałoprzecinkową; domyślna wartość to 4

Poza tymi atrybutami występują także istotne atrybuty konfiguracyjne w elemencie `unit`, które pozwalają skonfigurować parametry komunikacji dla poszczególnych urządzeń (w przeciwieństwie do parametrów z elementu `device`, które kontrolują globalnie transmisję w całej szynie M-Bus). Oto one:

- *address* - adres na szynie M-Bus urządzenia, z którym chcemy nawiązywać komunikację; domyślny adres to 254, czyli specjalna wartość oznaczająca wysłanie zapytania do wszystkich urządzeń znajdujących się na szynie - powinna ona być używana tylko w trakcie rozpoznawania konfiguracji nowo podłączonego urządzenia, później należy ją zamienić na adres, z którym zgłasza się urządzenie
- *reset* - umożliwia wysłanie do urządzenia po nawiązaniu połączenia żądania wykonania resetu (pomocne przy urządzeniach, których konfiguracja została zmieniona w nieznanym sposób); możliwe wartości tego parametru to:
 - *no* - wartość domyślna, równoznaczna z nieumieszczeniem tego atrybutu w konfiguracji; oznacza brak jakiegokolwiek resetu
 - *full* - wykonuje pełny reset aplikacji w urządzeniu, tzn. zeruje wszystkie ustawienia, liczniki, daty itp.
 - *setup* - wykonuje reset ustawień w urządzeniu, tzn. resetuje ustawienia transmisji, adres na szynie M-Bus oraz ustawione daty
- *select_data* - dokonuje wyboru danych do odczytu z urządzenia; w protokole M-Bus możliwy jest dość rozbudowany wybór danych do odczytu, jednak daemon obsługuje jedynie następujące wartości parametru:
 - *no* - wartość domyślna, równoznaczna z nieumieszczeniem tego atrybutu w konfiguracji; oznacza brak wyboru danych do odczytu, a więc odczyt danych domyślnie przesyłanych przez urządzenie
 - *all* - wybiera do odczytu wszystkie dane zgromadzone w urządzeniu
- *change_address* - adres urządzenia w protokole M-Bus, który ma ono przyjąć po zmianie adresu; domyślna wartość to 0 oznaczająca niewykonanie procedury zmiany adresu
- *reinitialize_on_error* - opcja ta wymusza wykonanie ponownej inicjalizacji z urządzeniem w razie stwierdzenia błędów w transmisji, co jest przydatne w przypadku urządzeń, z którymi komunikacja okresowo ulega zawieszeniu; dozwolone wartości to "yes" oraz "no"

Oprócz tego w każdym elemencie *param* można skonfigurować specjalne operacje (transformacje), które mają być wykonywane na poszczególnych wartościach przesyłanych do systemu SZARP.

Możliwe atrybuty to:

- *multiplier* - liczba przez którą należy pomnożyć otrzymaną wartość
- *divisor* - liczba przez którą należy podzielić otrzymaną wartość
- *modulo* - liczba modulo którą należy wziąć daną wartość
- *transform* - bardziej skomplikowana operacja, którą należy wykonać na parametrze; dostępne operacje to:
 - *lw* - pobranie z parametru 15 najmniej znaczących bitów
 - *mw* - pobranie z parametru bitów 16-30
 - *hw* - pobranie z parametru bitów 31-32.
- *special* - oznacza specjalną operację, którą należy wykonać, aby uzyskać wartość danego parametru; dostępne operacje to:
 - *prev* - wartość tego parametru zostanie wzięta jako wartość poprzedniego parametru odczytanego od urządzenia

Przykładowa konfiguracja w params.xml dla ciepłomierza Landis&Gyr WSD-6 z modułem komunikacyjnym WZD-MB:

- Przykładowa konfiguracja:

```
<device xmlns:mbus="http://www.praterm.com.pl/SZARP/ipk-extra" daemon="/opt/szarj
  path="/dev/ttyS2" speed="300" mbus:byte_interval="50000" mbus:databits="8" m
  mbus:parity="even" options="--askdelay 30240">
  <unit id="1" type="1" subtype="1" bufsize="1" mbus:address="0">
    <param name="Landis Gyr:Węzeł:czas pracy LSW" short_name="OnL" draw_name="--"
      mbus:transform="lw">
      <raport title="Landis Gyr" order="25"/>
    </param>
    <param name="Landis Gyr:Węzeł:czas pracy MSW" short_name="OnM" draw_name="--"
      mbus:transform="mw" mbus:special="prev">
      <raport title="Landis Gyr" order="26"/>
    </param>
    <param name="Landis Gyr:Węzeł:aktualny przepływ z licznika 1 LSW" short_name=
      draw_name="--" unit="--" prec="4" base_ind="auto" mbus:transform="lw">
      <raport title="Landis Gyr" order="1"/>
    </param>
    <param name="Landis Gyr:Węzeł:aktualny przepływ z licznika 1 MSW" short_name=
      draw_name="--" unit="--" prec="4" base_ind="auto" mbus:transform="mw" mbus
      <raport title="Landis Gyr" order="2"/>
    </param>
    <param name="Landis Gyr:Węzeł:temperatura powrotna" short_name="Tpow" draw_n
      unit="°C" prec="0" base_ind="auto">
      <raport title="Landis Gyr" order="8"/>
      <draw title="Landis Gyr" order="2" min="0" max="150" />
    </param>
    <param name="Landis Gyr:Węzeł:time point 1" short_name="--" draw_name="--" uni
    <param name="Landis Gyr:Węzeł:objętość LSW" short_name="VL" draw_name="--" un
      base_ind="auto" mbus:transform="lw">
      <raport title="Landis Gyr" order="15"/>
    </param>
    <param name="Landis Gyr:Węzeł:objętość MSW" short_name="VM" draw_name="--" un
      base_ind="auto" mbus:transform="mw" mbus:special="prev">
      <raport title="Landis Gyr" order="16"/>
    </param>
    <param name="Landis Gyr:Węzeł:numer fabryczny LSW" short_name="NoL" draw_nam
      prec="0" mbus:transform="lw">
      <raport title="Landis Gyr" order="29"/>
    </param>
    <param name="Landis Gyr:Węzeł:numer fabryczny MSW" short_name="NoM" draw_nam
      prec="0" mbus:transform="mw" mbus:special="prev">
      <raport title="Landis Gyr" order="30"/>
    </param>
    <param name="Landis Gyr:Węzeł:temperatura wyjściowa" short_name="Twy"
      draw_name="Temp. wyjściowa" unit="°C" prec="0" base_ind="auto">
      <raport title="Landis Gyr" order="7"/>
      <draw title="Landis Gyr" prior="11" order="1" min="0" max="150"/>
    </param>
    <param name="Landis Gyr:Węzeł:moc wyjściowa z licznika 2" short_name="Qwy2"
      draw_name="Moc z licz. 2" unit="kW" prec="0" base_ind="auto">
      <raport title="Landis Gyr" order="11"/>
  </unit>
</device>
```

```

    <draw title="Landis Gyr" order="8" min="0" max="2000" />
</param>
<param name="Landis Gyr:Węzeł:czas pracy błędnej LSW" short_name="ErL" draw_
    prec="0" mbus:transform="lw">
    <raport title="Landis Gyr" order="27"/>
</param>
<param name="Landis Gyr:Węzeł:czas pracy błędnej MSW" short_name="ErM" draw_
    prec="0" mbus:transform="mw" mbus:special="prev">
    <raport title="Landis Gyr" order="28"/>
</param>
<param name="Landis Gyr:Węzeł:moc wyjściowa z licznika 1 LSW" short_name="Q1
    unit="kW" prec="0" base_ind="auto" mbus:transform="lw">
    <raport title="Landis Gyr" order="9"/>
</param>
<param name="Landis Gyr:Węzeł:moc wyjściowa z licznika 1 MSW" short_name="Q1
    unit="kW" prec="0" base_ind="auto" mbus:transform="mw" mbus:special="pre
    <raport title="Landis Gyr" order="10"/>
</param>
<param name="Landis Gyr:Węzeł:aktualny przepływ z licznika 2" short_name="Gw
    unit="m3/h" prec="1" base_ind="auto">
    <raport title="Landis Gyr" order="3"/>
</param>
<param name="Landis Gyr:Węzeł:maksymalna moc wyjściowa" short_name="Qmax"
    draw_name="Moc maksymalna" unit="kW" prec="0" base_ind="auto">
    <raport title="Landis Gyr" order="12"/>
</param>
<param name="Landis Gyr:Węzeł:data maksimum mocy wyjściowej LSW" short_name=
    draw_name="-" unit="-" prec="0" base_ind="auto" mbus:transform="lw">
    <raport title="Landis Gyr" order="13"/>
</param>
<param name="Landis Gyr:Węzeł:data maksimum mocy wyjściowej MSW" short_name=
    unit="-" prec="0" base_ind="auto" mbus:transform="mw" mbus:special="prev
    <raport title="Landis Gyr" order="14"/>
</param>
<param name="Landis Gyr:Węzeł:energia z licznika 2 LSW" short_name="E2L" dra
    prec="0" base_ind="auto" mbus:transform="lw">
    <raport title="Landis Gyr" order="23"/>
</param>
<param name="Landis Gyr:Węzeł:energia z licznika 2 MSW" short_name="E2M" dra
    prec="0" base_ind="auto" mbus:transform="mw" mbus:special="prev">
    <raport title="Landis Gyr" order="24"/>
</param>
<param name="Landis Gyr:Węzeł:energia sumaryczna z licznika 1 LSW" short_nam
    unit="-" prec="0" base_ind="auto" mbus:transform="lw">
    <raport title="Landis Gyr" order="21"/>
</param>
<param name="Landis Gyr:Węzeł:energia sumaryczna z licznika 1 MSW" short_nam
    unit="-" prec="0" base_ind="auto" mbus:transform="mw" mbus:special="prev
    <raport title="Landis Gyr" order="22"/>
</param>
<param name="Landis Gyr:Węzeł:energia z licznika 1 w taryfie 1 LSW" short_nam
    unit="-" prec="0" base_ind="auto" mbus:transform="lw">
    <raport title="Landis Gyr" order="17"/>
</param>
<param name="Landis Gyr:Węzeł:energia z licznika 1 w taryfie 1 MSW" short_nam
    unit="-" prec="0" base_ind="auto" mbus:transform="lw" mbus:special="prev

```

```

        <raport title="Landis Gyr" order="18"/>
    </param>
    <param name="Landis Gyr:Węzeł:energia z licznika 1 w taryfie 2 LSW" short_name="E1"
        unit="-" prec="0" base_ind="auto" mbus:transform="lw">
        <raport title="Landis Gyr" order="19"/>
    </param>
    <param name="Landis Gyr:Węzeł:energia z licznika 1 w taryfie 2 MSW" short_name="E2"
        unit="-" prec="0" base_ind="auto" mbus:transform="lw" mbus:special="prev">
        <raport title="Landis Gyr" order="20"/>
    </param>
    <param name="Landis Gyr:Węzeł:data maksimum przepływu LSW" short_name="GmxDL"
        prec="0" base_ind="auto" mbus:transform="lw">
        <raport title="Landis Gyr" order="5"/>
    </param>
    <param name="Landis Gyr:Węzeł:data maksimum przepływu MSW" short_name="GmxDM"
        prec="0" base_ind="auto" mbus:transform="mw" mbus:special="prev">
        <raport title="Landis Gyr" order="6"/>
    </param>
    <param name="Landis Gyr:Węzeł:maksymalny przepływ" short_name="Gmax" draw_name="Gmax"
        unit="m3/h" prec="1" base_ind="auto">
        <raport title="Landis Gyr" order="4"/>
    </param>
</unit>
</device>
....

<drawdefinable>
...
    <param name="Landis Gyr:Węzeł:czas pracy" short_name="OnTime" draw_name="-" unit="min"
        <define type="DRAWDEFINABLE" formula="(*:*:czas pracy MSW) (*:*:czas pracy LSW)" />
    </param>
    <param name="Landis Gyr:Węzeł:aktualny przepływ z licznika 1" short_name="Gwyl" draw_name="Gwyl"
        unit="m3/h" prec="3" base_ind="auto">
        <define type="DRAWDEFINABLE"
            formula="(*:*:aktualny przepływ z licznika 1 MSW) (*:*:aktualny przepływ z licznika 1 LSW)" />
        <draw title="Landis Gyr" order="9" min="0" max="100" />
    </param>
    <param name="Landis Gyr:Węzeł:objętość" short_name="V" draw_name="Objętość" unit="m3"
        base_ind="auto">
        <define type="DRAWDEFINABLE" formula="(*:*:objętość MSW) (*:*:objętość LSW)" />
        <draw title="Landis Gyr" order="10" min="0" max="100000" />
    </param>
    <param name="Landis Gyr:Węzeł:numer fabryczny" short_name="No" draw_name="-" unit="int"
        <define type="DRAWDEFINABLE" formula="(*:*:numer fabryczny MSW) (*:*:numer fabryczny LSW)" />
    </param>
    <param name="Landis Gyr:Węzeł:czas pracy błędnej" short_name="ErTime" draw_name="ErTime"
        <define type="DRAWDEFINABLE" formula="(*:*:czas pracy błędnej MSW) (*:*:czas pracy błędnej LSW)" />
    </param>
    <param name="Landis Gyr:Węzeł:moc wyjściowa z licznika 1" short_name="Qwyl" draw_name="Qwyl"
        unit="kW" prec="2" base_ind="auto">
        <define type="DRAWDEFINABLE"
            formula="(*:*:moc wyjściowa z licznika 1 MSW) (*:*:moc wyjściowa z licznika 1 LSW)" />
        <draw title="Landis Gyr" order="7" min="0" max="2000" />
    </param>
    <param name="Landis Gyr:Węzeł:data maksimum mocy wyjściowej" short_name="QmaxD" draw_name="QmaxD"
        prec="0" base_ind="auto">

```

```

    <define type="DRAWDEFINABLE"
        formula="(*:*:data maksimum mocy wyjściowej MSW) (*:*:data maksimum mocy
</param>
<param name="Landis Gyr:Węzeł:energia z licznika 2" short_name="E2" draw_name="E
    prec="0" base_ind="auto">
        <define type="DRAWDEFINABLE" formula="(*:*:energia z licznika 2 MSW) (*:*:en
        <draw title="Landis Gyr" order="6" max="10000000" min="0"/>
</param>
<param name="Landis Gyr:Węzeł:energia sumaryczna z licznika 1" short_name="Ec1"
    unit="kWh" prec="0" base_ind="auto">
        <define type="DRAWDEFINABLE"
            formula="(*:*:energia sumaryczna z licznika 1 MSW) (*:*:energia sumaryczn
            <draw title="Landis Gyr" order="5" min="0" max="10000000"/>
</param>
<param name="Landis Gyr:Węzeł:energia z licznika 1 w taryfie 1" short_name="E1t1
    unit="kWh" prec="0" base_ind="auto">
        <define type="DRAWDEFINABLE"
            formula="(*:*:energia z licznika 1 w taryfie 1 MSW) (*:*:energia z liczn
            <draw title="Landis Gyr" order="3" min="0" max="10000000"/>
</param>
<param name="Landis Gyr:Węzeł:energia z licznika 1 w taryfie 2" short_name="E1t2
    unit="kWh" prec="0" base_ind="auto">
        <define type="DRAWDEFINABLE"
            formula="(*:*:energia z licznika 1 w taryfie 2 MSW) (*:*:energia z liczn
            <draw title="Landis Gyr" order="4" min="0" max="10000000"/>
</param>
<param name="Landis Gyr:Węzeł:data maksimum przepływu" short_name="GmaxD" draw_n
    base_ind="auto">
        <define type="DRAWDEFINABLE" formula="(*:*:data maksimum przepływu MSW) (*:*
</param>
...
</drawdefinable>

```

7.35. Sterownik melsdmn

- Zgodność ze specyfikacją: nieznana .

7.36. Sterownik muksdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Sterownik MUKS-MERPRO kotła olejowego KOG6.

7.37. Sterownik npozytondmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Liczniki energii elektrycznej Pozyton.
- Producent liczników Pozyton nie zezwolił na upublicznienie protokołu komunikacyjnego, wobec czego sterownik ten wymaga zamkniętej wtyczki, dostępnej w bibliotece szarp-prop-plugins.so.
- Konfiguracja: Obsługiwane parametry konfiguracyjne: voltage_factor = dzielnik przekładników napięciowych; current_factor - dzielnik przekładników prądowych; delay = - opóźnienie odpytywania; interface = optolrs485|currloop - typ interfejsu komunikacyjnego, codes - kody funkcji jakie mają być odczytywane.
- Przykładowa konfiguracja:

```

<device daemon="/opt/szarp/bin/npozytondmn" xmlns:pozyton="http://www.praterm.com.p
  path="/dev/ttyS0" pozyton:voltage_factor="60" pozyton:current_factor="200" pozyton
pozyton:codes="0.8.0/1;1.8.0/1;2.8.0/1;3.8.0/1;97.5.1/1;97.5.2/1;97.5.3/1;97.4.1*10;
<unit id="1" type="1" subtype="1" bufsize="1">
<param name="Sieć:LZQM-1:Energia EP plus lsw" .../>
<param name="Sieć:LZQM-1:Energia EP plus msw" .../>
<param name="Sieć:LZQM-1:Energia EP minus lsw" .../>
<param name="Sieć:LZQM-1:Energia EP minus msw" .../>
<param name="Sieć:LZQM-1:Energia EQ plus lsw" .../>
<param name="Sieć:LZQM-1:Energia EQ plus msw" .../>
<param name="Sieć:LZQM-1:Energia EQ minus lsw" .../>
<param name="Sieć:LZQM-1:Energia EQ minus msw" .../>
<param name="Sieć:LZQM-1:Napięcie U1" .../>
<param name="Sieć:LZQM-1:Napięcie U2" .../>
<param name="Sieć:LZQM-1:Napięcie U3" .../>
<param name="Sieć:LZQM-1:Prąd I1" .../>
<param name="Sieć:LZQM-1:Prąd I2" .../>
<param name="Sieć:LZQM-1:Prąd I3" .../>
<param name="Sieć:LZQM-1:Częstotliwość" .../>
<param name="Sieć:LZQM-1:Moc czynna faza 1" .../>
<param name="Sieć:LZQM-1:Moc czynna faza 2" .../>
<param name="Sieć:LZQM-1:Moc czynna faza 3" .../>
<param name="Sieć:LZQM-1:Moc czynna sumaryczna rezerwa" .../>
<param name="Sieć:LZQM-1:Moc bierna faza 1" .../>
<param name="Sieć:LZQM-1:Moc bierna faza 2" .../>
<param name="Sieć:LZQM-1:Moc bierna faza 3" .../>
<param name="Sieć:LZQM-1:Moc bierna sumaryczna rezerwa" .../>
<param name="Sieć:LZQM-1:Moc pozorna faza 1" .../>
<param name="Sieć:LZQM-1:Moc pozorna faza 2" .../>
<param name="Sieć:LZQM-1:Moc pozorna faza 3" .../>
<param name="Sieć:LZQM-1:Moc pozorna sumaryczna rezerwa" .../>
</unit>
</device>

```

7.38. Sterownik nrsdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Z-Elektronik PLC, Sterkom SK-2000 PLC, Sterkom SK-4000 PLC.
- Protokół komunikacji: Z-Elektronik.
- Konfiguracja: For usage instruction use '--help' option. For configuration details see dmncfg.h file; one extra attribute is provided by this daemon - extra:speed attribute for unit element sets special serial port communication speed for given unit. In the following example, default speed is set to 19200 bps, but speed for unit 2 is 9600 bps.

- Przykładowa konfiguracja:

```
<device ... speed="19200">
<unit id="1" .../>
<unit id="2" ... xmlns:extra="http://www.praterm.com.pl/SZARP/ipk-extra"
  extra:speed=9600" ... />
...
```

7.39. Sterownik pafdmn

- Zgodność ze specyfikacją: 2 .
- Obsługiwane urządzenia: Licznik energii elektrycznej Pafal 2EC8.

7.40. Sterownik polludmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Ciepłomierz Polustat-E.
- Protokół komunikacji: Reverse-engineered, Mbus based:

Datum/uhrzeit: #68#09#09#68#53#FE#51#04#6D#30#14#AD#01#05#16

Mittelungszeit: #68#09#09#68#53#FE#51#04#6D#30#14#AD#01#05#16

Mbus primaadresse setzen: #68#09#09#68#53#FE#51#04#6D#30#14#AD#01#05#16

Absolut maxima loshen: #68#09#09#68#53#FE#51#04#6D#30#14#AD#01#05#16

Zahlereinstellungen lesen: #10#40#FE#3E#16, #10#5B#FE#59#16

Meter confirmation: #E5

7.41. Sterownik pozytondmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Licznik energii POZYTON EAP-MS-B13DB76.
- Producent liczników Pozyton nie zezwolił na upublicznienie protokołu komunikacyjnego, wobec czego sterownik ten wymaga zamkniętej wtyczki, dostępnej w bibliotece szarp-prop-plugins.so.
- Konfiguracja: Uruchom z opcją --help aby zobaczyć instrukcję - w szczególności demon wymaga podania w linii polecenia kodów funkcji pytających.

7.42. Sterownik prodmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Sterownik ten może służyć do odczytu danych z systemu PRO 2000. Wymaga programu proxy uruchomionego na maszynie z działającym PRO 2000.
- Protokół komunikacji: Sterownik wykorzystuje prosty protokół tekstowy do wymiany danych z aplikacją proxy działającą na maszynie z uruchomionym PRO 200.
- Wymaga uruchomienia na serwerze z Windows programu obl_ch1.exe.
- Przykładowa konfiguracja:

```
<device daemon="/opt/szarp/bin/prodmn" path="/dev/null" pro2000:address="192.168.1.1"
  xmlns:pro2000="http://www.praterm.com.pl/SZARP/ipk-extra">
  <unit id="1" type="2" subtype="1" bufsize="1">
    <param name="Kocioł 1:DDE:Temperatura wody przed kotłem" short_name="Twe" unit="1"
      pro2000:param="7001" base_ind="auto">
      ....
```

Where:

```
pro2000:address is an IP address of machine running proxy
pro2000:port is a port number at which proxy listens for connections
pro2000:param number of param in pro system
```

7.43. Sterownik ratedmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Sterownik wykonuje obliczenia związane z taryfikowaniem energii elektrycznej na podstawie istniejących parametrów.
- Ten demon jest obecnie niepotrzebny - ten sam efekt może być uzyskany za pomocą parametrów w LUA.

7.44. Sterownik sampledmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: żadne -- jedynie wypływa 1
- Konfiguracja: żadna

7.45. Sterownik sbusdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Liczniki energii używające protokołu S-Bus (Saia-Burgess Bus).

7.46. Sterownik setdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: TCP client, for example SZARP 'setter' program.
- Protokół komunikacji: Daemon uses telnet-like protocol to allow setting values of handled parameters by network client, especially 'setter' program. On first connection, client gets list of handled parameters with their current values and allowed min/max values, one parameter per line in format:

```
"parameter name" [current] [min] [max]
```

Client can set value of parameter with command:

```
"parameter name" [new_value]
```

If value of parameter is changed, all connected clients gets refreshed info about parameter (in format like after first connect). Server can also set error info, one of strings: :E_MIN if new value is less then allowed minimum

:E_MAX if new value is grater then allowed maximum

:E_NOTALLOWED if client is not allowed to connect (based on client IP)

:E_OTHER for other unspecified server error

Error info string can be followed by optional message, that should be presented to user.

- Initial parameter values are read from SZARP database (last assessible values are used).
- Przykładowa konfiguracja:

```
<device daemon="/opt/szarp/bin/setdmn" xmlns:extra="http://www.praterm.com.pl/SZARP/"
path="/dev/null"
extra:tcp-port="8010" <!-- TCP port to listen on -->
extra:tcp-allowed="192.168.1.2" <!-- list of allowed IP addresses, delimited with sp
>
<unit id="1" type="1" subtype="1" bufsize="1">
<param name="Some:Setable:Parameter" ...
```

```

extra:min="0" <!-- minimum allowed value -->
extra:max="27.8" <!-- maximum allowed value -->
> ... </param>
</unit>
</device>

```

7.47. Sterownik sre2dmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Sumator liczników energii elektrycznej Tech-Agro SRE2.
- Konfiguracja: Version without configurable channels (8 channels possible). Available parameters:
 - Energy MSB I
 - Energy LSB I
 - ...
 - Energy MSB VIII
 - Energy LSB VIII
 - Power I
 - ...
 - Power VIII

7.48. Sterownik tcpdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Sterowniki Z-Elektronik, SK-2000 i SK-4000 z wykorzystaniem sieciowego serwera portów szeregowych, np. Moxa NPort W2250/2150 w trybie TCP Server.
- Protokół komunikacji: Z-Elektronik through TCP/IP link.
- Konfiguracja: Konfiguracja w params.xml, atrybuty z przestrzeni nazw 'tcp' są wymagane, chyba że napisano inaczej:
- Przykładowa konfiguracja:

```

<device
  xmlns:tcp="http://www.praterm.com.pl/SZARP/ipk-extra"
  daemon="/opt/szarp/bin/tcpdmn"
  tcp:tcp-ip="192.168.6.10"
    serial ports' server address
  tcp:tcp-port="4001"
serial ports TCP port
  tcp:tcp-keepalive="yes"
  czy połączenie TCP powinno mieć opcję Keep-Alive; dopuszczalne
wartości to "yes" i "no"
  tcp:nodata-timeout="15"
  czas w sekundach, po jakim ustawiamy 'NO_DATA' jeżeli dane nie

```

```

przyszły, opcjonalny - domyślnie 20 sekund
tcp:nodata-value="-1"
wartość (typu float) jaką wysyłamy zamiast 'NO_DATA' dla
parametrów typu send, domyślnie wysyłamy 0
>
<unit id="1">
  <param
    name="..."
    ...
  </param>
  ...
  <send
    param="..."
    type="min"
    ...
  </send>
  ...
</unit>
</device>

```

7.49. Sterownik tensdmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Waga tensometryczna Rowag MWT 6.

7.50. Sterownik testdmn

- Zgodność ze specyfikacją: 2 .
- Obsługiwane urządzenia: Demon czyta nieprzeliczone wartości parametrów z podanego pliku tekstowego, jeden parametr w linii.

7.51. Sterownik unitedmn

- Zgodność ze specyfikacją: 4 .
- Obsługiwane urządzenia: Urządzenia korzystające z protokołu Schneider Electric Uni-Telway.
- Protokół komunikacji: Schneider Electric Uni-Telway
- Demon pracuje w sieci Uni-Telway jako urządzenie slave. Odczytuje dane typu 'WORD' z adresów 0-8191.
- Konfiguracja: "address" attribute of "device" element describes daemon address on UniTE bus. Attributes of "unit" element describes device to read. "address" attribute of "param" element describes register to read.
- Przykładowa konfiguracja:

```

<device
  daemon="/opt/szarp/bin/unitedmn"
  path="/dev/ttyA11"
  speed="9600"
  unitelway:address="6">
<unit id="1" type="1" subtype="1" bufsize="1"
unitelway:network="0x00"
unitelway:station="0xFE"
unitelway:gate="0x00"
unitelway:module="0x00"
unitelway:channel="0x00">
<param ... unitelway:address="1000">
</param>
</unit>
</device>

```

7.52. Sterownik *wlkdmn*

- Zgodność ze specyfikacją: 3 .
- Obsługiwane urządzenia: Daemon do odczytu danych z plików WKL stacji meteorologicznej Vantage Pro 2 firmy Davis Instruments.
- Daemon został zaimplementowany nie jako narzędzie do komunikacji ze stacją bezpośrednio przez port szeregowy/USB, a jako parser plików w formacie WLK, które udostępnia stacja. W związku z tym wymagane jest zewnętrzne oprogramowanie, które odpowiedzialne będzie za komunikację ze stacją oraz pobieranie z niej plików WLK. Dostępne jest oprogramowanie na licencji typu Open Source, które bardzo dobrze realizuje to zadanie: *wview* (<http://www.wviewweather.com/>).

Daemon *wlkdmn* oprócz podstawowej funkcjonalności odczytywania danych bieżących z plików WLK i przekazywania ich do *parcooka*, posiada również możliwość odczytania danych historycznych z tychże plików (np. za podany okres w przeszłości), a następnie wyeksportowania ich do pliku w formacie odpowiednim dla programu *szbwriter* (patrz Sekcja 8.5). Uruchomienie *wlkdmn* w tym trybie pracy następuje poprzez podanie argumentu *--export* w linii poleceń. W tym trybie dostępne są następujące argumenty linii poleceń:

- *--file* - ścieżka do pliku, do którego wyeksportowane dane mają zostać zapisane; jeśli plik ten już istnieje, dane wyeksportowane zostaną dopisane na jego końcu; argument ten jest *obowiązkowy*
- *--from* - data (w formacie RRRR-MM-DD), od której należy zacząć eksport danych; jeśli zostanie pominięta zostaną wyeksportowane dane od najwcześniejszych dostępnych
- *--to* - data (w formacie RRRR-MM-DD), na której należy zakończyć eksport danych; jeśli zostanie pominięta zostaną wyeksportowane dane do najpóźniejszych dostępnych
- Konfiguracja: Demon do działania wymaga poprawnie skonfigurowanego i działającego oprogramowania *wview*. Jako ścieżkę do urządzenia podana powinna zostać ścieżka do katalogu, w którym *wview* przechowuje pliki WLK. W konfiguracji należy umieścić wszystkie 148 parametrów, gdyż taka ilość danych jest przechowywana w plikach WLK; jeżeli dany model stacji nie obsługuje wszystkich parametrów, które mogą być przechowywane w plikach WLK, to zostaną one wypełnione wartością *NO_DATA*.

- Przykładowa konfiguracja:

```

<device daemon="/opt/szarp/bin/wlkdmn" path="/var/wview/archive">
  <unit id="1" type="1" subtype="1" bufsize="1">
    <param name="WView:Simulator:maksymalna dobowa temperatura zewnetrzna" short_name="WView - temperatura zewnetrzna"
      <draw title="WView - temperatury powietrza" min="-40" max="50" prior="1" order="1"/>
      <raport title="WView - simulator" order="1"/>
    </param>
    <param name="WView:Simulator:minimalna dobowa temperatura zewnetrzna" short_name="WView - temperatura zewnetrzna"
      <draw title="WView - temperatury powietrza" min="-40" max="50" order="3"/>
      <raport title="WView - simulator" order="2"/>
    </param>
    <param name="WView:Simulator:maksymalna dobowa temperatura wewnetrzna" short_name="WView - temperatura wewnetrzna"
      <draw title="WView - temperatury powietrza" min="-40" max="50" order="4"/>
      <raport title="WView - simulator" order="3"/>
    </param>
    <param name="WView:Simulator:minimalna dobowa temperatura wewnetrzna" short_name="WView - temperatura wewnetrzna"
      <draw title="WView - temperatury powietrza" min="-40" max="50" order="6"/>
      <raport title="WView - simulator" order="4"/>
    </param>
    <param name="WView:Simulator:średnia dobowa temperatura zewnetrzna" short_name="WView - temperatura zewnetrzna"
      <draw title="WView - temperatury powietrza" min="-40" max="50" order="2"/>
      <raport title="WView - simulator" order="5"/>
    </param>
    <param name="WView:Simulator:średnia dobowa temperatura wewnetrzna" short_name="WView - temperatura wewnetrzna"
      <draw title="WView - temperatury powietrza" min="-40" max="50" order="5"/>
      <raport title="WView - simulator" order="6"/>
    </param>
    <param name="WView:Simulator:maksymalne dobowe ochłodzenie spowodowane wiatrem" short_name="WView - wpływ wiatru, słońca i wilgoci"
      <draw title="WView - wpływ wiatru, słońca i wilgoci" min="-40" max="50" prior="1" order="7"/>
      <raport title="WView - simulator" order="7"/>
    </param>
    <param name="WView:Simulator:minimalne dobowe ochłodzenie spowodowane wiatrem" short_name="WView - wpływ wiatru, słońca i wilgoci"
      <draw title="WView - wpływ wiatru, słońca i wilgoci" min="-40" max="50" order="8"/>
      <raport title="WView - simulator" order="8"/>
    </param>
    <param name="WView:Simulator:maksymalny dobowy punkt rosy" short_name="Trmx" draw_title="WView - punkt rosy"
      <draw title="WView - punkt rosy" min="0" max="50" prior="3" order="1"/>
      <raport title="WView - simulator" order="9"/>
    </param>
    <param name="WView:Simulator:minimalny dobowy punkt rosy" short_name="Trmn" draw_title="WView - punkt rosy"
      <draw title="WView - punkt rosy" min="0" max="50" order="3"/>
      <raport title="WView - simulator" order="10"/>
    </param>
    <param name="WView:Simulator:średnie dobowe ochłodzenie spowodowane wiatrem" short_name="WView - wpływ wiatru, słońca i wilgoci"
      <draw title="WView - wpływ wiatru, słońca i wilgoci" min="-40" max="50" order="11"/>
      <raport title="WView - simulator" order="11"/>
    </param>
    <param name="WView:Simulator:średni dobowy punkt rosy" short_name="Trsr" draw_title="WView - punkt rosy"
      <draw title="WView - punkt rosy" min="0" max="50" order="2"/>
      <raport title="WView - simulator" order="12"/>
    </param>
    <param name="WView:Simulator:maksymalna dobowa zewnetrzna wilgotność" short_name="WView - wilgotność"
      <draw title="WView - wilgotność" min="0" max="100" prior="4" order="1"/>
      <raport title="WView - simulator" order="13"/>
    </param>
    <param name="WView:Simulator:minimalna dobowa zewnetrzna wilgotność" short_name="WView - wilgotność"

```

```

    <draw title="WView - wilgotność" min="0" max="100" order="3"/>
    <raport title="WView - simulator" order="14"/>
</param>
<param name="WView:Simulator:maksymalna dobowa wewnętrzna wilgotność" short_name="wvw"
    <draw title="WView - wilgotność" min="0" max="100" order="4"/>
    <raport title="WView - simulator" order="15"/>
</param>
<param name="WView:Simulator:minimalna dobowa wewnętrzna wilgotność" short_name="wvmin"
    <draw title="WView - wilgotność" min="0" max="100" order="5"/>
    <raport title="WView - simulator" order="16"/>
</param>
<param name="WView:Simulator:średnia dobowa zewnętrzna wilgotność" short_name="wvsn"
    <draw title="WView - wilgotność" min="0" max="100" order="2"/>
    <raport title="WView - simulator" order="17"/>
</param>
<param name="WView:Simulator:maksymalne dobowe ciśnienie atmosferyczne" short_name="wvmaxp"
    <draw title="WView - ciśnienie atmosferyczne" min="700" max="800" prior="5" order="1"/>
    <raport title="WView - simulator" order="18"/>
</param>
<param name="WView:Simulator:minimalne dobowe ciśnienie atmosferyczne" short_name="wvminp"
    <draw title="WView - ciśnienie atmosferyczne" order="3" min="700" max="800" prior="5" order="1"/>
    <raport title="WView - simulator" order="19"/>
</param>
<param name="WView:Simulator:średnie dobowe ciśnienie atmosferyczne" short_name="wvsnp"
    <draw title="WView - ciśnienie atmosferyczne" min="700" max="800" order="2" prior="5" order="1"/>
    <raport title="WView - simulator" order="20"/>
</param>
<param name="WView:Simulator:maksymalna dobowa prędkość wiatru" short_name="wvmaxv"
    <draw title="WView - wiatr" min="0" max="200" prior="6" order="1"/>
    <raport title="WView - simulator" order="21"/>
</param>
<param name="WView:Simulator:średnia dobowa prędkość wiatru" short_name="wvsnv"
    <draw title="WView - wiatr" order="2" min="0" max="200" prior="6" order="1"/>
    <raport title="WView - simulator" order="22"/>
</param>
<param name="WView:Simulator:całkowita dobowa odlegość przebyta przez wiatr" short_name="wvsum"
    <draw title="WView - wiatr" min="0" max="4000" order="3" prior="6" order="1"/>
    <raport title="WView - simulator" order="23"/>
</param>
<param name="WView:Simulator:maksymalna dobowa prędkość 10-minutowa wiatru" short_name="wvmax10"
    <draw title="WView - wiatr" min="0" max="200" order="4" prior="6" order="1"/>
    <raport title="WView - simulator" order="24"/>
</param>
<param name="WView:Simulator:kierunek wiatru o maksymalnej dobowej prędkości" short_name="wvdir"
    <value int="0" name="N"/>
    <value int="1" name="NNE"/>
    <value int="2" name="NE"/>
    <value int="3" name="ENE"/>
    <value int="4" name="E"/>
    <value int="5" name="ESE"/>
    <value int="6" name="SE"/>
    <value int="7" name="SSE"/>
    <value int="8" name="S"/>
    <value int="9" name="SSW"/>
    <value int="10" name="SW"/>
    <value int="11" name="WSW"/>

```

```

    <value int="12" name="W"/>
    <value int="13" name="WNW"/>
    <value int="14" name="NW"/>
    <value int="15" name="NNW"/>
    <raport title="WView - simulator" order="25"/>
  </param>
  <param name="WView:Simulator:kierunek wiatru o maksymalnej dobowej pręđ. 10min"
    <value int="0" name="N"/>
    <value int="1" name="NNE"/>
    <value int="2" name="NE"/>
    <value int="3" name="ENE"/>
    <value int="4" name="E"/>
    <value int="5" name="ESE"/>
    <value int="6" name="SE"/>
    <value int="7" name="SSE"/>
    <value int="8" name="S"/>
    <value int="9" name="SSW"/>
    <value int="10" name="SW"/>
    <value int="11" name="WSW"/>
    <value int="12" name="W"/>
    <value int="13" name="WNW"/>
    <value int="14" name="NW"/>
    <value int="15" name="NNW"/>
    <raport title="WView - simulator" order="26"/>
  </param>
  <param name="WView:Simulator:całkowity dobowy opad deszczu" short_name="hdc" d
    <draw title="WView - deszcz" min="0" max="1000" prior="7" order="1"/>
    <raport title="WView - simulator" order="27"/>
  </param>
  <param name="WView:Simulator:maksymalna dobowa intensywność opadu deszczu" sho
    <draw title="WView - deszcz" min="0" max="200" order="2"/>
    <raport title="WView - simulator" order="28"/>
  </param>
  <param name="WView:Simulator:dobowa dawka promieniowania UV" short_name="UV" d
    <draw title="WView - promieniowanie" min="0" max="5" prior="8" order="1"/>
    <raport title="WView - simulator" order="29"/>
  </param>
  <param name="WView:Simulator:maksymalny dobowy indeks UV" short_name="UVimx" d
    <draw title="WView - promieniowanie" min="0" max="16" order="2"/>
    <raport title="WView - simulator" order="30"/>
  </param>
  <param name="WView:Simulator:czas maksymalnej dobowej temperatury zewnętrznej"
    <raport title="WView - simulator" order="31"/>
  </param>
  <param name="WView:Simulator:czas minimalnej dobowej temperatury zewnętrznej"
    <raport title="WView - simulator" order="32"/>
  </param>
  <param name="WView:Simulator:czas maksymalnej dobowej temperatury wewnętrznej"
    <raport title="WView - simulator" order="33"/>
  </param>
  <param name="WView:Simulator:czas minimalnej dobowej temperatury wewnętrznej"
    <raport title="WView - simulator" order="34"/>
  </param>
  <param name="WView:Simulator:czas maks. dobowego ochłodzenia spow. wiatrem" sh
    <raport title="WView - simulator" order="35"/>
  </param>

```

```
<param name="WView:Simulator:czas min. dobowego ochłodzenia spow. wiatrem" short_name="Czas min. dobowego ochłodzenia spow. wiatrem"
  <raport title="WView - simulator" order="36"/>
</param>
<param name="WView:Simulator:czas maksymalnego dobowego punktu rosy" short_name="Czas maksymalnego dobowego punktu rosy"
  <raport title="WView - simulator" order="37"/>
</param>
<param name="WView:Simulator:czas minimalnego dobowego punktu rosy" short_name="Czas minimalnego dobowego punktu rosy"
  <raport title="WView - simulator" order="38"/>
</param>
<param name="WView:Simulator:czas maksymalnej dobowej zewnętrznej wilgotności" short_name="Czas maksymalnej dobowej zewnętrznej wilgotności"
  <raport title="WView - simulator" order="39"/>
</param>
<param name="WView:Simulator:czas minimalnej dobowej zewnętrznej wilgotności" short_name="Czas minimalnej dobowej zewnętrznej wilgotności"
  <raport title="WView - simulator" order="40"/>
</param>
<param name="WView:Simulator:czas maksymalnej dobowej wewnętrznej wilgotności" short_name="Czas maksymalnej dobowej wewnętrznej wilgotności"
  <raport title="WView - simulator" order="41"/>
</param>
<param name="WView:Simulator:czas minimalnej dobowej wewnętrznej wilgotności" short_name="Czas minimalnej dobowej wewnętrznej wilgotności"
  <raport title="WView - simulator" order="42"/>
</param>
<param name="WView:Simulator:czas maksymalnego dobowego ciśnienia atm." short_name="Czas maksymalnego dobowego ciśnienia atm."
  <raport title="WView - simulator" order="43"/>
</param>
<param name="WView:Simulator:czas minimalnego dobowego ciśnienia atm." short_name="Czas minimalnego dobowego ciśnienia atm."
  <raport title="WView - simulator" order="44"/>
</param>
<param name="WView:Simulator:czas maksymalnej dobowej prędkości wiatru" short_name="Czas maksymalnej dobowej prędkości wiatru"
  <raport title="WView - simulator" order="45"/>
</param>
<param name="WView:Simulator:czas maks. dobowej prędkości 10min wiatru" short_name="Czas maks. dobowej prędkości 10min wiatru"
  <raport title="WView - simulator" order="46"/>
</param>
<param name="WView:Simulator:czas maks. dobowej intensywności opadu deszczu" short_name="Czas maks. dobowej intensywności opadu deszczu"
  <raport title="WView - simulator" order="47"/>
</param>
<param name="WView:Simulator:czas maksymalnego dobowego indeksu UV" short_name="Czas maksymalnego dobowego indeksu UV"
  <raport title="WView - simulator" order="48"/>
</param>
<param name="WView:Simulator:dobowa mapa pogody" short_name="Map" draw_name="Map"
  <raport title="WView - simulator" order="49"/>
</param>
<param name="WView:Simulator:dobowe prawidłowo odebrane pakiety z wiatromierza" short_name="Dobowe prawidłowo odebrane pakiety z wiatromierza"
  <raport title="WView - simulator" order="50"/>
</param>
<param name="WView:Simulator:maksymalna dobowa energia słoneczna" short_name="Maksymalna dobowa energia słoneczna"
  <draw title="WView - promieniowanie" min="0" max="300" order="3"/>
  <raport title="WView - simulator" order="51"/>
</param>
<param name="WView:Simulator:całkowita dobowa energia słoneczna" short_name="Całkowita dobowa energia słoneczna"
  <draw title="WView - promieniowanie" min="0" max="4000" order="4"/>
  <raport title="WView - simulator" order="52"/>
</param>
<param name="WView:Simulator:czas oświetlenia słonecznego" short_name="ts" draw_name="ts"
  <draw title="WView - promieniowanie" min="0" max="1000" order="5"/>
  <raport title="WView - simulator" order="53"/>
</param>
```

```

</param>
<param name="WView:Simulator:sumaryczne dzienne parowanie" short_name="par" draw_n
  <draw title="WView - parowanie" min="0" max="50" prior="9" order="1"/>
  <raport title="WView - simulator" order="54"/>
</param>
<param name="WView:Simulator:maksymalne dobowe ocieplenie spowodowane słońcem"
  <draw title="WView - wpływ wiatru, słońca i wilgoci" min="-40" max="50" orde
  <raport title="WView - simulator" order="55"/>
</param>
<param name="WView:Simulator:minimalne dobowe ocieplenie spowodowane słońcem"
  <draw title="WView - wpływ wiatru, słońca i wilgoci" min="-40" max="50" orde
  <raport title="WView - simulator" order="56"/>
</param>
<param name="WView:Simulator:średnie dobowe ocieplenie spowodowane słońcem" sh
  <draw title="WView - wpływ wiatru, słońca i wilgoci" min="-40" max="50" orde
  <raport title="WView - simulator" order="57"/>
</param>
<param name="WView:Simulator:maksymalna dobowa wartość indeksu THSW" short_nam
  <draw title="WView - indeksy temperaturowe" min="-40" max="50" prior="10" or
  <raport title="WView - simulator" order="58"/>
</param>
<param name="WView:Simulator:minimalna dobowa wartość indeksu THSW" short_name
  <draw title="WView - indeksy temperaturowe" min="-40" max="50" order="2"/>
  <raport title="WView - simulator" order="59"/>
</param>
<param name="WView:Simulator:maksymalna dobowa wartość indeksu THW" short_name
  <draw title="WView - indeksy temperaturowe" min="-40" max="50" order="3"/>
  <raport title="WView - simulator" order="60"/>
</param>
<param name="WView:Simulator:minimalna dobowa wartość indeksu THW" short_name=
  <draw title="WView - indeksy temperaturowe" min="-40" max="50" order="4"/>
  <raport title="WView - simulator" order="61"/>
</param>
<param name="WView:Simulator:ilość stopniodni zimnych" short_name="SDz" draw_n
  <draw title="WView - indeksy temperaturowe" min="0" max="200" order="5"/>
  <raport title="WView - simulator" order="62"/>
</param>
<param name="WView:Simulator:maks. dobowa temperatura z mokrego termometru" sh
  <draw title="WView - wpływ wiatru, słońca i wilgoci" min="-40" max="50" orde
  <raport title="WView - simulator" order="63"/>
</param>
<param name="WView:Simulator:minimalna dobowa temperatura z mokrego termometru
  <draw title="WView - wpływ wiatru, słońca i wilgoci" min="-40" max="50" orde
  <raport title="WView - simulator" order="64"/>
</param>
<param name="WView:Simulator:średnia dobowa temperatura z mokrego termometru"
  <draw title="WView - wpływ wiatru, słońca i wilgoci" min="-40" max="50" orde
  <raport title="WView - simulator" order="65"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku N wiatru" short_name="twN
  <raport title="WView - simulator" order="66"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku NNE wiatru" short_name="t
  <raport title="WView - simulator" order="67"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku NE wiatru" short_name="tw

```

```
<raport title="WView - simulator" order="68"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku ENE wiatru" short_name="t
  <raport title="WView - simulator" order="69"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku E wiatru" short_name="twE
  <raport title="WView - simulator" order="70"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku ESE wiatru" short_name="t
  <raport title="WView - simulator" order="71"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku SE wiatru" short_name="tw
  <raport title="WView - simulator" order="72"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku SSE wiatru" short_name="t
  <raport title="WView - simulator" order="73"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku S wiatru" short_name="twS
  <raport title="WView - simulator" order="74"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku SSW wiatru" short_name="t
  <raport title="WView - simulator" order="75"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku SW wiatru" short_name="tw
  <raport title="WView - simulator" order="76"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku WSW wiatru" short_name="t
  <raport title="WView - simulator" order="77"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku W wiatru" short_name="twW
  <raport title="WView - simulator" order="78"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku WNW wiatru" short_name="t
  <raport title="WView - simulator" order="79"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku NW wiatru" short_name="tw
  <raport title="WView - simulator" order="80"/>
</param>
<param name="WView:Simulator:czas dominacji kierunku NNW wiatru" short_name="t
  <raport title="WView - simulator" order="81"/>
</param>
<param name="WView:Simulator:czas maksymalnej energii słonecznej" short_name="
  <raport title="WView - simulator" order="82"/>
</param>
<param name="WView:Simulator:czas maks. dobowego ocieplenia spow. słońcem" sho
  <raport title="WView - simulator" order="83"/>
</param>
<param name="WView:Simulator:czas min. dobowego ocieplenia spow. słońcem" sho
  <raport title="WView - simulator" order="84"/>
</param>
<param name="WView:Simulator:czas maksymalnej dobowej wartości indeksu THSW" s
  <raport title="WView - simulator" order="85"/>
</param>
<param name="WView:Simulator:czas minimalnej dobowej wartości indeksu THSW" sh
  <raport title="WView - simulator" order="86"/>
</param>
```

```
<param name="WView:Simulator:czas maksymalnej dobowej wartosci indeksu THW" short_name="tTHW" draw_name="tTHW"
  <raport title="WView - simulator" order="87"/>
</param>
<param name="WView:Simulator:czas minimalnej dobowej wartosci indeksu THW" short_name="tTHW_min" draw_name="tTHW_min"
  <raport title="WView - simulator" order="88"/>
</param>
<param name="WView:Simulator:czas maks. dobowej temp. z mokrego termometru" short_name="tTHW_max" draw_name="tTHW_max"
  <raport title="WView - simulator" order="89"/>
</param>
<param name="WView:Simulator:czas min. dobowej temp. z mokrego termometru" short_name="tTHW_min" draw_name="tTHW_min"
  <raport title="WView - simulator" order="90"/>
</param>
<param name="WView:Simulator:ilość stopniodni ciepłych" short_name="SDc" draw_name="SDc"
  <draw title="WView - indeksy temperaturowe" min="0" max="200" order="6"/>
  <raport title="WView - simulator" order="91"/>
</param>
<param name="WView:Simulator:flagi aktualnego rekordu" short_name="fl" draw_name="fl"
  <raport title="WView - simulator" order="92"/>
</param>
<param name="WView:Simulator:czas aktualnego rekordu" short_name="t" draw_name="t"
  <raport title="WView - simulator" order="93"/>
</param>
<param name="WView:Simulator:aktualna temperatura zewnętrzna" short_name="Tz" draw_name="Tz"
  <draw title="WView - temperatury powietrza" min="-40" max="50" order="8"/>
  <raport title="WView - simulator" order="94"/>
</param>
<param name="WView:Simulator:maksymalna temperatura zewnętrzna" short_name="Tz_max" draw_name="Tz_max"
  <draw title="WView - temperatury powietrza" min="-40" max="50" order="7"/>
  <raport title="WView - simulator" order="95"/>
</param>
<param name="WView:Simulator:minimalna temperatura zewnętrzna" short_name="Tz_min" draw_name="Tz_min"
  <draw title="WView - temperatury powietrza" min="-40" max="50" order="9"/>
  <raport title="WView - simulator" order="96"/>
</param>
<param name="WView:Simulator:temperatura wewnętrzna" short_name="Tw" draw_name="Tw"
  <draw title="WView - temperatury powietrza" min="-40" max="50" order="10"/>
  <raport title="WView - simulator" order="97"/>
</param>
<param name="WView:Simulator:ciśnienie atmosferyczne" short_name="Pa" draw_name="Pa"
  <draw title="WView - ciśnienie atmosferyczne" min="700" max="800" order="4"/>
  <raport title="WView - simulator" order="98"/>
</param>
<param name="WView:Simulator:wilgotność zewnętrzna" short_name="fz" draw_name="fz"
  <draw title="WView - wilgotność" min="0" max="100" order="6"/>
  <raport title="WView - simulator" order="99"/>
</param>
<param name="WView:Simulator:wilgotność wewnętrzna" short_name="fw" draw_name="fw"
  <draw title="WView - wilgotność" min="0" max="100" order="7"/>
  <raport title="WView - simulator" order="100"/>
</param>
<param name="WView:Simulator:liczba kliknięć deszczomierza" short_name="nd" draw_name="nd"
  <raport title="WView - simulator" order="101"/>
</param>
<param name="WView:Simulator:maksymalna intensywność opadu deszczu" short_name="nd_max" draw_name="nd_max"
  <draw title="WView - deszcz" min="0" max="4000" order="3"/>
  <raport title="WView - simulator" order="102"/>
</param>
```

```

</param>
<param name="WView:Simulator:prędkość wiatru" short_name="vw" draw_name="Prędkość wiatru"
  <draw title="WView - wiatr" min="0" max="200" order="6"/>
  <raport title="WView - simulator" order="103"/>
</param>
<param name="WView:Simulator:maksymalna prędkość wiatru" short_name="vwx" draw_name="Maksymalna prędkość wiatru"
  <draw title="WView - wiatr" min="0" max="200" order="5"/>
  <raport title="WView - simulator" order="104"/>
</param>
<param name="WView:Simulator:aktualny kierunek wiatru" short_name="Dw" draw_name="Aktualny kierunek wiatru"
  <value int="0" name="N"/>
  <value int="1" name="NNE"/>
  <value int="2" name="NE"/>
  <value int="3" name="ENE"/>
  <value int="4" name="E"/>
  <value int="5" name="ESE"/>
  <value int="6" name="SE"/>
  <value int="7" name="SSE"/>
  <value int="8" name="S"/>
  <value int="9" name="SSW"/>
  <value int="10" name="SW"/>
  <value int="11" name="WSW"/>
  <value int="12" name="W"/>
  <value int="13" name="WNW"/>
  <value int="14" name="NW"/>
  <value int="15" name="NNW"/>
  <raport title="WView - simulator" order="105"/>
</param>
<param name="WView:Simulator:dominujący kierunek wiatru" short_name="Dwd" draw_name="Dominujący kierunek wiatru"
  <value int="0" name="N"/>
  <value int="1" name="NNE"/>
  <value int="2" name="NE"/>
  <value int="3" name="ENE"/>
  <value int="4" name="E"/>
  <value int="5" name="ESE"/>
  <value int="6" name="SE"/>
  <value int="7" name="SSE"/>
  <value int="8" name="S"/>
  <value int="9" name="SSW"/>
  <value int="10" name="SW"/>
  <value int="11" name="WSW"/>
  <value int="12" name="W"/>
  <value int="13" name="WNW"/>
  <value int="14" name="NW"/>
  <value int="15" name="NNW"/>
  <raport title="WView - simulator" order="106"/>
</param>
<param name="WView:Simulator:prawidłowo odebrane pakiety z wiatromierza" short_name="Dwdp" draw_name="Prawidłowo odebrane pakiety z wiatromierza"
  <raport title="WView - simulator" order="107"/>
</param>
<param name="WView:Simulator:aktualna energia słoneczna" short_name="Es" draw_name="Aktualna energia słoneczna"
  <draw title="WView - promieniowanie" min="0" max="300" order="7"/>
  <raport title="WView - simulator" order="108"/>
</param>
<param name="WView:Simulator:maksymalna energia słoneczna" short_name="Esmx" draw_name="Maksymalna energia słoneczna"
  <draw title="WView - promieniowanie" min="0" max="300" order="6"/>

```

```

    <raport title="WView - simulator" order="109"/>
  </param>
  <param name="WView:Simulator:aktualny indeks UV" short_name="UVi" draw_name="I"
    <draw title="WView - promieniowanie" min="0" max="16" order="9"/>
    <raport title="WView - simulator" order="110"/>
  </param>
  <param name="WView:Simulator:maksymalny indeks UV" short_name="UVimx" draw_name="I"
    <draw title="WView - promieniowanie" min="0" max="16" order="8"/>
    <raport title="WView - simulator" order="111"/>
  </param>
  <param name="WView:Simulator:temperatura liści 1" short_name="Tl1" draw_name="T"
    <draw title="WView - temperatury liści i gleby" min="-40" max="50" prior="11" order="1" />
    <raport title="WView - simulator" order="112"/>
  </param>
  <param name="WView:Simulator:temperatura liści 2" short_name="Tl2" draw_name="T"
    <draw title="WView - temperatury liści i gleby" min="-40" max="50" order="2" />
    <raport title="WView - simulator" order="113"/>
  </param>
  <param name="WView:Simulator:temperatura liści 3" short_name="Tl3" draw_name="T"
    <draw title="WView - temperatury liści i gleby" min="-40" max="50" order="3" />
    <raport title="WView - simulator" order="114"/>
  </param>
  <param name="WView:Simulator:temperatura liści 4" short_name="Tl4" draw_name="T"
    <draw title="WView - temperatury liści i gleby" min="-40" max="50" order="4" />
    <raport title="WView - simulator" order="115"/>
  </param>
  <param name="WView:Simulator:dodatkowe informacje o promieniowaniu" short_name="I"
    <raport title="WView - simulator" order="116"/>
  </param>
  <param name="WView:Simulator:prognoza pogody" short_name="Prg" draw_name="Prog"
    <raport title="WView - simulator" order="117"/>
  </param>
  <param name="WView:Simulator:parowanie" short_name="par" draw_name="Parowanie"
    <draw title="WView - parowanie" min="0" max="50" order="2" />
    <raport title="WView - simulator" order="118"/>
  </param>
  <param name="WView:Simulator:temperatura gleby 1" short_name="Tg1" draw_name="T"
    <draw title="WView - temperatury liści i gleby" min="-40" max="50" order="5" />
    <raport title="WView - simulator" order="119"/>
  </param>
  <param name="WView:Simulator:temperatura gleby 2" short_name="Tg2" draw_name="T"
    <draw title="WView - temperatury liści i gleby" min="-40" max="50" order="6" />
    <raport title="WView - simulator" order="120"/>
  </param>
  <param name="WView:Simulator:temperatura gleby 3" short_name="Tg3" draw_name="T"
    <draw title="WView - temperatury liści i gleby" min="-40" max="50" order="7" />
    <raport title="WView - simulator" order="121"/>
  </param>
  <param name="WView:Simulator:temperatura gleby 4" short_name="Tg4" draw_name="T"
    <draw title="WView - temperatury liści i gleby" min="-40" max="50" order="8" />
    <raport title="WView - simulator" order="122"/>
  </param>
  <param name="WView:Simulator:temperatura gleby 5" short_name="Tg5" draw_name="T"
    <draw title="WView - temperatury liści i gleby" min="-40" max="50" order="9" />
    <raport title="WView - simulator" order="123"/>
  </param>

```

```
<param name="WView:Simulator:temperatura gleby 6" short_name="Tg6" draw_name="W"
  <draw title="WView - temperatury liści i gleby" min="-40" max="50" order="10" />
  <raport title="WView - simulator" order="124"/>
</param>
<param name="WView:Simulator:wilgotność gleby 1" short_name="fg1" draw_name="W"
  <draw title="WView - wilgotność liści i gleby" min="0" max="50" order="5"/>
  <raport title="WView - simulator" order="125"/>
</param>
<param name="WView:Simulator:wilgotność gleby 2" short_name="fg2" draw_name="W"
  <draw title="WView - wilgotność liści i gleby" min="0" max="50" order="6"/>
  <raport title="WView - simulator" order="126"/>
</param>
<param name="WView:Simulator:wilgotność gleby 3" short_name="fg3" draw_name="W"
  <draw title="WView - wilgotność liści i gleby" min="0" max="50" order="7"/>
  <raport title="WView - simulator" order="127"/>
</param>
<param name="WView:Simulator:wilgotność gleby 4" short_name="fg4" draw_name="W"
  <draw title="WView - wilgotność liści i gleby" min="0" max="50" order="8"/>
  <raport title="WView - simulator" order="128"/>
</param>
<param name="WView:Simulator:wilgotność gleby 5" short_name="fg5" draw_name="W"
  <draw title="WView - wilgotność liści i gleby" min="0" max="50" order="9"/>
  <raport title="WView - simulator" order="129"/>
</param>
<param name="WView:Simulator:wilgotność gleby 6" short_name="fg6" draw_name="W"
  <draw title="WView - wilgotność liści i gleby" min="0" max="50" order="10"/>
  <raport title="WView - simulator" order="130"/>
</param>
<param name="WView:Simulator:wilgotność liści 1" short_name="fl1" draw_name="W"
  <draw title="WView - wilgotność liści i gleby" min="0" max="15" prior="12" order="1" />
  <raport title="WView - simulator" order="131"/>
</param>
<param name="WView:Simulator:wilgotność liści 2" short_name="fl2" draw_name="W"
  <draw title="WView - wilgotność liści i gleby" min="0" max="15" order="2"/>
  <raport title="WView - simulator" order="132"/>
</param>
<param name="WView:Simulator:wilgotność liści 3" short_name="fl3" draw_name="W"
  <draw title="WView - wilgotność liści i gleby" min="0" max="15" order="3"/>
  <raport title="WView - simulator" order="133"/>
</param>
<param name="WView:Simulator:wilgotność liści 4" short_name="fl4" draw_name="W"
  <draw title="WView - wilgotność liści i gleby" min="0" max="15" order="4"/>
  <raport title="WView - simulator" order="134"/>
</param>
<param name="WView:Simulator:dodatkowa temperatura 1" short_name="Td1" draw_name="W"
  <draw title="WView - temperatury dodatkowe" min="-40" max="50" prior="13" order="1" />
  <raport title="WView - simulator" order="135"/>
</param>
<param name="WView:Simulator:dodatkowa temperatura 2" short_name="Td2" draw_name="W"
  <draw title="WView - temperatury dodatkowe" min="-40" max="50" order="2"/>
  <raport title="WView - simulator" order="136"/>
</param>
<param name="WView:Simulator:dodatkowa temperatura 3" short_name="Td3" draw_name="W"
  <draw title="WView - temperatury dodatkowe" min="-40" max="50" order="3"/>
  <raport title="WView - simulator" order="137"/>
</param>
```

```
<param name="WView:Simulator:dodatkowa temperatura 4" short_name="Td4" draw_name="Td4"
  <draw title="WView - temperatury dodatkowe" min="-40" max="50" order="4"/>
  <raport title="WView - simulator" order="138"/>
</param>
<param name="WView:Simulator:dodatkowa temperatura 5" short_name="Td5" draw_name="Td5"
  <draw title="WView - temperatury dodatkowe" min="-40" max="50" order="5"/>
  <raport title="WView - simulator" order="139"/>
</param>
<param name="WView:Simulator:dodatkowa temperatura 6" short_name="Td6" draw_name="Td6"
  <draw title="WView - temperatury dodatkowe" min="-40" max="50" order="6"/>
  <raport title="WView - simulator" order="140"/>
</param>
<param name="WView:Simulator:dodatkowa temperatura 7" short_name="Td7" draw_name="Td7"
  <draw title="WView - temperatury dodatkowe" min="-40" max="50" order="7"/>
  <raport title="WView - simulator" order="141"/>
</param>
<param name="WView:Simulator:dodatkowa wilgotność 1" short_name="fd1" draw_name="fd1"
  <draw title="WView - wilgotności dodatkowe" min="0" max="100" prior="14" order="1"/>
  <raport title="WView - simulator" order="142"/>
</param>
<param name="WView:Simulator:dodatkowa wilgotność 2" short_name="fd2" draw_name="fd2"
  <draw title="WView - wilgotności dodatkowe" min="0" max="100" order="2"/>
  <raport title="WView - simulator" order="143"/>
</param>
<param name="WView:Simulator:dodatkowa wilgotność 3" short_name="fd3" draw_name="fd3"
  <draw title="WView - wilgotności dodatkowe" min="0" max="100" order="3"/>
  <raport title="WView - simulator" order="144"/>
</param>
<param name="WView:Simulator:dodatkowa wilgotność 4" short_name="fd4" draw_name="fd4"
  <draw title="WView - wilgotności dodatkowe" min="0" max="100" order="4"/>
  <raport title="WView - simulator" order="145"/>
</param>
<param name="WView:Simulator:dodatkowa wilgotność 5" short_name="fd5" draw_name="fd5"
  <draw title="WView - wilgotności dodatkowe" min="0" max="100" order="5"/>
  <raport title="WView - simulator" order="146"/>
</param>
<param name="WView:Simulator:dodatkowa wilgotność 6" short_name="fd6" draw_name="fd6"
  <draw title="WView - wilgotności dodatkowe" min="0" max="100" order="6"/>
  <raport title="WView - simulator" order="147"/>
</param>
<param name="WView:Simulator:dodatkowa wilgotność 7" short_name="fd7" draw_name="fd7"
  <draw title="WView - wilgotności dodatkowe" min="0" max="100" order="7"/>
  <raport title="WView - simulator" order="148"/>
</param>
</unit>
</device>
```

7.53. Sterownik zmdm

- Zgodność ze specyfikacją: 2 .
- Obsługiwane urządzenia: Licznik energii elektrycznej LandisGyr ZMD405CT44.2407.

- Konfiguracja: Demon może na zadany okres czasu zostać dezaktywowany (zamyka port, wstawia do segmentu pamięci parcook'a wartości SZARP_NO_DATA śpi). Okresy dezaktywacji demona określa się za pomocą parametrów linii komend. Np. wywołanie programu w sposób następujący:

```
zmddmn 3 /dev/ttyS0 -t 12:00 10 -t 18:05 13
```

spowoduje, że w godzinach 12:00-12:10 oraz 18:05-18:18 demon nie będzie aktywny. Ilość okresów dezaktywacji jest dowolna.

8. Format bazy SzarpBase

SzarpBase to nazwa nowego formatu bazy danych historycznych SZARP. Zastosowano prosty format plikowy, który ma ułatwić operacje na bazie (kopiowanie, modyfikację i usuwanie danych), uprościć aplikacje korzystające z bazy a także uniezależnić system SZARP od ostatniego komercyjnego elementu, jakim była biblioteka CodeBase.

8.1. Format plików bazy

Baza jest trzymana w drzewie katalogów o strukturze Grupa/Jednostka/Parametr, gdzie Grupa, Jednostka i Parametr to odpowiednie części pełnej nazwy parametru. Każdy plik zawiera dane o próbkach 10-minutowych jednego parametru z jednego miesiąca. Nazwa pliku jest tworzona na podstawie daty (roku i miesiąca), którego dotyczą dane.

Pełna ścieżka do pliku bazy składa się z:

1. Nazwy parametru poddanej konwersji - wszystkie znaki ASCII o kodach spoza zakresów 48 (0) - 49 (9), 65 (A) - 90 (Z), 97 (a) - 122 (z) (czyli wszystko poza cyframi i literami alfabetu łacińskiego) zamieniane są na znak o kodzie 64 (_), poza polskimi literami, które zamieniane są na odpowiedniki z alfabetu angielskiego (np. 'ą' na 'a' itp.) i poza znakiem ':' który zamieniany jest na '/'.
2. Znak /.
3. Daty (miesiąca) w formacie RRRRMM, gdzie RRRR to rok (cztery cyfry), a MM to miesiąc (dwie cyfry).
4. Kropki (.)
5. Rozszerzenia *szb*.

Przykładowo informacje o parametrze *Kociół 1: Sterownik: temperatura zadana* z lutego 2003 roku będą trzymane w pliku o ścieżce dostępu *Kociol_1/Sterownik/temperatura_zadana/200302.szb*.

UWAGA! Sposób konwersji wymusza ostrzejsze wymagania dotyczące nazw parametrów niż sam format konfiguracji IPK - muszą one być różne także po konwersji!

Wszystkie daty dotyczące bazy odnoszą się do strefy czasowej Greenwich (UTC).

Każdy plik składa się tablicy liczb typu short int (2 bajty, Little Endian) ze znakiem, będących wartościami próbek parametru. Tablica może mieć długość taką, ile próbek mieści się w danym miesiącu, np. dla lutego 2003 (28 dni) i próbek 10-minutowych będzie to $6*24*28 = 4032$ liczb. Specjalna wartość -32768 oznacza brak danych. Danych na końcu pliku może brakować, 'dziury' w środku muszą być

wypełniane wartością -32768. Z formatu pliku nie wynika bezpośrednio, co ile zbierane są próbki. W bazie nie są zapisywane żadnego rodzaju średnie.

Standardowo baza w nowym formacie znajduje się w katalogu `/opt/szarp/prefix/szbase`.

8.2. Konwersja bazy na nowy format

Konwersja istniejącej bazy danych na nowy format nie jest trudna. Czynności do wykonania na serwerze to po kolei:

1. *Instalacja najnowszej wersji SZARP* - starsze mogą nie obsługiwać nowego formatu bazy lub posiadać nieusunięte błędy. Upgrade SZARP'a będzie też zwykle potrzebny wszędzie, gdzie baza ma być oglądana.
2. *Wyłączenie parstarta* - zalecane, będziemy operować na bazie i konfiguracji. Cała operacja nie powinna trwać zbyt długo, więc dziura w bazie raczej nie powinna przekroczyć 1 lub 2 próbek.
3. *Konwersja bazy* - uruchamiamy program `cb2szb`, jeżeli trzeba to z odpowiednią opcją `'-Dprefix=nazwa'`. Program wypisuje stopień zaawansowania konwersji. Stara baza nie jest kasowana. Konwersję można przerwać w dowolnej chwili, program wypisze wtedy tak zwany znacznik czasowy (timestamp), który można użyć wraz z opcją `'-t'` do kontynuowania konwersji w momencie kiedy została przerwana. W przypadku błędów w bazie należy uruchomić program z przełącznikiem `'-m'`, spowoduje to zastąpienie błędnych próbek 10-minutowych wartościami średnimi (8-godzinnymi lub - gdyby te także okazały się błędne - dziennymi).

Aby uniknąć konieczności przesyłania całej wygenerowanej nowej bazy przez mechanizm BODAS, można konwersję równolegle uruchomić na innych komputerach, gdzie baza ma być przegrana, np. na serwerze `praterm.com.pl`.

Na uszkodzonej bazie sztu zaobserwowano objaw w postaci utworzenia plików o dacie z 1969 roku. Pliki te należało skasować, aby działał program przeglądający.

4. *Zmiana indeksów na "auto"*. Operacja ta jest opcjonalna, ale pozbycie się tradycyjnych indeksów było jednym z powodów powstania nowego formatu bazy, więc warto to zrobić. Wszystkie wartości atrybutów `base_ind` w pliku konfiguracyjnym `params.xml` należy zamienić na `"auto"`. Od tego momentu na konfiguracji nie można już wywoływać `szarp2ipk` - zobacz też Sekcja 5.3.4. Następnie można wygenerować konfigurację w formacie SZARP 2.1 za pomocą `ipk2szarp` (bez opcji `'-o'!`).
5. *Zmiana konfiguracji parstart.cfg*. Plik `/etc/szarp/parstart.cfg` należy zmodyfikować tak, aby zamiast programu `meaner` uruchamiany był `meaner3` - odpowiednie zmienne powinny mieć wartość:


```
MEANER=0
MEANER3=1
```
6. *Kontrola poprawności szarp.cfg*. Plik `/etc/szarp/szarp.cfg` powinien zawierać deklaracje parametrów wymaganych przez program `meaner3` oraz program przeglądający korzystający z nowej bazy. Najprościej osiągnąć to przez upewnienie się, że na początku pliku znajduje się dyrektywa `$include` wczytująca najnowszą wersję pliku `szarp_in.cfg`. Warto sprawdzić także, czy mechanizm BODAS przegra katalogi `szbase`.
7. *Ponowne uruchomienie parstarta*. I to powinno być wszystko. W razie czego informacji o przyczynach błędów należy szukać w logach. Zobacz też Sekcja 9.

8.3. Program extrszb

Program extrszb jest tekstowym narzędziem służącym do odczytywania danych z bazy w formacie SzarpBase i konwersji ich do postaci, która może być wczytana przez arkusz kalkulacyjny.

Program extrszb pozwala na wczytanie danych z bazy w formacie SzarpBase do arkusza kalkulacyjnego. Jest to narzędzie uruchamiane z linii poleceń. Argumentami programu są nazwy parametrów, dla których mają zostać odczytane dane. Program wypisuje wyniki działania na standardowe wyjście lub do zadanego pliku. Na wyjściu program produkuje tabelę, której kolumny odpowiadają parametrom, a w wierszach znajdują się kolejne wartości próbek parametrów - wiersze są posortowane po czasie. Odczytywanie parametrów z bazy można przerwać przez wciśnięcie **Ctrl-C**. Opcje i argumenty mogą być podane w dowolnej kolejności, chyba że coś innego wynika z ich opisów. Możliwe są następujące:

Notatka: Najbardziej aktualny spis możliwych opcji można uzyskać uruchamiając program z opcją `--help`.

• Wybór parametrów

Wyboru parametrów można dokonać na jeden z dwóch sposobów:

- Podając pełne nazwy parametrów jako **argumenty** do programu. Można podać dowolną liczbę parametrów. Ponieważ w większości wypadków nazwy parametrów zawierają spacje, trzeba zamknąć je w cudzysłowy.
- Używając opcji `--file=<NAME>` lub `-f <NAME>`, której argumentem jest nazwa pliku zawierającego nazwy parametrów, po jednej nazwie na linię. Opcję można podać wiele razy i można ją łączyć z podawaniem parametrów za pomocą argumentów programu.

Program sprawdza czy parametry o podanych nazwach istnieją w konfiguracji, jeżeli nie, wypisuje komunikat o nazwie błędnego parametru i kończy działanie z błędem. Konieczne jest podanie przynajmniej jednego parametru, za pomocą dowolnego z powyższych sposobów. Niepodanie żadnego parametru powoduje zakończenie programu z błędem.

• Wybór zakresu czasowego

Zakres czasu dla jakiego mają być wypisane dane można określić na jeden z dwóch sposobów - albo podając czas początkowy i końcowy, albo określając miesiąc dla jakiego mają być wypisane dane. Podanie zakresu czasowego na jeden z tych sposobów jest wymagane.

- Opcja `--date-format=<STRING>` lub `-D <STRING>` określa format, w jakim ma być podana data dla opcji `-s` i `-e`. Format powinien być zgodny z zawartym w opisie funkcji *strptime* (zobacz **man strptime**). Domyślnie jest to „%Y-%m-%d %H:%M”, czyli data podawana jest w postaci np. „2004-03-11 23:50”. Format dotyczy opcji które występują w linii komend za nim, a przed kolejną zmianą formatu. W ten sposób można podać różne formaty dla daty początkowej i końcowej (dwukrotnie używając opcji `-D`).
- Opcja `--start=<DATE>` lub `-s <DATE>` określa datę i czas pierwszej próbki, jaka zostanie odczytana z bazy. Podana data może ulec zaokrągleniu zależnie od zażądanego typu średniej (np. jeżeli chcemy średnie miesięczne, liczyć się będzie tylko podany rok i miesiąc). Jeżeli podamy tą opcję, wymagana jest także opcja `-e`. Format daty jest określany przez opcję `-D` lub domyślny

(zobacz wyżej). Podany czas jest interpretowany zgodnie ze strefą czasową ustawioną w systemie operacyjnym.

- Opcja **--end=<DATE>** lub **-e <DATE>** określa datę i czas pierwszej próbki, która **nie zostanie uwzględniona** przy odczytywaniu danych. W przypadku próbek 10-minutowych należy więc podać czas o 10 minut późniejszy niż ostatnia dana, jaką chcemy odczytać. Podany czas musi być późniejszy niż podany czas pierwszej próbki, także po zaokrągleniu wynikającym z typu zażądaney średniej. Zobacz też uwagi do poprzedniej opcji.
- Opcja **--month=<YYYYMM>** lub **-m <YYYYMM>** umożliwia podanie zakresu dat w alternatywny sposób. Jej argumentem jest napis składający się z sześciu cyfr określających kolejno rok a następnie miesiąc, który nas interesuje - np. „200311” to listopad 2003 roku. Inaczej niż w przypadku poprzednich opcji, czas nie jest interpretowany względem ustawionej strefy czasowej. Wypisywana jest zawartość pliku bazy SzarpBase odnosząca się do próbek z podanego miesiąca, a więc czas odnosi się do strefy Greenwich. Stąd też zależnie od tego czy mamy do czynienia z czasem letnim czy zimowym, na wyjściu możemy otrzymać próbki np. od godziny 1:00 pierwszego dnia miesiąca, do godziny 0:50 pierwszego dnia następnego miesiąca (bo daty na wyjściu podane są w lokalnej strefie czasowej).

Opcji **-s** i **-e** nie można łączyć z **-m** - należy wybrać jeden ze sposobów określania zakresu czasowego.

• Typ wypisywanych próbek

- Opcja **--probe=<TYPE>** lub **-p <TYPE>** określa typ próbki (rodzaj średniej), jaka będzie wypisywana. Możliwe są następujące:
 - *10min* - pojedyncza próbka, czyli średnia z 10 minut, jest to wartość domyślna.
 - *hour* - średnia godzinna.
 - *8hour* - średnia z 8 godzin.
 - *day* - średnia z dnia.
 - *week* - średnia z tygodnia.
 - *month* - średnia z miesiąca.
 - liczba - długość próbki w sekundach, np. dla liczenia średnich 12 godzinnych powinniśmy podać wartość $60 * 60 * 12$ czyli *43200*.

Typ wybranej próbki ma wpływ zarówno na interpretację ustawionej daty końca i początku danych, jak i na postać wypisywanej daty.

- **Format wyjścia** Program umożliwia wypisywanie danych w jednym z kilku wybranych formatów. Domyślnie dane wypisywane są na standardowe wyjście w formacie CSV - pola tekstowe oddzielone przecinkami. Wiersze zawierające tylko wartości BRAK DANYCH nie są drukowane. Wartości te w pozostałych wierszach są zastępowane przez napis *NO_DATA*. Domyślne ustawienia można zmienić jedną z następujących opcji:

- **--csv** lub **-c** ustawia format danych wyjściowych na CSV, jest to ustawienie domyślne.
- **--xml** lub **-x** ustawia format danych na XML. Na wyjściu produkowany jest dokument o formacie określonym przez schemat RelaxNG z pliku *resources/dtd/szbase-extr.rng*. Format pliku jest dość prosty, oto przykład:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<extracted xmlns="http://www.praterm.com.pl/SZARP/extr" raw="no">
```

```

<header>
  <param>Kocioł 1:Sterownik:temperatura zadana</param>
  <param>Kocioł 1:Sterownik:temperatura wody za kotłem</param>
</header>
<data>
  <row>
    <time>2003-11-02 10:00</time>
    <value>105.0</value><value>22.7</value>
  </row>
  <row>
    <time>2003-11-02 10:10</time>
    <value>105.0</value><value>22.6</value>
  </row>
  <row>
    <time>2003-11-02 10:20</time>
    <value>105.0</value><value>22.7</value>
  </row>
  <row>
    <time>2003-11-02 10:30</time>
    <value>105.0</value><value>22.7</value>
  </row>
  <row>
    <time>2003-11-02 10:40</time>
    <value>105.0</value><value>22.7</value>
  </row>
  <row>
    <time>2003-11-02 10:50</time>
    <value>105.0</value><value>22.7</value>
  </row>
</data>
</extracted>

```

- **--openoffice** lub **-O** powoduje wygenerowanie pliku w formacie OpenOffice Calc (arkusza kalkulacyjnego z pakietu OpenOffice). Wybranie tego formatu zmusza także do określenia pliku wyjściowego (za pomocą opcji **-o**) - nie jest możliwe wypisanie pliku na standardowe wyjście.
- **--output=<FILE>** lub **-o <FILE>** ustawia plik wyjściowy (zamiast standardowego wyjścia). Opcja ta implikuje **-O** (format wyjściowy OpenOffice Calc), chyba że zostanie ustawiony inny format za pomocą opcji **-c** lub **-x**.
- **--no-data=<TEXT>** lub **-N <TEXT>** ustawia tekst, jaki będzie wypisywany zamiast wartości BRAK DANYCH, domyślnie jest to *NO_DATA*.
- **--delimiter=<TEXT>** lub **-d <TEXT>** ustawia separator dla formatu CSV (domyślnie są to dwa znaki - przecinek i spacja). Użycie opcji implikuje format CSV, nie można jej więc łączyć z **-O** lub **-x**.
- **--empty** lub **-E** powoduje, że wypisywane będą także wiersze zawierające same wartości BRAK DANYCH - domyślnie wiersze takie są opuszczane.
- **--raw** lub **-r** określa, że wartości parametrów mają być wypisane bezpośrednio w takiej postaci, w jakiej znajdują się w bazie, bez konwersji wynikającej z zadeklarowanej precyzji parametrów. Domyślnie konwersja taka jest wykonywana.
- **Wypisywanie postępu konwersji**
 - Opcja **--progress** lub **-P** powoduje, że na `stderr` wypisywane będą informacje o postępach w konwersji w szczególności przybliżone procentowe zaawansowanie konwersji (określane na

podstawie przedziału czasowego jaki pozostał). Po zakończeniu właściwej konwersji program może wypisać informację o kompresji pliku OpenOffice. Program informuje też o zakończeniu działania wypisując komunikat *Completed*.

- **Ustawianie zmiennych libpar**

- Tradycyjnie dla programów SZARP opcja **-D<name>=<value>** pozwala na ustalenie wartości zmiennej biblioteki libpar. Zwykle będzie potrzeba wybrania bazy danych przez użycie opcji typu *-Dprefix=prza*.

- **Pozostałe opcje**

- **--debug=<LEVEL>** - ustala poziom ważności wypisywanych komunikatów, od 0 (tylko poważne błędy) do 10 (wszystko), domyślnie jest to 2.
- **--help** lub **-?** - wypisuje pomoc dla programu.
- **--usage** - wypisuje skróconą pomoc dla programu - spis opcji i argumentów.
- **--version** lub **-V** - wypisuje informację o wersji programu.

Program korzysta z biblioteki libpar, wczytuje za jej pomocą następujące parametry:

- **IPK** - ścieżka do pliku konfiguracyjnego z formacie IPK, parametr wymagany, czytany z sekcji globalnej.
- **datadir** - główny katalog z bazą w formacie SzarpBase, parametr wymagany, czytany z sekcji globalnej.
- **oo_script** - ścieżka do skryptu kompresującego podany plik XML do dokumentu OpenOffice'a. Parametr jest czytany z sekcji *extrszb*, nie jest wymagany, ale bez niego nie będzie można generować dokumentów OpenOffice.

8.4. Program szbedit

Program szbedit służy do edycji plików bazy SzarpBase - dodawania, usuwania i modyfikowania danych. Przyjmuje jeden argument - ścieżkę do pliku z danymi. Do otwarcia pliku użytkownik musi mieć prawa zapisu do niego.

Jeżeli nazwa pliku zawiera rok i miesiąc, program na tej podstawie jest w stanie wyświetlając dane podawać czas lokalny. Jeśli nie, to podaje czas w strefie czasowej UTC.

Opis elementów ekranu i sposobu obsługi można uzyskać po uruchomieniu programu i naciśnięciu klawisza **F1**.

8.5. Import danych do bazy - program szbwriter

Program szbwriter jest aplikacją umożliwiającą wydajne importowanie do bazy w formacie SzarpBase danych z innego źródła. Umożliwia:

- Automatyczne dodawanie do konfiguracji nieistniejących w niej parametrów.
- Automatyczne tworzenie nazw wykresów, jednostek, okien wykresów i nazw skróconych.
- Wyliczanie zakresu wartości dla wykresów parametrów.
- Obsługę parametrów o wartościach przechowywanych w dwóch słowach.

Program został wykorzystany do realizacji importu do SZARP'a danych z regulatorów węzłów obsługiwanych przez system InTouch lub Mikrob PRO-2000 (zobacz Sekcja 10).

8.5.1. Konfiguracja programu

Program korzysta z biblioteki libpar (Sekcja 4). Wymaga czytanych z sekcji *meaner3* lub globalnej parametrów *datadir*, *IPK* i czytanych z sekcji *szbwriter* opcjonalnych *log* oraz *log_level*. Opis znaczenia tych parametrów znajduje się w rozdziale Sekcja 9.1. Można oczywiście podawać także w linii komend parametry inicjalizujące zmienne libpar, takie jak *-Dprefix=xxx*.

Program korzysta z opcjonalnego parametru *double_match*, czytanego z sekcji *szbwriter*. Parametr ten ma postać wzorca powłoki (shell pattern) opisującego nazwy parametrów, które mają być zapisywane w dwóch słowach (4 bajty). Przykładowo, dla węzłów Samson, parametr będzie miał zwykle postać:

```
:szbwriter
double_match=@(*-energia|*-objetosc)
```

Więcej informacji o wzorcach powłoki można uzyskać wydając komendę **info fnmatch**.

W tej samej sekcji pliku *szarp.cfg* może się znajdować parametr *fill_how_many*. Jego wartością jest liczba, określająca ile ewentualnych pustych próbek należy przy zapisie zastąpić aktualnie zapisywaną wartością. Przydatny jest do likwidowania 'dziur w wykresach', występujących w sytuacji gdy przerwy między danymi wejściowymi wynoszą więcej niż 10 minut. Domyślna wartość parametru to 0. Oprócz wypełniania 10-minutowych luk, uzupełniane są również odpowiadające im 10-sekundowe próbki. Można wyłączyć zapisywanie 10-sekundowych próbek, uruchamiając program z parametrem *-p*.

W linii komend można podać jeden, opcjonalny, parametr - nazwę konfiguracji. Wykorzystywany jest on tylko, jeżeli konfiguracja opisana parametrem *IPK* nie istnieje i będzie tworzona od nowa.

Dodatkowe opcje programu to *-n*, wymuszająca dodanie nowo napotkanych parametrów do konfiguracji i zapis tak zmodyfikowanej konfiguracji do pliku *params.xml*, oraz *-p*, wyłączająca zapisywanie obok zwykłych próbek 10-minutowych także próbek 10-sekundowych (o ile są skonfigurowane w pliku *szarp.cfg*).

Listę opcji i sposób konfiguracji programu można wyświetlić uruchamiając program z opcją *--help*.

8.5.2. Dane wejściowe

Program czyta dane ze standardowego wejścia. Spodziewa się linii następującej postaci:

```
<nazwa parametru> <rok> <miesiac> <dzien> <godzina> <minuta> <wartość>
```

Jeżeli któreś z pól ma zawierać spację, to powinno być wzięte w podwójny cudzysłów. Znak cudzysłowu może się pojawić, o ile zostanie poprzedzony znakiem '\\'. Przykład:

```
"Bardzo:fajna:nazwa z \"cudzysłowem\"" 2004 10 28 20 19 6.38
```

Dodatkowo, jeżeli nazwa parametru kończy się wyrażeniem w nawiasach kwadratowych, to przyjmuje się że wyrażenie w nawiasach jest nazwą jednostki dla danego parametru. Nazwa jednostki dłuższa niż 5 znaków zostanie obcięta do pierwszych 5 znaków.

Program wczytuje kolejne linie z wejścia. Jeżeli parametr o podanej nazwie nie istnieje w konfiguracji a użyto opcji -n, jest do niej dodawany jako parametr definiowalny.

Jeżeli nazwa parametru pasuje do wzorca dla parametrów dwusłownych (patrz Sekcja 8.5.1), to precyzja parametru jest ustawiana na 0, tworzone są dwa parametry definiowalne o nazwach powstałych przez dodanie do nazwy z wejścia napisów 'lsw' i 'msw', a sam parametr jest dodawany jako parametr definiowalny przeglądającego, z odpowiednią formułą sklejającą dwa parametry definiowalne.

Jeżeli nowy parametr nie jest "dwusłowny", to jego precyzja jest ustalana na podstawie ilości miejsc po przecinku w polu wartości.

Zarówno ustalanie, czy parametr ma być zapisywany w dwóch słowach, jak i ustalanie precyzji parametru odbywa się tylko raz - przy pierwszym wystąpieniu parametru na wejściu. Nie jest możliwa zmiany tych własności później, gdyż wymagałoby to konwersji zapisanych wcześniej do bazy danych.

Podana wartość jest zapisywana do bazy, przy czym jeżeli bezpośrednio po sobie wystąpi więcej wartości, które mieszczą się w jednej 10-minutowej próbce (a więc np. wartość z 10:10 i 10:15), to do bazy zostanie zapisana ich średnia.

Jeżeli skonfigurowano zapisywanie próbek 10-sekundowych za pomocą programu prober, oraz nie użyto opcji -p, to program zapisze także próbki 10-sekundowe.

Puste miejsca zostaną wypełnione wartościami NO_DATA.

Ze względu na fakt, że program stara się jak najdłużej nie zamykać otwartego pliku, najbardziej wydajnie będzie pracował jeżeli wartości dla jednego parametru będą występowały bezpośrednio po sobie.

Podsumowując powyższe - prawidłowe i efektywne działanie programu wymaga aby linie wejściowe były pogrupowane najpierw po nazwie parametru, a następnie posortowane po dacie i czasie wartości. Najprościej uzyskać to przez wygenerowanie pliku, w którym wartości daty i czasu będą zapisane na stałej ilości znaków (dopełnione zerami), a następnie posortowanie go programem sort.

9. Instrukcja obsługi programu meaner3

Program meaner3 jest następną wersją demona meaner zapisującego wartości parametrów do bazy. Najważniejsze różnice w nowej wersji to zmiana formatu bazy danych na *SzarpBase* (Sekcja 8), udoskonalony mechanizm Bodas o większych możliwościach, a także wiele innych poprawek, między innymi lepsze logowanie informacji o ewentualnie pojawiających się błędach.

9.1. Konfiguracja programu

Program korzysta z biblioteki libpar, wczytuje za jej pomocą następujące parametry:

- *IPK* - ścieżka do pliku konfiguracyjnego z formacie IPK, parametr wymagany, czytany z sekcji globalnej.
- *datadir* - główny katalog z bazą w formacie SzarpBase, parametr wymagany, czytany z sekcji globalnej.
- *parcook_path* - ścieżka do pliku, używana do tworzenia identyfikatora dla mechanizmów komunikacji z programem parcook (IPC). Parametr wymagany, czytany z sekcji globalnej
- *log_level* - poziom logowania, parametr opcjonalny czytany z sekcji *meaner3* bądź globalnej. Domyślna wartość - 2.
- *log* - plik z logiem programu; parametr opcjonalny czytany z sekcji *meaner3* bądź globalnej.

9.2. Wykonywanie zadań w programie meaner3

W systemie konieczne jest okresowe wykonywanie pewnych działań - np. synchronizacji danych z innymi komputerami. Można by do tego wykorzystywać standardowe mechanizmy systemu operacyjnego (cron), ale ze względu na konieczność synchronizacji tych działań z zapisem do bazy SZARP, utworzono oddzielny mechanizm, realizowany przez proces zapisujący do bazy - *meaner3*.

Działanie mechanizmu jest następujące:

Konfiguracja składa się z szeregu sekcji, opisujących zadania do wykonania. Program po zakończeniu cyklu zapisu danych do bazy (wykonywane co 10 minut) rozpoczyna przeglądanie sekcji w kolejności ich wystąpienia w pliku konfiguracyjnym. Jeżeli czas uruchomienia sekcji zgadza się z aktualnym czasem, bądź też sekcja była w poprzednim cyklu wykonana z błędem i ma być powtórzona, wykonywana jest przypisana do sekcji komenda.

Jeżeli wykonywana sekcja jest skonfigurowana do *wykonywania równoległego*, program przechodzi do przeglądania następnych sekcji. Jeśli nie, przeglądanie sekcji jest przerywane do czasu zakończenia wykonywania ostatnio przeglądanej sekcji.

Wykonywanie sekcji kończy się albo z powodu zakończenia wykonywania przypisanej do niej komendy, albo z powodu przekroczenia limitu czasu dla sekcji. Limit ten jest ustalany dla każdej sekcji osobno, w szczególności sekcja może nie mieć limitu. Jeżeli sekcja skończy się z błędem lub zostanie przerwana z powodu przekroczenia limitu czasu, to będzie powtórzona w następnych cyklach, o ile zostanie to skonfigurowane.

Jeżeli sekcja nie ma limitu czasowego lub jej limit czasowy jest na tyle duży, że jest nadal wykonywana w trakcie kolejnego cyklu w którym powinna być uruchomiona, jest ignorowana.

Sekcja z dużym limitem (lub bez limitu) nie skonfigurowana do wykonywania równoległego blokuje oczywiście przeglądanie sekcji do czasu jej zakończenia.

W pliku konfiguracyjnym (zwykle */etc/szarp/szarp.cfg*) wygląda to tak, że w sekcji *execute* (lub w głównej) powinien znajdować się parametr *execute_sections*, zawierająca oddzielone spacjami nazwy sekcji do wykonania (kolejność ma znaczenie!). Na przykład:

```
:execute
```

```
execute_sections=sekcja1 sekcja2 sekcja3
```

Każda z sekcji zawiera następujące parametry:

- *time* (parametr wymagany) - czas uruchomienia sekcji w formacie *<dzień tygodnia> <miesiąc> <dzień> <godzina> <minuty>*. Dzień tygodnia powinien być z zakresu od 1 (poniedziałek) do 7 (niedziela), miesiąc - od 1 do 12, dzień miesiąca - od 1 do 31, godzina od 0 do 23, minuty - od 0 do 59, z tym że minuty są rozpoznawane z dokładnością do 10 minut, a więc nie ma znaczenia, czy wpiszemy 10, 11 czy 19. Można podać kilka wartości oddzielonych przecinkami, bądź też zakresów (w formacie *<liczba>-<liczba>*). Gwiazdka (*) oznacza całość dostępnego zakresu. Przykład:

```
:sekcja1
time=1-4,6-7 * * 8-15 10, 30, 50
```

Powyższy zapis oznacza uruchamianie sekcji codziennie oprócz piątków, w godzinach od 8 do 15, w każde nieparzyste 10 minut po okrągłej godzinie.

- *command_line* (parametr wymagany) - komenda do wykonania, przekazywane do wykonania przez powłokę - program sh. Może więc także zawierać instrukcje warunkowe, pętle itp.
- *retry* (parametr opcjonalny) - ilość kolejnych cykli po zakończeniu komendy z błędem, w których będzie ponawiana próba jej wykonania. Domyślnie jest to 0 - brak powtórzeń.
- *limit* (parametr opcjonalny) - limit czasu (w sekundach) na wykonanie komendy. Limit ten jest mierzony zawsze od początku danego cyklu (a nie od chwili uruchomienia). Liczba ujemna oznacza ilość sekund przed końcem obecnego cyklu. Jeżeli więc cykl trwa zwykle 10 minut (600 sekund), to wartość -1 jest równa wartości 599. Jest to także wartość domyślna. Na sekundę przed upływem limitu program otrzymuje sygnał SIGTERM, sekundę później SIGKILL. Wartość 0 oznacza brak limitu (komenda będzie działać do samodzielnego zakończenia lub zakończenia programu *meaner3*).
- *parallel* (parametr opcjonalny) - jeżeli ma wartość *yes* sekcja dopuszcza możliwość wykonywania innych równoległe z nią. Każda inna wartość oznacza *no* (i jest to wartość domyślna). W praktyce, zgodnie z opisanym wyżej mechanizmem uruchamiania sekcji, uruchamiane będą kolejno wszystkie sekcje z parametrem *parallel* ustawionym na "yes" oraz pierwsza z "no".

Poniższy przykład pokazuje konfigurację podtrzymującą tunel na komputer *praterm.com.pl*, z wykonywanym równoległe przepisywaniem danych na lokalny terminal i serwer *praterm.com.pl* oraz ze ściąganiem danych z innego komputera wykonywanym co godzinę, ale po zakończeniu przepisywania danych.

```
:execute

execute_sections=tunel local_backup praterm_push other_get

:tunel
time=* * * * *
command_line=/opt/szarp/bin/ssh_tunel $prefix$@praterm.com.pl 1234
limit=0
parallel=yes

:local_backup
time=* * * * *
command_line=/usr/bin/rsync $rsync_options$ $szarp_prefix$/$prefix$ \
terminal:
parallel=yes
```

```

:praterm_push
time=* * * * *
command_line=/usr/bin/rsync $rsync_options$ $szarp_prefix$/$prefix$ \
$prefix$@praterm.com.pl:/opt/szarp

:other_get
time=* * * * 0
command_line=/usr/bin/rsync $rsync_options$ other:/opt/szarp/dane /opt/szarp

```

9.3. Parametry informacyjne programu

Program *meaner3* w trakcie działania zapisuje zestaw kilkunastu parametrów informujących o stanie programu i wykonywanych zadaniach. Parametry te nazywane są *parametrami informacyjnymi*. Wartości parametrów odnoszą się do cyklu rozpoczynającego się o czasie zapisu parametru, tzn. np. wartości parametrów z godziny 11:40 odnoszą się do cyklu który rozpoczyna się o 11:40 a kończy zapisem o 11:50 (przyjmując długość cyklu 10 minut). Oznacza to w praktyce, że część parametrów informacyjnych jest zapisywana z datą wsteczną, np. informacja o wykonanych zadaniach odnosząca się do cyklu rozpoczynającego się o 11:40 jest dostępna dopiero o 11:50. W związku z tym dziury w zapisie części parametrów informacyjnych mogą nie pokrywać się z przerwami w działaniu programu.

Zapis parametrów informacyjnych następuje na początku cyklu, po zapisaniu właściwych parametrów. Parametry te są zapisywane zawsze, niezależnie od tego czy są zawarte w konfiguracji (podobne rozwiązanie powinien też implementować program przeglądający). Jeżeli w konfiguracji znajdzie się parametr o nazwie identycznej z jakimś parametrem informacyjnym, to jego zawartość zostanie nadpisana (bo parametry informacyjne są zapisywane po zwykłych).

Dostępne są następujące parametry:

- *Status:Meaner3:program uruchomiony* - parametr ma wartość 1 jeśli w danym cyklu program był uruchomiony lub BRAK DANYCH jeżeli nie był uruchomiony.
- *Status:Meaner3:ilość parametrów* - ilość wszystkich parametrów zawartych w pliku konfiguracyjnym.
- *Status:Meaner3:ilość parametrów ze sterowników* - ilość parametrów których wartości odczytywane są ze sterowników.
- *Status:Meaner3:ilość parametrów definiowanych* - ilość parametrów w sekcji *defined* pliku konfiguracyjnego IPK.
- *Status:Meaner3:ilość parametrów definiowalnych przeglądającego* - ilość parametrów w sekcji *drawdefinable* pliku konfiguracyjnego IPK.
- *Status:Meaner3:ilość parametrów zapisywanych do bazy* - - ilość parametrów z atrybutem *base_index* (czyli tych dla których funkcja *TParam::IsInBase()* zwróciła 1).
- *Status:Meaner3:ilość parametrów zapisywanych niepustych* - ilość parametrów zapisywanych do bazy które w danym cyklu miały wartości różne od BRAK DANYCH.
- *Status:Meaner3:ilość parametrów poprawnie zapisanych* - ilość parametrów, podczas zapisu których nie zgłoszono błędu, wliczane są w to także parametry z wartościami BRAK DANYCH, normalnie wartość tego parametru powinna być taka jak *Status:Meaner3:ilość parametrów zapisywanych do bazy*.

- *Status:Meaner3:ilość parametrów zapisanych z błędem* - ilość parametrów podczas których zgłoszono błąd zapisu, wartość różna od 0 oznacza zwykle problemy (awaria dysku, dysku pełen, brak uprawnień od zapisu lub tym podobne).
- *Status:Execute:ilość skonfigurowanych sekcji* - ilość sekcji *execute* znalezionych w pliku konfiguracyjnym SZARP.
- *Status:Execute:ilość sekcji do uruchomienia w cyklu* - ilość sekcji, które w danym cyklu powinny zostać uruchomione (włączając w to sekcje powtarzane).
- *Status:Execute:ilość sekcji uruchomionych w cyklu* - ilość sekcji jakie zostały w danym cyklu uruchomione.
- *Status:Execute:ilość sekcji zakończonych z sukcesem* - ilość sekcji jakie zostały w danym cyklu uruchomione i zakończyły się samoistnie z sukcesem (kodem powrotu równym 0).
- *Status:Execute:ilość sekcji zakończonych z błędem* - ilość sekcji które zakończyły się samoistnie z kodem powrotu różnym od 0.
- *Status:Execute:ilość sekcji przerwanych* - ilość sekcji jakie zostały przerwane z powodu przekroczenia limitu czasu.

10. Import danych z regulatorów węzłów Samson (system InTouch) lub Mikrob (program PRO-2000)

W systemie SZARP istnieje mechanizm pozwalający na automatyczny import danych w formacie CSV, w szczególności generowanych przez system InTouch zbierający dane z regulatorów węzłów Samson lub system PRO-2000 firmy Mikrob.

Mechanizm ten uruchamiany jest przez skrypt `script/samson2/get_sam`. Ściąga on potrzebne pliki z komputera z InTouch'em, a następnie przepuszcza je przez perlowy skrypt `script/samson2/get_sam`, który generuje plik wejściowy, po posortowaniu i odrzuceniu wpisów obecnych przy poprzednim uruchomieniu podawany do programu `szbwriter`. Dodatkowo skrypt `script/samson2/convert_names` jest używany do konwersji nazw okien programu przeglądającego.

Mechanizm sprawdza dane z dnia obecnego i poprzedniego, może być uruchamiany z dowolną częstotliwością, ale należy zwrócić uwagę, aby nie uruchamiać go przed ukończeniem poprzedniego przebiegu.

Konfiguracja SZARP'a jest generowana automatycznie na podstawie danych z węzłów. Nie ma sensu jej modyfikacja, gdyż może ona zostać automatycznie zmodyfikowana. W szczególności tracone są wszelkie informacje nieobsługiwane przez bibliotekę IPK.

Instalacja systemu polega na wykonaniu następujących kroków:

1. Upewniamy się, że na serwerze SZARP mamy zainstalowaną odpowiednio nową wersję systemu SZARP.
2. Na komputerze z działającym systemem InTouch/PRO-2000 instalujemy pakiet Cygwin wraz z serwerem ssh i programem rsync (zobacz rozdział Uruchamianie serwerów sshd i crond (<http://www.szarp.org/szarp/doc/howto/html/cygwin-servers.html>) w opisie Cygwina w SZARP HOWTO. Alternatywnie dostęp może być zapewniony przez udostępnienie katalogu przez Sambę.

3. InTouch/PRO-2000 musi być skonfigurowany w ten sposób, aby raz dziennie zapisywał dane z węzłów do plików CSV w jakimś katalogu.
4. Wymieniamy klucze, tak, aby użytkownik root z serwera SZARP mógł logować się bez hasła na komputer z InTouchem/PRO-2000 i za pomocą mechanizmu rsync ściągać wygenerowane pliki CSV.
5. Ustalamy prefiks konfiguracji z danymi z węzłów i tworzymy odpowiedni katalog
/opt/szarp/<prefix>/config i /opt/szarp/<prefix>/szbase.
6. W katalogu /opt/szarp/<prefix>/config tworzymy plik `samsons_LASTDATE` zawierający datę pierwszych danych, które chcemy ściągnąć, w formacie RRRR-MM-DD, np. 2004-09-01.
7. W pliku /etc/szarp/szarp.cfg w sekcji o nazwie *samsony* umieszczamy następujące parametry:
 - *config_prefix* - prefiks nowo tworzonej konfiguracji.
 - *remote_host* - nazwa (lub numer IP) komputera z systemem InTouch.
 - *dynamic_host* - "yes" jeżeli komputer z InTouchem ma dynamiczne IP, które będziemy pobierać z serwera praterm.com.pl. Jako identyfikator komputera służyć będzie wartość parametru *remote_host*.
 - *remote_user* - nazwa użytkownika na komputerze z InTouch, z którego konta będziemy ściągać pliki z danymi.
 - *remote_path* - ścieżka do katalogu z plikami CSV (na komputerze Windows).
 - *params_group* - nazwa grupy parametrów (parametry będą miały nazwę postaci Grupa:Nazwa węzła:nazwa parametru).
 - *date_format* - format daty używany w nazwach plików na komputerze z InTouchem/PRO-2000, powinien to być prawidłowy parametr 'format' dla programu **date**. Domyślna wartość to „,+%Y-%m-%d”.
 - *separator* - napis (znak) używany jako separator pól w pliku CSV, domyślnie jest to przecinek, dla systemu PRO-2000 powinien to być tabulator ('\t').
 - *nodata_str* - napis oznaczający brak wartości, domyślnie przyjmowany jest napis pusty, dla systemu PRO-2000 powinien to być napis 'brak'.

Dodatkowo, w sekcji *szbwriter* ustawiamy parametr *double_match*, zawierający wzorzec powłoki, do którego mają pasować nazwy parametrów zapisywanych na dwóch słowach. Jeżeli dane z węzłów zbierane są rzadziej niż co 10 minut, możemy ustawić też wartość parametru *fill_how_many*. Zobacz też Sekcja 8.5.1.

Przykładowy fragment pliku `szarp.cfg`:

```
:samsony

# prefiks tworzonej konfiguracji
config_prefix=skis
# nazwa zdalnego komputera
remote_host=skis
# czy jest to dynamiczne IP
dynamic_host=yes
# nazwa zdalnego uzytkownika
remote_user=szarp
# sciezka do raportow
remote_path=/cygdrive/d/Raporty/Odczyty
# nazwa grupy z parametrami
params_group=Samsony
```

```
:szbwriter
# shell pattern dla parametrow dwuslownych - zobacz 'info fnmatch'
double_match=@(*-energia|*-objetosc)
```

8. Kopiujemy skrypty `get_sam`, `process_csv` i `convert_names` z podkatalogu `script/samson2` w źródłach SZARP'a do katalogu `/opt/szarp/bin`.
9. Ustalamy uruchamianie skryptu `get_sam` w zadanym odstępie czasowym (np. co pół godziny) - albo za pomocą `crontaba`, albo za pomocą mechanizmu `execute` programu `meaner3` (zalecane opcje `parallel=yes` i `limit=0` - zobacz Sekcja 9.2).

Dodatkowo, istnieje możliwość ustalenia przez użytkowników nazw okien w programie przeglądającym (domyślnie są to "Samsony:<numer węzła>"). Odbywa się to przez umieszczenie na komputerze z Windows w katalogu z plikami CSV pliku `nazwy.txt`. Powinien on zawierać w kolejnych liniach starą (domyślną) i nową nazwę węzła, oddzielone myślnikiem, bez dodatkowych spacji. Przykład:

```
W112-Cicha 8
W115-Głośna 7
W528-Jana Sobieskiego 34
```

Nazwy mogą zawierać polskie znaki (w kodowaniu CP-1250!), są obcinane do 30 znaków, nie mogą zawierać znaków cudzysłowu i ukośnika. Konwersja jest dokonywana tekstowo na pliku `params.xml` konfiguracji. Nowe nazwy obowiązują do zmiany, przy czym zaleca się nieusuwanie starych wpisów, tylko dodawanie nowych na końcu. Przykładowo, jeżeli węzeł W112 miał się nazywać Cicha 8, a potem zmieniono jego nazwę na Cicha 12, to w pliku powinny być obecne wpisy:

```
W112-Cicha 8
Cicha 8-Cicha 12
```

Pozwoli to na prawidłową konwersję także parametrów z węzła W112, które pojawią się w konfiguracji w przyszłości.

11. BODAS - Baza Ogólnopolska Danych Archiwalnych Systemu SZARP

11.1. Koncepcja systemu

Celem pierwotnym było stworzenie centrum internetowego, na którym składowane i regularnie uaktualniane byłyby bazy danych historycznych wszystkich systemów posiadających SZARPa. Z czasem okazało się, że należy to uogólnić na przesyłanie i uaktualnianie baz między różnymi komputerami w taki sposób, aby proces przesyłania trwał jak najkrócej.

Struktura systemu w najprostszej postaci zakłada, że istnieje jeden centralny serwer, na który zgrywane są dane z wszystkich obiektów (ciepłowni, węzłów). Transmisja inicjowana jest z obiektów. Dodatkowo obiekty mogą wysyłać swoje dane na lokalne terminale oraz ewentualnie bezpośrednio na inne obiekty.

Użytkownicy indywidualni oraz ewentualnie inne obiekty pobierają dane z serwera centralnego - tu także transmisja inicjowana jest z obiektów, serwer centralny zawsze jest stroną bierną.

Opisany schemat czasami może się komplikować - czyli mogą istnieć serwery pośredniczące, które dla pewnej grupy obiektów stanowią "serwer centralny".

11.2. Realizacja BODASa

Używane są obecnie 2 implementacje. Pierwsza z nich używa programu do różnicowego przesyłania danych **rsync** (z pewnymi opisanymi dalej komplikacjami) i wykorzystywana jest do automatycznego przesyłania danych między obiektami. Druga wykorzystuje protokół oparty na rsyncu, wymiana danych odbywa się między SZARP Synchronization Server zainstalowanym na serwerze centralnym, a graficznym klientem Szarp Synchronization Client zainstalowanym na komputerze użytkownika chcącego oglądać dane historyczne. Obie implementacje w rezultacie powodują utworzenie lokalnej kopii wszystkich plików tworzących bazę SZARP. Oznacza to że mechanizmy te mogą być używane wymiennie, a w razie potrzeby baza może być po prostu skopiowana/przeniesiona w dowolny inny sposób.

Rsync używany jest przez ssh, zwykle z wymianą kluczy między przesyłającymi dane komputerami aby uniknąć wpisywania hasła. Domyślnie przesyłane są dane tylko z 2 ostatnich miesięcy. Jeżeli potrzeba zsynchronizować starsze pliki (bo np. ręcznie zmodyfikowano historyczne dane), to na obiekcie źródłowym dla danych modyfikowany jest plik `/opt/szarp/<prefix>/szbase_stamp`. W związku z tym aby sprawdzić czy trzeba przysłać całość czy tylko 2 ostatnie miesiące, należy sprawdzić datę modyfikacji tego pliku. Może być to robione przez ssh, ale wymaga to wtedy zestawienia dwóch połączeń - najpierw do sprawdzenia daty pliku, później do przesłania danych. Dostępny jest więc mechanizm pozwalający na zmniejszenie obciążenia serwera - data modyfikacji pliku może być sprawdzona przez protokół HTTP, łączący się do serwera **stampd** działającego na porcie 81. Całość odpowiednich operacji wykonują skrypty `/opt/szarp/bin/szbpush` i `/opt/szarp/bin/szrsync` służące odpowiednio do wysyłania i odbierania danych. Szczegóły ich użycia można sprawdzić wywołując je z opcją `-h`.

Więcej informacji na temat drugiej implementacji znajduje się w rozdziale o serwerze synchronizacji - Sekcja 14 oraz w dokumentacji programu SSC - <http://szarp.org/szarp/doc/ssc/html/ssc.html>.

11.3. Konfiguracja BODASa do wysyłania danych z obiektu

Należy upewnić się, że zarówno na serwerze SZARPa, jak i na komputerze, na który będziemy chcieli wysyłać, jest zainstalowany program **rsync**. Poza tym, jeśli będziemy się łączyć przez sieć, komputery powinny być skonfigurowane do połączenia ssh bez konieczności wpisywania hasła. Użytkownikiem łączącym się będzie tu root, zaś na zdalnym komputerze najlepiej utworzyć specjalnego użytkownika, najlepiej bez hasła, z możliwością logowania się tylko za pomocą wymienionego klucza.

System SZARP może obsługiwać jednocześnie wiele *sekcji push*, które mogą dotyczyć różnych docelowych lokalizacji kopii bazy. Na przykład możemy chcieć mieć backup lokalny wykonywany co godzinę, kopię na innym terminalu w tej samej sieci uaktualnianą co 10 minut oraz łączyć się z serwerem internetowym, aby uaktualnić centralny backup bazy.

BODASa konfigurujemy podobnie jak inne rodzaje sekcji *execute* (zobacz opis). W pliku konfiguracyjnym SZARPa (`/etc/szarp/szarp.cfg`) trzeba zadeklarować żądane sekcje *execute*. Robimy to przez umieszczenie w sekcji o nazwie **execute** parametru **execute_sections**. Jego wartością powinna być lista oddzielonych spacjami sekcji *push*:

```
:execute
execute_sections=bodas1 bodas2 bodas3 ...
```

Teraz dla każdej z zadeklarowanych sekcji *push* należy utworzyć oddzielną sekcję w pliku konfiguracyjnym, a więc:

```
:bodas1
```

```
...
```

```
:bodas2
```

```
....
```

Kolejność wartości w parametrze *execute_sections* wyznacza kolejność uruchamiania sekcji przez SZARPa.

W każdej z tych sekcji należy umieścić następujące parametry:

- *command_line* - komendę do wykonania, np.

```
command_line=/opt/szarp/bin/szbpush -c user@server baza1 baza2
```

Zwyczajowo nazwa użytkownika *user* jest identyczna z nazwą komputera wysyłającego dane. Na komputerze *server* należy wcześniej utworzyć użytkownika *user*:

```
adduser user
```

oraz katalog `/opt/szarp/user`, z prawami dostępu ograniczonymi tylko do właściciela:

```
mkdir /opt/szarp/user
```

```
chown user:user /opt/szarp/user
```

Należy także skonfigurować połączenie ssh między komputerami tak, aby użytkownik *root* z komputera wysyłającego mógł się logować bez podawania hasła jako *user* na komputer *server*.

- *retry* - specyfikuje ile razy należy powtarzać nieudaną próbę transmisji. Jeżeli parametr ten jest większy niż 0, to w przypadku niemożności nawiązania połączenia lub po zgłoszeniu błędu przez program **rsync**, po następnym zapisie do bazy (a więc 10 minut później) próba synchronizacji zostanie powtórzona. Jeżeli więc chcemy, aby po nieudanej transmisji SZARP jeszcze 3 razy próbował ją powtórzyć, należy ustawić

```
retry=3
```

- *time* - parametr określa, kiedy uruchamiać daną sekcję. Jego postać to: *dzień_tygodnia miesiąc dzień_miesiąca godzina minuta*. Dzień tygodnia musi być liczbą z zakresu 1 (poniedziałek) do 7 (niedziela), miesiąc liczbą 1 - 12, dzień miesiąca 1 - 31, godzina 0 - 23, minuta 0 - 59. Minuty są zawsze zaokrąglane do 10 minut w dół, więc zarówno 13 jak i 15 oznacza tak naprawdę 10. Zamiast liczby może wystąpić „gwiazdka”, oznaczająca dowolną wartość. Jeżeli więc chcemy aby sekcja wykonywała się co 10 minut, wpisujemy

```
time=* * * * *
```

Jeżeli codziennie o północy, to możemy wpisać:

```
time=* * * 0 5
```

Jeżeli w każdą środę kwietnia o 13:40, to wpisujemy

```
time=3 4 * 13 45
```

Więcej informacji można znaleźć w rozdziale o mechanizmie *execute* - Sekcja 9.2.

Oto przykładowy fragment pliku konfiguracyjnego `/etc/szarp/szarp.cfg`, zawierający konfigurację BODASa dla wykonywania co 10 minut lokalnej kopii i codziennie o północy synchronizacji z bazą centralną na serwerze internetowym:

```

:execute
execute_sections=local_copy remote_copy

:local_copy
command_line=/opt/szarp/bin/szbpush terminal1 leg1
time=* * * * *

:remote_copy
command_line=/opt/szarp/bin/szbpush -c leg1@szarp.com leg1
time=* * * 0 5

```

12. Statusy modemów - opis zwracanych wartości.

12.1. Opis wartości

Do komunikacji między modemami używany jest skrypt `modem_pooler.sh`. Zwraca on pewne wartości, które następnie są widoczne w programie przeglądającym w postaci wykresu "Statystyki modemów". Brak wartości oznacza, że ciągu danych 10 minut nie podejmowano próby komunikacji z modemem.

- "-9" - za mało czasu na odpytanie danego modemu.
- "-5" - nie można połączyć się z modemem na węźle (nie można zestawić połączenia ppp). Jeżeli wartość ta występuje dla wszystkich węzłów, sugeruje to problem z modemem po stronie ciepłowni. Jeżeli dla pojedynczego węzła, może to oznaczać że modem jest odłączony, uszkodzony lub że modem (lub cały komputer) na węźle jest wyłączony,
- "0" - połączono z węzłem i ściągnięto dane. Jeżeli mimo wartości 0 na wykresach parametrów z węzła brak jest wartości, oznacza to że komputer na węźle nie zbiera danych.
- "30" - przekroczono czas oczekiwania na odpowiedź - prawdopodobna przyczyna to zakłócenia na linii.
- Pozostałe wartości dodatnie są to błędy zwracane przez program `rsync`, jeżeli występują przez dłuższy czas, mogą oznaczać błąd w konfiguracji komunikacji z węzłami.

13. ISL

Rozdział opisuje elementy systemu ISL, udostępniającego dane z systemu SZARP za pomocą protokołu HTTP.

13.1. Biblioteka serwera HTTP

Program *paramd* oparty jest na prostej bibliotece realizującej funkcjonalność serwera HTTP. Biblioteka implementuje niewielki, ale funkcjonalny podzbiór protokołów HTTP w wersji 1.0 i 1.1, ale bez połączeń stałych (nagłówek "Connection: Keep-Alive"), gdyż implementowany serwer jest jednowątkowy i obsługuje tylko 1 połączenie na raz. Biblioteka odpowiada wersją protokołu taką samą, w jakiej przyszło zgłoszenie klienta. Obsługiwane są metody HTTP GET oraz POST. W przypadku błędów, biblioteka odsyła informację z kodem błędu HTTP, zgodnie z RFC 2616. Jedynym dostępnym kodowaniem wyjściowym jest UTF-8 (Unicode). Obsługiwana jest opcjonalna autoryzacja *Basic*, opcjonalnie połączenia HTTP można tunelować przez protokół SSL. Wykorzystywana jest biblioteka OpenSSL, zapewniająca obsługę protokołu SSL w wersji 2 i 3 oraz TSL.

Możliwe jest uruchomienie w ramach jednej aplikacji wielu serwerów, z różnymi opcjami, na różnych portach. Każdy serwer działa w ramach oddzielnego procesu.

Serwer testowany był z większością popularnych przeglądarek.

13.2. Konfiguracja serwera HTTP

Serwer wykorzystuje bibliotekę *libpar*, główna konfiguracja jest więc zapisana w pliku `/etc/szarp/szarp.cfg`, którego format opisany jest w rozdziale Sekcja 4.1.

W sekcji o nazwie odpowiadającej nazwie konkretnego programu (a więc np. *paramd*) umieszczony jest parametr *servers*. Zawiera on listę oddzielonych spacjami sekcji, z których każda opisuje jeden proces - server.

Każda z takich sekcji zawiera następujące parametry:

- *port* - port TCP/IP, na którym serwer będzie przyjmować połączenia, domyślnie przydzielane są porty o numerach od 8081. Oczywiście nie zadziała to jeżeli w systemie będzie uruchamianych kilka aplikacji korzystających z biblioteki serwera, gdyż obie będą chciały przydzielać sobie porty poczynając od 8081.
- *timeout* - czas w sekundach, po jakim w razie braku reakcji klienta połączenie TCP/IP będzie zamknięte. Dotyczy to zarówno sytuacji, gdy klient nie zdążył przesłać całego żądania HTTP. Domyślna wartość to 30.
- *allowed_ip* - lista adresów IP, z których dozwolone jest łączenie się z serwerem, w formacie zgodnym z funkcją `fnmatch`. Domyślnie jest to `*` (dozwolone są połączenia z dowolnego adresu). W przypadku tego parametru zapis listy wzorców (zgodny z funkcją `fnmatch`) znajduje się poniżej (*lista-wzorców* oznacza kolejne wzorce oddzielone znakiem `'\'`).
 - *?(lista-wzorców)* - Wyrażenie jest prawdziwe, gdy żaden lub jeden wzorzec z listy wzorców został dopasowany.
 - **(lista-wzorców)* - Wyrażenie jest prawdziwe, gdy żaden lub dowolna liczba wzorców z listy wzorców została dopasowana.
 - *+(lista-wzorców)* - Wyrażenie jest prawdziwe, gdy co najmniej jeden wzorzec z listy wzorców został dopasowany.
 - *@(lista-wzorców)* - Wyrażenie jest prawdziwe, gdy dokładnie jeden wzorzec z listy wzorców został dopasowany.
 - *!(lista-wzorców)* - Wyrażenie jest prawdziwe, gdy żaden wzorzec z listy wzorców nie został dopasowany.

- *auth* - czy serwer ma żądać autoryzacji (typu HTTP Basic). Autoryzacja będzie włączona jeżeli parametr ma wartość *yes*, domyślna wartość to *no*.
- *access* - ścieżka do pliku definiującego prawa dostępu do zasobów serwera, ma znaczenie tylko jeżeli włączona jest autoryzacja. Domyślnie jest to plik *access.conf* w katalogu głównym. Plik ten zawiera linie o następującej postaci:

```
[ścieżka do węzła]:[nazwa użytkownika]
```

Linie o innej postaci są ignorowane. Ścieżka opisuje parametr lub węzeł w drzewie parametrów, a nazwa użytkownika oznacza użytkownika, który ma prawo do ustawiania danego parametru lub grupy parametrów zawartej w podanym węźle. Ścieżki do węzłów i parametrów powinny być podane w kodowaniu UTF-8. Przykładowy plik może wyglądać tak:

```
/:root
/Kocioł 1:palacz1
/Kocioł 2/Sterownik/temperatura zadana:palacz2
```

Do ustawienia parametru potrzebne jest także podanie hasła. Serwer sprawdza, czy użytkownik o podanej nazwie ma prawo do dostępu do danej ścieżki, a także czy istnieje taki użytkownik na serwerze i czy podane hasło jest prawidłowe. Korzystanie z haseł systemowych wymaga obecności w katalogu */etc/pam.d* pliku *paramd* o następującej zawartości:

```
##PAM-1.0
auth      required      /lib/security/pam_stack.so service=system-auth
```

Plik ten jest automatycznie tworzony także po wydaniu komendy **make ssl** w katalogu ze źródłami programu.

- *ssl* - *yes* jeżeli serwer ma korzystać z protokołu HTTPS (HTTP tunelowany przez SSL), domyślnie jest to *no*.
- *keypass* - hasło użyte do zakodowania prywatnego klucza RSA serwera, domyślnie "paramd". Zobacz też opis opcji *key*.
- *key* - ścieżka do pliku zawierającego prywatny klucz RSA serwera. Klucz ten może być wygenerowany na przykład przez wydanie komendy **make ssl** w katalogu ze źródłami programu. Tworzony jest plik *server.pem*, kopiowany także do katalogu */opt/szarp/resources*. Hasło używane do zakodowania klucza jest zapisane w pliku *config.ssl* (parametr *output_password*). Aby serwer działał, to samo hasło musi być podane jako wartość parametru *keypass* w konfiguracji programu. Domyślnie klucz jest czytany z pliku *server.pem* w katalogu głównym.
- *cert* - ścieżka do pliku zawierającego certyfikat serwera. Jest on generowany razem z kluczem (patrz opis poprzedniej opcji) i zapisywany w tym samym pliku - choć użytkownik może też podać inny plik z prawidłowym certyfikatem x509. Wygenerowany certyfikat ma następujące parametry:

```
E = pawel@praterm.com.pl
CN = [nazwa komputera]
OU = ISL
O = PRATERM SA
L = Warsaw
ST = Poland
C = PL
```

Ważność wygenerowanego certyfikatu wynosi 1 rok. Domyślnie certyfikat jest czytany z pliku *server.pem* w katalogu głównym.

Przykładowy plik konfiguracyjny może wyglądać na przykład tak:

```

:paramd

servers=local remote

:local
port=8081
allowed_ip=127.0.0.1

:remote
port=8082
auth=yes
ssl=yes
keypass=paramd
access=/opt/szarp/resources/paramd_access.conf
key=/opt/szarp/resources/server.pem
cert=/opt/szarp/resources/server.pem

```

13.3. Program paramd

Program `paramd` jest serwerem HTTP, który udostępnia informacje o wartościach parametrów zbieranych przez system SZARP w postaci dokumentów XML lub HTML. Dodatkowo udostępnia on również informacje o raportach zdefiniowanych w IPK, a także pozwala na definiowanie własnych raportów.

13.3.1. Konfiguracja programu

Konfiguracja programu jest trzymana w pliku `/etc/szarp/szarp.cfg`. Parametry są standardowo czytane z sekcji o nazwie `paramd`. Jeśli nie zostaną znalezione w tej sekcji, to będą szukane w sekcji głównej. Specyficzne dla programu są następujące parametry:

- *PTT* - ścieżka do pliku `PTT.act`, zawierającego opis dostępnych w systemie parametrów. Parametr jest wymagany.
- *log* - plik z logiem programu, do którego będą wypisywane komunikaty informacyjne i o błędach. Plik ten musi być dostępny do zapisu dla użytkownika uruchamiającego program. Domyślnie jest to plik `paramd.log` w aktualnym katalogu.
- *log_level* - poziom logowania, od 0 (tylko błędy), do 10. Domyślnie jest to poziom 2. Dokładniej, poziomy mają następujące znaczenie:
 - 0 istotne błędy, powodujące przerwanie wykonywania programu lub jego istotnej części (np. brak pamięci), mogące też świadczyć o błędach w programie.
 - 1 ostrzeżenia o niemożności wykonania istotnego fragmentu programu, świadczące zazwyczaj o błędach w konfiguracji.
 - 2 podstawowe informacje o wykonanych działaniach, mogące służyć do oceny rodzaju i częstości wykonywania zadań (np. podstawowa informacja o zapytaniach obsłużonych przez serwer HTTP).
 - 3 dodatkowe informacje statystyczne i wydajnościowe.
 - 4 komunikaty o błędach o charakterze tymczasowym (np. niemożność połączenia się z serwerem).

- 5 komunikaty przydatne przy testowaniu nowej konfiguracji.
- 10 komunikaty pomocne przy szukaniu przyczyn błędów w działaniu programu (debugging).
- *update_freq* - częstotliwość w sekundach odświeżania informacji o wartościach parametrów. Wartości parametrów są odczytywane tylko na żądania klientów, ale nie częściej niż co podaną ilość sekund. Domyślną wartością jest 10, co odpowiada częstotliwości odświeżania wartości próbek w systemie SZARP.
- *parcook_path* - ścieżka do pliku, który posłuży jako klucz do generowania identyfikatora pamięci dzielonej parametrów. Zwykle jest to ścieżka do programu parcook lub netpar systemu SZARP. Programy te korzystają do tworzenia segmentów pamięci dzielonej z parametru o takiej samej nazwie z pliku `/etc/szarp/szarp.cfg`, ale czytane zawsze z sekcji głównej. Parametr jest wymagany.
- *html_header* - napis, który będzie wypisywany na początku każdego dokumentu HTML generowanego przez program. Powinien w szczególności zawierać poprawny nagłówek HTML, a więc tagi otwierające `<html>` i `<body>`. Domyślny nagłówek ma postać:

```
<html>
<head>
<title>
Paramd output
</title>
</head>
<body>
```

Prostym sposobem na automatyczne odświeżanie wysyłanych przez program stron HTML jest wstawienie do nagłówka html następującego tekstu:

```
<meta http-equiv="Refresh" content="10; http://hostname:8081">
```

Ponieważ serwer używa kodowania Unicode (UTF-8), więc tekst wstawiony jako zawartość tego parametru także musi być zakodowany w UTF-8.

- *html_footer* - tekst odłączany na koniec każdego dokumentu HTML generowanego przez program. Domyślnie zawiera tylko odpowiednie tagi zamykające:

```
</body>
</html>
```

Podobnie jak w przypadku poprzedniego parametru, tekst powinien być zakodowany w UTF-8.

Poza tym w pliku powinna znaleźć się konfiguracja jednego lub więcej serwerów HTTP, zgodnie z opisem w rozdziale Sekcja 13.2.

13.3.2. Dostęp do zasobów serwera paramd

Paramd udostępnia informacje o parametrach pogrupowanych według ich hierarchicznej nazwy lub raportów, w których są umieszczone. Jak uzyskać informacje o parametrach opisuje rozdział Sekcja 13.3.2.1. Natomiast funkcje związane z raportami (wykorzystywane głównie przez program `reporter3`) opisane są w Sekcja 13.3.2.2.

13.3.2.1. Dostęp do informacji o parametrach

Przechowywana przez program informacja o parametrach ma postać drzewa. Dostęp do niego uzyskuje się przez specyfikowanie ścieżki dostępu podobnie jak do zasobów WWW. Dokładne informacje o postaci ścieżki (lokatora - URI) znajdują się niżej. Lokator musi być zakodowany w UTF-8, przy czym obsługiwane jest kodowanie za pomocą sekwencji `%kod`. Są one obowiązkowe np. dla spacji.

Alternatywnie, w lokatorze można używać zamiast polskich liter ich angielskich odpowiedników, a zamiast spacji i ukośnika występującego w nazwie parametru znaku podkreślenia (szczegóły w następnym rozdziale). Dokumenty serwowane są z nagłówkiem "Expires:" zgodnym z częstotliwością odświeżania wartości parametrów.

Dostęp do informacji udostępnianych przez serwer wymaga podania, np. w oknie przeglądarki internetowej, odpowiedniego adresu zasobu. Adres ten ma postać:

protokół://adres_serwera:port/ścieżka?opcje

Protokół może przyjmować wartość *http* dla połączeń zwykłych, lub *https* dla szyfrowanych (o ile ich obsługa została włączona w konfiguracji serwera).

Adres serwera specyfikuje adres internetowy komputera, na którym działa program.

Port (po dwukropku) jest numerem portu, podanym w pliku konfiguracyjnym programu.

Ścieżka jest to ścieżka dostępu do żądanego zasobu. Może być pusta, co oznacza dostęp do korzenia drzewa parametrów. Jeśli jest niepusta, to zawiera listę kolejnych węzłów drzewa, począwszy od korzenia, oddzielonych ukośnikami. Przykład:

http://localhost:8081/Kociół 1/Sterownik/temperatura zadana

Ścieżka w tej postaci zostanie poprawnie zinterpretowana tylko jeśli przeglądarka potrafi odpowiednio zakodować występujące w niej polskie litery i spacje. To także jednak nie wystarczy, jeżeli w nazwie węzła znajdzie się np. ukośnik. W związku z tym polskie litery można zastąpić ich angielskimi odpowiednikami (zamiast "ą" - a, "Ą" - "A", "ń" - "n", "ź" i "ż" - "z" itp.). Zamiast spacji i ukośnika w nazwie parametru można stosować znak podkreślenia. Tak więc cała ścieżka zapisana w ten sposób może wyglądać tak

http://localhost:8081/Kociol_1/Sterownik/temperatura_zadana

Węzły takie jak np. "Kociół 1" są rodzajem katalogów, zawierają tylko inne węzły. Natomiast węzeł "temperatura zadana" odpowiada parametrowi. Każdy parametr posiada atrybuty, dostęp do nich jest możliwy przez dodanie na końcu ścieżki znaku "@" i nazwy atrybutu. Na przykład:

http://localhost:8081/Kociol_1/Sterownik/temperatura_zadana@value

Dostępne są następujące atrybuty:

- *name* - nazwa parametru.
- *full_name* - pełna nazwa parametru , w postaci nazw węzłów oddzielonych dwukropkami (notacja używana w systemie SZARP).
- *short_name* - skrócona nazwa parametru.
- *unit* - nazwa jednostki parametru.
- *ipc_ind* - indeks parametru w pamięci dzielonej.
- *prec* - precyzja parametru - liczby do 5 włącznie oznaczają ilość cyfr po przecinku, powyżej są to wartości specjalne. Atrybut ten jest używany wewnętrznie przez serwer do interpretowania wartości parametru odczytanej z pamięci dzielonej.
- *value* - wartość parametru. Napis "unknown" oznacza brak dostępności tej informacji (np. niedziałający serwer SZARP, brak komunikacji ze sterownikami itp.).
- *v_u* - wartość parametru wraz z jednostką.

Po znaku zapytania można podać opcje do zapytania. Opcje mają postać "nazwa=wartość" i są oddzielane od siebie znakiem "&". Na przykład:

`http://localhost:8081/Kociol_1/Sterownik?_opcja1=wart1&_opcja2=wart2`

Obecnie dostępna jest jedna opcja o nazwie "output". Określa ona, co ma pojawić się na wyjściu serwera (zostać wysłane do przeglądarki). Może ona przyjmować następujące wartości:

- *html* - wartość domyślna, określa że wygenerowana ma zostać strona HTML. Dla parametrów strona zawiera informacje o wartościach atrybutów oraz link do węzła nadrzędnego. Dla pozostałych węzłów zawiera listę węzłów potomnych (jeśli są to parametry, to także ich wartości i nazwę skróconą), oraz link do węzła nadrzędnego. Elementy listy są także linkami, po ich kliknięciu następuje przejście do odpowiedniego węzła.
- *xml* - serwer wysyła zawartość danego węzła wraz z węzłami potomnymi w postaci XML. Zawsze serwowany jest kompletny dokument XML. Dokument ten ma ustawioną domyślną przestrzeń nazw XML (XML namespace) na identyfikator `http://www.praterm.com.pl/ISL/params`.
- *text* - wartość przyjmowana także dla innych, nieznanymi wartości opcji. Produkuje wyjście takie jak opcja "xml", ale generowany jest inny nagłówek HTTP, z typem zawartości "text/plain". Pozwala to na wyświetlenie dokumentu XML w postaci czystego tekstu także w przeglądarkach nie obsługujących XML.

Jeżeli podana ścieżka zawiera nazwę atrybutu, to wartość opcji "output" nie może być równa "html" - jeśli jest różna od "xml", to traktowana jest jak "text". Wysyłany jest dokument XML zawierający węzeł z wartością atrybutu parametru w kodowaniu UTF-8.

Wszystkie generowane dokumenty XML zgodne są z DTD, które można znaleźć w pliku `params.dtd`.

13.3.2.2. Dostęp do informacji o raportach

`Paramd` udostępnia informacje o parametrach umieszczonych w raportach w sposób analogiczny do opisanego w rozdziale Sekcja 13.3.2.1. Dodatkowe funkcjonalności są realizowane za pośrednictwem odpowiednio przygotowanej ścieżki dostępu do zasobu (URI). `Paramd` obsługuje następujące ścieżki:

- *protokół://adres_serwera:port/xreports* - udostępnia w postaci dokumentu XML listę raportów zdefiniowanych w IPK.
- *protokół://adres_serwera:port/xreports?title=nazwa_raportu* - udostępnia w postaci dokumentu XML listę parametrów raportu wraz z wartościami.
- *protokół://adres_serwera:port/custom?title=klucz_identyfikacyjny* - udostępnia w postaci dokumentu XML listę parametrów raportu definiowalnego wraz z wartościami (tworzenie parametrów definiowalnych w `paramd` w rozdziale Sekcja 13.3.4).

13.3.3. Ustawianie wartości parametrów

Ustawianie wartości parametrów jest bardzo proste - polega na dodaniu do ścieżki wpisywanej w przeglądarce opcji *put*, z wartością będącą żadaną wartością parametru. Ustawianie parametrów możliwe jest tylko przy wykorzystaniu połączenia szyfrowanego. Cała ścieżka może więc mieć postać:

https://localhost:8082/Kociol_1/Sterownik/temperatura_zadana?put=92.3

Należy zwrócić uwagę, że dla różnych typów parametrów dopuszczalne są tylko pewne wartości. Dla większości będą to wartości liczbowe. Inne możliwe zestawy wartości to:

- *Tak, Nie*
- *Pochmurno, Zmiennie, Slonecznie*
- *Plus, Minus*

Przy próbie ustawienia wartości parametru przeglądarka poprosi o nazwę użytkownika i hasło. Podany użytkownik musi istnieć na komputerze, na którym działa serwer paramd z podanym hasłem, a poza tym użytkownik musi mieć zapisany w konfiguracji serwera dostęp do podanego parametru (zobacz Sekcja 13.3.1). Przy kolejnym ustawieniu wartości parametru podawanie hasła nie będzie już konieczne, chyba że od ostatniego dostępu do parametru upłynęło więcej niż 5 minut. Ma to zabezpieczyć przed sytuacją, gdy ktoś z uprawnieniami ustawi wartość parametru, po czym zostawi komputer z włączoną przeglądarką.

13.3.4. Definiowanie własnych raportów

Funkcjonalność ta jest wykorzystywana przez program `raporter3`. Raporty definiowalne można dodać do `paramd` przesyłając odpowiednio przygotowany XML metodą POST HTTP na adres:

protokół://adres_serwera:port/custom/add

Przesyłany XML powinien być zgodny ze schematem RelaxNG umieszczonym w pliku `resources/dtd/params-list.rng` (lokalizacja pliku względem głównego katalogu SZARP), używającym przestrzeni nazw o identyfikatorze `http://www.praterm.com.pl/SZARP/list`. W odpowiedzi otrzymuje się klucz identyfikacyjny, a zdefiniowany raport będzie dostępny pod adresem:

protokół://adres_serwera:port/custom/kucz_identyfikacyjny

Zdefiniowany raport będzie dostępny w `paramd` dopóki będą do niego odwołania (tzn. jeżeli przez określony okres czasu nikt się nie odwoła do powyższego adresu, raport ten będzie usunięty).

13.4. Wizualizacja z wykorzystaniem ISL - HOWTO

Rozdział ten opisuje pokrótce założenia działania, sposób instalacji i konfiguracji oraz tworzenie schematów wizualizacji z wykorzystaniem technologii ISL.

13.4.1. Opis działania

Sposób działania ISL jest dość prosty, wykorzystuje istniejące darmowe narzędzia i otwarte technologie. Aby lepiej zrozumieć co i jak chcemy osiągnąć, prześledzimy drogę jaką przebywają dane technologiczne, aby pojawić się na ekranie użytkownika. Wygodniej będzie zacząć od końca, czyli właśnie od ekranu użytkownika.

Jako program kliencki wykorzystywana jest przeglądarka Firefox w wersji 1.5 lub późniejszej - te wersje obsługują język SVG. SVG jest zaprojektowanym specjalnie dla Internetu językiem opisu grafiki wektorowej opartym na XML, stworzonym przez organizację World Wide Web Consortium, autora między innymi standardów HTML czy XML.

Wyświetlany przez przeglądarkę obraz jest dokumentem SVG, czyli plikiem XML z opisem prezentowanego obrazu. Jednym z elementów dokumentu jest skrypt w interpretowanym przez przeglądarkę języku JavaScript (ważne - kod JavaScript jest wykonywany po stronie przeglądarki, a nie serwera). Zadaniem skryptu jest odświeżanie elementów obrazu zależnych od wartości prezentowanych parametrów. Mogą to być zarówno wypisywane liczby, jak i inne atrybuty obrazu.

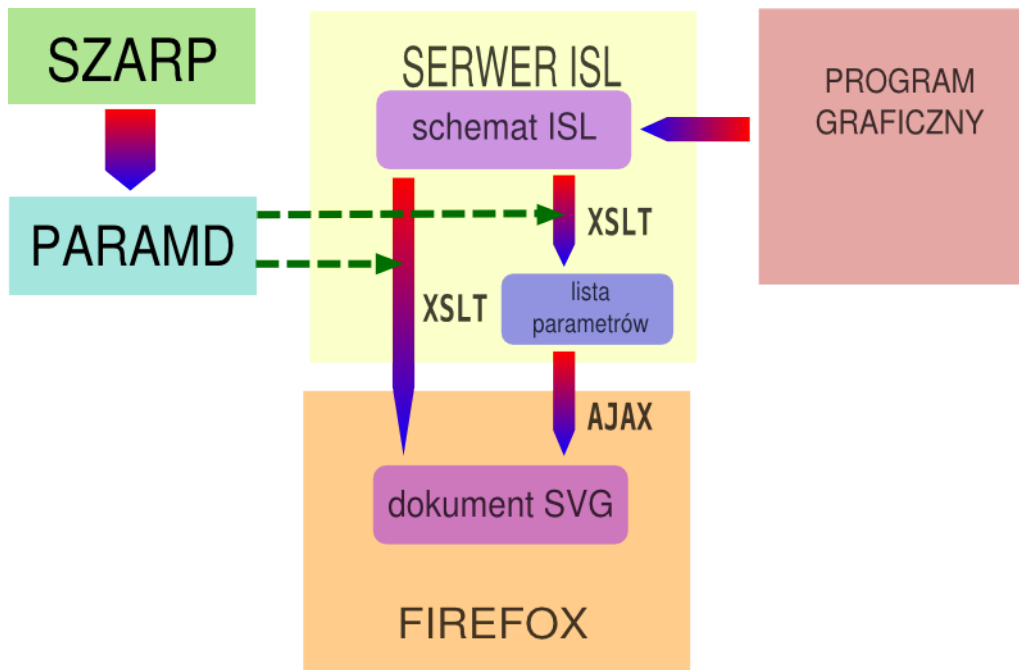
Przeglądarka Firefox pobiera dokument SVG z serwera ISL. Jest nim specjalnie przygotowana konfiguracja serwera Apache, czyli darmowego, najpopularniejszego na świecie serwera WWW. Serwer ISL ma 2 zadania - przygotowanie wyświetlanego przez przeglądarkę dokumentu SVG, a także odpowiadanie na zadawane przez kod JavaScript pytania o aktualne wartości parametrów.

Zarówno dokument SVG jak i dane o wartościach parametrów powstają na podstawie informacji zawartych w przygotowanym przez osobę konfigurującą system schemacie ISL. Schemat ISL jest statycznym (a więc nie zawierającym JavaScript) dokumentem SVG, zawierającym dodatkowo informacje na temat elementów rysunku zależnych od parametrów z systemu SZARP. Możliwość mieszania danych różnego typu (grafika wektorowa i dane o parametrach) jest jedną z istotniejszych cech standardu XML, z którego korzysta SVG. Schemat ISL przygotowuje się w programie graficznym - dalsze rozdziały zawierają dokładniejsze informacje na ten temat.

Serwer ISL, po otrzymaniu od przeglądarki użytkownika żądania pobrania schematu wizualizacji, wczytuje schemat ISL i zastępuje występujące w nim odniesienia do parametrów przez ich wartości. Podobnie odbywa się aktualizacja wyświetlanych wartości przez kod JavaScript. Wykorzystany jest tu mechanizm o nazwie AJAX. JavaScript wysyła co zadany czas (np. 10 sekund) do serwera żądanie pobrania dokumentu XML zawierającego aktualne wartości parametrów. Serwer, na podstawie schematu ISL, ustala jakie parametry w nim występują, a następnie przesyła ich listę z aktualnymi wartościami do przeglądarki. Kod JavaScript wczytuje tą listę i podmienia wyświetlane elementy, tak aby pokazać świeżo pobrane, być może zmienione wartości parametrów. W ten sposób nie trzeba pobierać całego dokumentu, także modyfikacja elementów w przeglądarce jest szybsza niż rysowanie całego obrazu od nowa.

Przetwarzanie schematów ISL na wyświetlane dokumenty SVG oraz na listę aktualnych wartości parametrów wykonywane jest za pomocą uruchamianego przez serwer Apache fragmentu kodu w języku Python, który to z kolei używa szablonów XSLT. XSLT to kolejny oparty na XML język, służący do opisu przekształceń (transformacji) dokumentów XML. W trakcie przekształceń, poza dodawaniem elementów (takich jak np. kod w JavaScript) czy usuwaniem zbędnych (lista aktualnych wartości parametrów jest tak naprawdę nadal schematem ISL, ale z usuniętymi wszystkimi elementami poza informacjami o parametrach), konieczne jest pobieranie wartości parametrów z systemu SZARP.

Szablony XSLT wyposażone są w mechanizm pozwalający na pobieranie fragmentów innych dokumentów XML. Wartości parametrów są więc udostępniane serwerowi ISL w postaci dokumentów XML przez serwer parametrów systemu SZARP, program paramd (zobacz Sekcja 13.3).



13.4.2. Instalacja i konfiguracja serwera ISL

W przypadku instalacji korzystających ze wsparcia firmy Praterm serwer ISL zwykle będzie zainstalowany i skonfigurowany. Zrobienie tego we własnym zakresie jest także dość proste.

Serwer ISL instalujemy na maszynie, na której zainstalowany i uruchomiony jest system SZARP - zakładam, że jest to komputer z Debianem. Instalacja sprowadza się do zainstalowania za pomocą komendy np. **apt-get** pakietu *szarp-xsltd* (wymagane są uprawnienia administratora):

```
# apt-get install szarp-xsltd
```

Jako zależności zainstalowany zostanie pakiet *szarp-paramd* oraz serwer Apache wraz z modulem *mod-python*. Do konfiguracji serwera Apache zostanie dodany *site* (czyli logiczny serwer) obsługujący ISL, na porcie 8083. Plik konfiguracyjny serwera Apache to **/etc/apache2/sites-available/szarp**, będący linkiem na plik **/opt/szarp/resources/xsltd/apache2-szarp**.

Następnie musimy upewnić się, że w konfiguracji systemu SZARP włączone jest uruchamianie programów *paramd* i tworzenie linku do konfiguracji dla serwera ISL. W pliku **/etc/szarp/parstart.cfg** sprawdzamy, czy w pierwszej sekcji (przeznaczonej dla serwera) zmienne *PARAMD* i *XSLTD* ustawione są na *1*. Jeśli nie, zmieniamy ich wartość a następnie restartujemy oprogramowanie SZARP za pomocą komendy (wymagane są uprawnienia administratora):

```
# /etc/init.d/parstart restart
```

Jeśli serwer Apache był wcześniej zainstalowany, także może być konieczne jego przeładowanie:

```
# /etc/init.d/apache2 reload
```

Po restarcie SZARP'a katalog **/etc/szarp/default** powinien być linkiem na katalog z konfiguracją SZARP'a, czyli **/opt/szarp/prefix**, gdzie *prefix* oznacza unikalny identyfikator konfiguracji, podany przy konfiguracji pakietu *szarp-server* (w przypadku "ciepłowniczych" instalacji SZARP jest to zwykle czteroliterowy skrót nazwy miasta, np. *prza* dla Przasnysza). W tym katalogu, w podkatalogu **config/isl** będziemy umieszczać schematy ISL.

Dodatkowo wszystkie pliki w katalogu **/opt/szarp/prefix/config** o nazwie odpowiadającej wzorcowi **.apache* zostaną wczytane do pliku konfiguracyjnego serwera Apache. Pozwala to na ustawienie dodatkowych opcji. Przykładowy plik **index.apache** ustala przekierowania domyślnie ładowanej strony - domyślnie ładowany będzie plik **schemat.isl**:

```
RewriteRule ^/isl/$ /isl/schemat.isl [R,L]
RewriteRule ^/isl$ /isl/schemat.isl [R,L]
RewriteRule ^/$ /isl/schemat.isl [R,L]
RewriteRule ^/isl/index.html$ /isl/schemat.isl [R,L]
```

Po zmianach w pliku należy przeładować serwer Apache komendą:

```
# /etc/init.d/apache2 reload
```

Innym istotnym plikiem, którego lokalizację warto zapamiętać, jest plik **/opt/szarp/prefix/config/params.xml**, zawierający konfigurację serwera SZARP dla danego serwera.

13.4.3. Uruchomienie przykładowego schematu

Rozpoczniemy od uruchomienia przykładowego schematu. Znajduje się on w pliku **termometr.isl**, który możemy znaleźć albo w źródłach SZARP, albo w zainstalowanym pakiecie, w podkatalogu **isl/examples/isl/new**.

Plik kopiujemy do katalogu **/opt/szarp/prefix/config/isl**. Upewniamy się, że w pliku jest umieszczone odpowiednie odniesienie do parametru SZARP. W tym celu otwieramy plik w dowolnym edytorze tekstowym (najlepiej wyposażonym w podświetlanie składni) i szukamy ciągu znaków *isl:href* (o jego dokładnym znaczeniu dowiemy się później). Cała znaleziona linia powinna mieć postać:

```
isl:uri="http://localhost:8081/Siec/Sterownik/temperatura_zewnetrzna@value"
```

Możemy zaznaczyć ten link (to co jest w środku cudzysłowu) i skopiować go do paska adresu przeglądarki WWW.

Ostrzeżenie

Jeśli przeglądarkę uruchamiamy na innym komputerze niż serwer SZARP, to musimy zamiast localhost podać adres serwera i upewnić się, że program paramd będzie przyjmował połączenia z naszego komputera.

Przeglądarka powinna wyświetlić wartość parametru, np. w postaci:

```
<?xml version="1.0"?>
<params xmlns="http://www.praterm.com.pl/ISL/params">
<attribute name="value">16.9</attribute>
</params>
```

Jeśli otrzymamy komunikat, że nie można połączyć się z serwerem, należy sprawdzić czy program paramd jest odpowiednio skonfigurowany, działa i przyjmuje połączenia z naszego adresu.

Jeśli zaś otrzymamy informację, że nie znaleziono strony ("Węzeł niezaleziony"), oznacza to że konfiguracji naszego serwera SZARP nie ma parametru o nazwie odpowiadającej ścieżce podanej w pliku termometr.isl. Musimy podać jakiś inny parametr, znalezienie go zaczynamy od wpisania w pasek adresu przeglądarki pierwszej części poprzednio podanego adresu, czyli np. <http://localhost:8081/>. Powinniśmy zobaczyć stronę zawierającą specjalny link *Reports* oraz nazwy grup parametrów dostępnych na serwerze.

Notatka: Nazwy parametrów SZARP składają się z trzech części oddzielonych dwukropkiem, z których pierwszą nazywamy zwykle grupą, a drugą jednostką. Np. w przypadku parametru *Sieć:Sterownik:temperatura zewnetrzna* mamy do czynienia z grupą *Sieć*, jednostką *Sterownik* i parametrem *temperatura zewnetrzna*. Przy podawaniu adresów dla programu paramd dwukropki zastępujemy ukośnikami.

Wybieramy grupę, jednostkę i parametr (patrz informacja wyżej), przy czym najlepiej aby parametr ten miał wartości mieszczące się w zakresie podobnym jak temperatura zewnetrzna (czyli -40 do +50), w przeciwnym razie nasz termometr będzie miał problem z wyświetleniem wartości parametru. Podmieniamy odnośnik w pliku termometr.isl na adres uzyskany w przeglądarce, nie zapominając dodać części po znaku @. W pliku znajduje się drugi odnośnik do tego samego parametru - powtarzamy szukanie *isl:href* i podmieniamy drugi wpis (zachowując część po znaku @). Zapisujemy zmodyfikowany plik.

Następnie uruchamiamy przeglądarkę Firefox i wpisujemy adres <http://localhost:8083/isl/termometr.isl>. Jeśli wszystko zrobiliśmy poprawnie, to powinniśmy zobaczyć termometr pokazujący wartość wpisanego do pliku parametru.

Po modyfikacji pliku (albo dodaniu nowego) nie jest konieczne restartowanie żadnej usługi - jedynie użycie przycisku "Odśwież" w przeglądarce.

13.4.4. Co można umieścić w schemacie

Poniższy rozdział opisuje budowę plików ISL. Jeśli zamierzasz jedynie wklejać wartości tekstowe parametrów do schematów, możesz go na razie opuścić i przejść do kolejnego rozdziału opisującego edycję schematów za pomocą programu Inkscape.

Jak już pisaliśmy dokumenty ISL są plikami graficznymi w formacie SVG, z zwartymi dodatkowymi informacjami dotyczącymi pobierania wartości parametrów z systemu SZARP. Sam format SVG oparty jest na standardzie XML. Więcej o standardzie XML można znaleźć np. w Wikipedii (<http://pl.wikipedia.org/wiki/XML>), dla nas bardzo istotne jest, że dokumenty XML są plikami tekstowymi, które można oglądać i edytować w zwykłym edytorze tekstowym.

Notatka: Jeśli używasz Windows, nie próbuj otwierać dokumentów XML w Notatniku - prawdopodobnie otrzymasz jedną długą, nieczytelną linię - Notatnik nie obsługuje końców linii typu uniksowego. Jeśli nie masz żadnego lepszego edytora, możesz użyć Wordpada.

Rysunki SVG są zwykle na tyle skomplikowanym dokumentem, że jego edycja ręczna (w edytorze tekstowym) nie jest możliwa. Ale elementów związanych z ISL jest niedużo i można je łatwo wyszukać w dokumencie.

Wszystkie istotne elementy występują w przykładowym dokumencie `termometr.isl`, który zainstalowaliśmy w poprzednim rozdziale. Otwieramy ten dokument w edytorze tekstowym i szukamy ciągów znaków *isl*. Po kolei w dokumencie napotkamy:

```
<svg xmlns:isl="http://www.praterm.com.pl/ISL/params" ... />
```

Jest to deklaracja przestrzeni nazw ISL, która mówi że elementy i atrybuty o nazwach zaczynających się od *isl*: należą do przestrzeni nazw ISL. Słowo wyjaśnienia: dokument XML składa się z elementów, które mogą zawierać atrybuty opisujące te elementy, a także zawartość tekstową. Atrybuty posiadają nazwę i wartość. Elementy mogą być dowolnie zagnieżdżane. Początek i koniec elementu oznaczamy nazwą elementu ujętą w nawiasy ostre (przy czym oznaczenie końca ma dodatkowo jeszcze ukośnik), wartości atrybutów zapisujemy w cudzysłowie. Przykładów reprezentacja struktury książki w XML:

```
<książka>
<rozdział numer="1" tytuł="Wstęp">
To jest treść rozdziału 1.
</rozdział>
<rozdział numer="2" tytuł="Wielki wybuch">
To jest treść rozdziału 2.
</rozdział>
<rozdział numer="3" tytuł="Epilog">
To jest treść rozdziału 3.
</rozdział>
</książka>
```

Mechanizm przestrzeni nazw pozwala na mieszanie w jednym dokumencie XML elementów i atrybutów różnych typów - przez przedrostek *isl*: oznaczamy elementy i atrybuty, które nie mają być traktowane jako obiekty graficzne SVG, ale jako dodatkowe informacje związane z ISL. Tak więc powyższa deklaracja przestrzeni nazw jest pierwszym i niezbędnym elementem do działania całego systemu.

```
<rect style="..." ...
height="305.396825"
isl:uri="http://localhost:8081/Siec/Sterownik/temperatura_zewnetrzna@value"
isl:target="height"
isl:shift="145"
isl:scale="2.8"
... />
```

Tu mamy do czynienia z dość skomplikowanym przykładem. Element *rect* opisuje prostokąt, atrybut *height* odpowiada jego aktualnej wysokości. Na rysunku prostokąt ten reprezentuje słupek rtęci termometru. Chcemy, aby słupek ten zwiększał się wraz ze wzrostem temperatury. Atrybut *isl:uri* zawiera link do parametru, którego wartość chcemy reprezentować. Końcówka *@value* oznacza, że interesuje nas wartość liczbową parametru (parametr ma też inne właściwości, takie jak np. jednostka czy nazwa). Atrybut *isl:target* informuje nas, że wartość parametru będzie wpływała na atrybut *height* elementu *rect*, czyli na wysokość prostokąta. W ten sposób uzależniamy wysokość prostokąta (słupka rtęci) od wartości parametru.

Nie jest to jednak wystarczające - wysokość prostokąta powinna przyjmować wymiary wynikające z jego położenia na rysunku, które raczej nie będą bezpośrednio odpowiadać mierzonej temperaturze - np. dla temperatury 0 stopni słupek powinien mieć około połowy swojej maksymalnej wielkości. Do przeskalowania temperatury na odpowiednią wysokość służą atrybuty liczbowe *isl:scale* i *isl:shift*. Wartość atrybutu *height* wyliczana jest ostatecznie według wzoru:

```
height = [wartość parametru] * isl:scale + isl:shift
```

Współczynniki skalowania i przesunięcia dobrać można eksperymentalnie - po narysowaniu słupka rてcy w programie graficznym w pozycji 0 należy sprawdzić aktualną wysokość słupka i przyjąć ją jako *isl:shift*, następnie narysować słupek w pozycji +50 stopni i zobaczyć o ile zwiększyła się wysokość prostokąta - wartość ta podzielona przez 50 da prawidłową wartość *isl:scale*.

Podobnej transformacji mogą podlegać inne atrybuty. Dostępne atrybuty i ich znaczenie opisane są w specyfikacji SVG, dostępnej na stronach World Wide Web Consortium (<http://www.w3.org/Graphics/SVG/>)

```
<tspan ...
isl:uri="http://localhost:8081/Siec/Sterownik/temperatura_zewnetrzna@v_u"
>?</tspan>
```

Ten przykład jest dużo prostszy - mamy do czynienia z elementem *tspan*, reprezentującym fragment tekstu. Zawartość tego fragmentu ma być zastąpiona wartością parametru podanego w atrybucie *isl:uri*. Zwróćmy uwagę, że używamy tu atrybutu parametru o nazwie *v_u* - jest to skrót od *value* i *unit* - czyli otrzymujemy wartość parametru wraz z towarzyszącą mu nazwą jednostki. Jest to najprostsza konstrukcja, pozwalająca na wklejenie do schematu napisu z aktualną wartością parametru.

13.4.5. Edycja schematów w programie Inkscape

Możliwa jest ręczna edycja plików SVG zgodnie z zasadami opisanymi w rozdziale powyżej, nie jest to jednak zbyt wygodne. Prostszy sposób edycji całości schematu, zarówno jego postaci graficznej, jak i informacji związanych z parametrami, jest wykorzystanie programu Inkscape wraz z odpowiednią wtyczką.

Inkscape jest darmowym programem do edycji grafiki wektorowej, ze strony www.inkscape.org (<http://www.inkscape.org>) można ściągnąć wersję dla Windows, pakiety dla większości dystrybucji powinny być dostępne bezpośrednio w ramach dystrybucji. Dla Debiana instalacja sprowadza się do wydania komendy **apt-get install inkscape**.

13.4.5.1. Instalacja wtyczki do edycji schematów ISL

Do edycji schematów ISL potrzebna jest wtyczka, którą można bezpłatnie pobrać ze stron SZARP'a.

- Jeśli korzystamy z Debiana, wtyczkę możemy zainstalować komendą:

```
# apt-get install szarp-isledit
```
- Dla innych dystrybucji Linuksa musimy ściągnąć/skompilować program *isledit*, umieścić go katalogu **/opt/szarp/bin**, ew. pliki tłumaczenia umieścić w katalogu **opt/szarp/resources/locales/pl_PL/LC_MESSAGES/**. Następnie plik *isledit.inx* ze źródeł SZARP'a kopiujemy do katalogu ze wtyczkami Inkscape, np. **/usr/share/inkscape/extensions**.
- Jeśli korzystamy z Windows, musimy ściągnąć i zainstalować wersję SZARP dla Windows. Następnie plik *isledit.inx* znajdujący się przy domyślnej instalacji w katalogu **C:\Program Files\Szarp\resources** kopiujemy do katalogu ze wtyczkami Inkscape - domyślnie jest to **C:\Program Files\Inkscape\share\extensions**. Jeśli instalowaliśmy SZARP'a w innym miejscu niż domyślne, musimy za pomocą edytora tekstowego poprawić ścieżki wpisane w pliku *isledit.inx* (ten plik też jest w formacie XML).

Jeśli wtyczka została poprawnie zainstalowana, po uruchomieniu programu Inkscape w menu *Efekty* powinniśmy mieć dostępną pozycję *ISL Editor*.

13.4.5.2. Konfiguracja wtyczki - wybór konfiguracji

Po wybraniu z menu *Efekty* opcji *ISL Editor* pokaże się okienko, w którym musimy podać ścieżkę do pliku *params.xml* z konfiguracją SZARP, z której będziemy chcieli pobierać parametry. Można wykorzystać plik z konfiguracji obecnej na komputerze na którym pracujemy, możemy też po prostu skopiować odpowiedni plik *params.xml*.

13.4.5.3. Tworzenie elementów tekstowych

Na początek spróbujemy utworzyć schemat wyświetlający w postaci tekstowej wartość parametru. W tym celu wybieramy z paska narzędzi programu narzędzie tworzenie obiektów tekstowych (klawisz skrót F8) i dodajemy do rysunku dowolny tekst - niech będzie to np. ?. Następnie wybieramy narzędzie zaznaczania (F1), możemy trochę zwiększyć rozmiar wprowadzonej litery rozciągając ją. Mając zaznaczony nasz nowy obiekt wybieramy z menu *Efekty* opcję *ISL Editor*. W okienku podajemy ścieżkę do konfiguracji SZARP (pliku *params.xml*) - program zapamięta wprowadzoną wartość, więc za kolejnym razem możemy po prostu kliknąć Ok.

W okienku uruchomionej wtyczki na górze znajduje się pole ze ścieżką do wybranego parametru (jeśli stworzymy element, to będzie ono początkowo puste). Za pomocą przycisku **Wybierz parametr** otwieramy okienko z drzewkiem parametrów do wyboru - wybieramy jakiś parametr i klikamy Ok. Domyślnie wstawiana jest tylko wartość parametru (ścieżka kończy się na *@value*), jeśli chcemy aby pokazywana była także jednostka, zaznaczamy pole **Pokaż nazwę jednostki** - końcówka ścieżki przyjmie wartość *@v_u*.

To wszystko - po zamknięciu okienka i zapisaniu pliku możemy go skopiować na serwer SZARP do katalogu ze schematami i uruchomić przeglądarkę żeby obejrzeć stworzony schemat.

13.4.5.4. Modyfikacja atrybutów elementów

Uzyskanie np. efektu słupka rtęci którego wysokość zależy od temperatury wymaga uzależnienia wysokości prostokąta reprezentującego słupek termometru od wartości parametru. W tym celu, podobnie jak opisano wyżej, zaczynamy od utworzenia obiektu na którym będzie operować - tworzymy prostokąt (skrót F4), następnie mając go zaznaczonego uruchamiamy z menu wtyczkę. Wybieramy parametr i klikamy na opcję "Zastąp wartość atrybutu". Z rozwijanej listy wybieramy atrybut zaznaczonego obiektu, który chcemy uzależnić od wartości parametru.

14. Aplikacja SSS.

14.1. Krótki opis.

SSS(Szarp Sync Server) to serwer synchronizacji plików. Uruchomiony na serwerze, pozwala użytkownikom na pobieranie danych historycznych za pomocą aplikacji SSC. Poza serwerem głównym mogą istnieć mirrory, które udostępniają wszystkie lub tylko część baz dostępnych na serwerze głównym. Konfiguracja użytkowników jest trzymana na serwerze głównym i synchronizowana na mirrory. Użytkownicy z reguły zawsze łączą się na serwer główny, który ew. odsyła klientowi informację o przekierowaniu na mirror.

14.2. Konfiguracja programu.

Program czyta konfigurację z sekcji sss pliku `szarp.cfg`. Obsługiwane opcje:

- *key_file* - ścieżka do pliku zawierającego klucz prywatny wykorzystywany do szyfrowania przesyłanych danych
- *ca_file* - ścieżka do pliku zawierającego klucz publiczny centrum autoryzacji, przez które został podpisany klucz prywatny
- *passphrase* - hasło do używanego klucza prywatnego
- *user* - identyfikator użytkownika, z którego uprawnieniami ma chodzić proces obsługujący połączenia z klientem
- *userdbfile* - ścieżka do pliku zawierającego listę użytkowników

Wszystkie w/w opcje są obowiązkowe.

Na serwerze (mirrorze) instalujemy również paczkę `szarp-sss`. Zainstaluje ona dwa skrypty. `update_ssuserdb.sh` i `sss_bases.pl`. Pierwszy jest odpalany z `crona` co 15 min i aktualizuje bazę użytkowników, więc wszystkie wprowadzone zmiany przez interfejs `www`, będą zaktualizowane na wszystkich mirrorach za 15 minut. Drugi skrypt tworzy listę baz dostępnych na danym mirrorze raz na dobę (coś około północy).

Na początku możemy odpalić skrypty ręcznie, ponieważ interfejs `www` wymaga obecności pliku `sss_bases` w `/opt/szarp/PREFIX/`. Inaczej po dodaniu serwera nie zobaczymy go na liście serwerów na głównej stronie.

14.3. Baza użytkowników.

Baza użytkowników ma postać pliku XML'owego. Ścieżka do tego pliku powinna zostać podana w sekcji sss pliku `szarp.cfg`. Przykładowa zawartość pliku:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<users xmlns="http://www.praterm.com.pl/SZARP/sync-users">
  <user name="user" password="098f6bcd4621d373cade4e832627b4f6" basedir="/opt/szarp" s
  <user name="user2" password="ad0234829205b9033196ba818f7a872b" basedir="/opt/szarp"
  <server name="prat" ip="62.233.142.85"/>
  <server name="kato" ip="222.222.222.1"/>
</users>
```

Pojedynczego użytkownika opisuje element *user*. Wszystkie pokazane na przykładzie atrybuty tego elementu są obowiązkowe, a ich znaczenie następujące:

- *name* - nazwa użytkownika
- *password* - skrót MD5 hasła użytkownika. Można go wygenerować np. komendą `echo -n "haslo" | md5sum`
- *basedir* - katalog bazowy, tj. katalog którego podkatalogi są wybierane do synchronizacji
- *sync* - wyrażenie regularne opisujące podkatalogi katalogu bazowego, które zostają użytkownikowi udostępnione

- *hwkey* - klucz sprzętowy danego komputera. Pozostawienie pustego pola oznacza klucz wyłączony, czyli akceptowany będzie każdy klucz przysłany przez klienta. Klucz o wartości 0 oznacza wyłączenie możliwości logowania się przez użytkownika. Klucz o wartości -1 spowoduje, że przy pierwszym połączeniu klucz zostanie zarejestrowany, czyli pole uzupełnione.
- *expired* - data ważności konta w formacie YYYYMMDD, gdzie "-1" oznacza że konto nigdy nie straci ważności.
- *server* - nazwa konfiguracji z szarp.cfg serwera do którego jest przypisany użytkownik np. dla rambo jest to prat
- *comment* - komentarz na temat użytkownika, widoczny tylko przez administratora.

Pojedynczy server opisuje element *server*. Wszystkie pokazane na przykładzie atrybuty tego elementu są obowiązkowe, a ich znaczenie następujące:

- *name* - nazwa konfiguracji z szarp.cfg server sss
- *ip* - adres ip serwera sss

Po zmianie zawartości bazy użytkowników należy wysłać do programu SSS sygnał *HUP*, co spowoduje wczytanie zaktualizowanej wersji bazy.

14.4. Protokół SSS/SSC

Jest to opis wersji 2 protokołu sss/ssc. Po zainicjalizowaniu połączenia przez klienta protokół komunikacji wygląda następująco:

- *klient* - wysyła liczbę 2 bajtową oznaczającą numer protokołu z jakiego korzysta. Dzięki temu możliwa jest obsługa starych klientów. Server jest odpowiedzialny za poprawną obsługę protokołów.
- *klient* - wysyła trzy stringi, nazwę użytkownika, hasło, i wygenerowany klucz sprzętowy, kończy wiadomość.
- *server* - sprawdza poprawność hasła i klucza sprzętowego, oraz czy użytkownik jest przypisany do tego serwera sss. W odpowiedzi wysyła liczbę 2 bajtową zawierającą wiadomość zdefiniowaną w strukturze Message, czyli AUTH_OK lub AUTH_FAILURE lub AUTH_REDIRECT.
- *server* - jeśli autoryzacja przebiegła pomyślnie (AUTH_OK) serwer przysyła liczbę 2 bajtową oznaczającą komunikat (np zbliżająca się utrata ważności konta, stara wersja protokołu itp). jeśli autoryzacja nie udała się, server przysyła wiadomość w postaci liczby 2 bajtowej z komunikatem dlaczego np(konto nieważne), jeśli autoryzacja jest przekierowana na inny serwer, to wysyłany jest string zawierający adres ip poprawnego serwera.
- *server* - wysyła liczbę baz do pobrania, "0" oznacza błąd, dalej synchronizacja tak jak było... todo

14.5. Interfejs WWW do administracji użytkownikami

Dla ułatwienia życia administratora i użytkowników, oraz aby dać programistom SZARP możliwość pobawienia się fajnym pythonowym frameworkiem, powstał interfejs WWW do administracji użytkownikami SSS. Interfejs ten pozwala na tworzenie, edycję i usuwanie kont użytkowników przez administratora, użytkownikom daje możliwość podglądu ich ustawień oraz zmiany/przypomnienia hasła.

Interfejs składa się z 3 niezależnych elementów, z których każdy może znajdować się fizycznie na innej maszynie.

- Skrypt `/opt/szarp/bin/ssconf.py` odpowiada za bezpośrednie modyfikacje pliku XML z konfiguracją użytkowników, musi być więc zainstalowany bezpośrednio na maszynie na której działa serwer SSS. Korzysta z konfiguracji w pliku `szarp.cfg` (ścieżka do bazy itp.) oraz z dodatkowego pliku `/etc/szarp/ssconf.cfg`. Zawiera on login i hasło administratora (dlatego też powinien być czytany tylko przez root'a), nazwę głównego serwera oraz adres TCP/IP na którym skrypt ma słuchać - skrypt jest serwerem XML-RPC słuchającym na podanym adresie. Dostęp do serwera wymaga autoryzacji - albo hasłem administratora, albo konkretnego użytkownika - wtedy dostępne są tylko operacje dla konkretnego użytkownika. Uruchamianie skryptu sterowane jest przez zmienną `SSCONF` w pliku konfiguracyjnym `/etc/szarp/parstart.cfg`, a sam skrypt wchodzi w skład paczki `szarp-daemons`.

Program `ssconf.py` ma też drugi tryb działania, pozwalający na rejestrację z linii poleceń nowego klucza użytkownika jeśli klucz dotychczas był w stanie 'Waiting' (-1). W tym celu łączy się do swojej kopii działającej jako serwer - pozwala to uniknąć problemów z synchronizacją. Program w tym trybie jest wywoływany przez serwer SSS po tym jak użytkownik podłączy się do SSS w celu rejestracji nowego klucza.

- Aplikacja WWW wykorzystująca framework Pylons - konfigurowana przez plik `/etc/szarp/ssweb.ini`. Aplikacja serwuje strony WWW, do pobierania i modyfikacji danych łączy się przez protokół XML-RPC ze skryptem `ssconf.py`. Aplikacja powinna mieć dostęp do działającego serwera SMTP - jest to niezbędne do wysyłania użytkownikom informacji o nowym czy zresetowanym hasle. W domyślnej konfiguracji administrator dostaje też powiadomienia mailem o błędach w działaniu aplikacji.

Aplikacja wchodzi w skład oddzielnej paczki `szarp-sssweb` (nie wymagającej innych elementów SZARP'a). Może być uruchomiona jako samodzielny serwer (przydatne do testowania i debugingu), albo jako skrypt FastCGI (dokładniej tzw. External Fast CGI Server) do którego łączy się trzeci element - serwer WWW.

- Trzeci element to serwer WWW obsługujący FastCGI, a dokładniej konfigurację typu External Server. W pliku `README.Debian` paczki `szarp-sssweb` znajduje się przykład konfiguracji wykorzystującej serwer Apache2 i moduł `mod_fastcgi`.

14.6. Serwer komentarzy (remarks server)

14.6.1. Konfiguracja

Aby uruchomić `remarks_server.py` potrzebny jest plik konfiguracyjny, oraz postgresowa baza danych. Przykładowy plik konfiguracyjny wygląda następująco:

```
[database]
name=remarks
user=remarks_server
password=zażółć_gęśła_jażń
server_name=localhost
[connection]
host=127.0.0.1
port=5432
```

```
certfile=/etc/szarp/sss_ca.pem
keyfile=/etc/szarp/sss_key.pem
```

Wszystkie pozycje są obowiązkowe. Polcany właściciel bazy jest taki jak w pliku, ponieważ schemat bazy zawiera odwołania do tego użytkownika. Podany port jest domyślnym portem na którym stawiany jest serwer postgresa.

Plik konfiguracyjny powinien nazywać się `/etc/szarp/remark_server_config.ini`

14.6.2. Baza danych

Serwer komentarzy używa postgresowej bazy danych. Jej schemat jest w repozytorium razem z kodem programu jako `utils/remarks_server/remarks.schema`. Aby uruchomić program należy w `/etc/szarp/parstart.cfg` ustawić zmienną `REMARKS_SERVER` na 1. Specyficznym zachowaniem programu jest to, że nie da się go wystartować bez wpisu o serwerze do bazy danych.

Tak więc aby stworzyć bazę danych należy wykonać na bazie następujące polecenia:

```
createuser -P remarks_server
createdb -O remarks_server remarks
psql remarks < remarks.schema
```

15. Komputery przenośne.

15.1. Spis treści

15.2. Krótki opis i wymagane biblioteki.

Najczęściej komputery przenośne korzystają z modemów Sony Ericsson GCxx oraz połączeń EDGE-GPRS (ISP z reguły ERA/IDEA). Dokładny opis instalacji modemów z rodziny SE GCxx znajdziemy w HOWTO. Do obsługi/konfiguracji komputerów przenośnych wymagane są dwa skrypty: `ipk2mobile.pl` i `switch_controller.pl`.

Wymagane biblioteki:

- libxml-parser-perl
- xdialog
- edge-gprs

15.3. Konfiguracja.

Oba skrypty (ipk2mobile.pl, switch_controller.pl) znajdujące się z reguły w /usr/work/szarp/script/mobile musimy przegrać do /opt/szarp/bin.

Params.xml:

- Element "params" - musimy dopisać np. xmlns:switch="http://www.praterm.com.pl/SZARP/ipk-extra". "switch" możemy zastąpić czymkolwiek innym. Ponadto musimy dopisać użytkownika, który będzie przełączał typy regulatorów (najczęściej palacz), czyli dopisujemy: switch:user="palacz". Przykładowy element params:

```
params xmlns="http://www.praterm.com.pl/SZARP/ipk" xmlns:exec="http://www.praterm.com.pl/SZARP/ipk-exec"
xmlns:switch="http://www.praterm.com.pl/SZARP/ipk-extra" version="1.0" read_freq="1000000000"
```

- Element "device" - jako ścieżkę podajemy zmyśloną sobie nazwę regulatora np. np1036, i umieszczamy prawdziwą ścieżkę w atrybucie switch:real_path="...". Musimy jeszcze umieścić atrybut switch:name="...", który odpowiada za nazwę typu regulatora widoczną w scc.
- Ostatnim elementem będzie daemon testdmn odpowiedzialny za rejestrowanie numeru podłączonego węzła. Ścieżkę musimy podać następującą: "/opt/szarp/\$prefix/regulator_name", gdzie oczywiście \$prefix zamieniamy na odpowiedni prefix bazy. Przykładowa konfiguracja:

```
device daemon="/opt/szarp/bin/testdmn" path="/opt/szarp/szw5/regulator_name" exec:fr
unit id="1" type="1" subtype="1" bufsize="1"
  param name="Węzeł przesyłowy Sztum:Status:monitorowany węzeł" prec="0" base_ind="a
  define type="RPN" formula="NULL"/
  raport title="Status monitoringu"/
  draw title="Status monitoringu" min="0" max="100"/
  /param
/unit
/device
```

Ostatnim etapem jest uruchomienie ipk2mobile, z odpowiednim prefiksem,np.

```
/opt/szarp/bin/ipk2mobile.pl -p szw5
```

lub

```
/opt/szarp/bin/ipk2mobile.pl --prefix=szw5
```

Zostaną wygenerowane dwa pliki: /opt/szarp/\$prefix/sudoers i /opt/szarp/\$prefix/switch_menu.in. Ten pierwszy jest automatycznie przegrywany do /etc a ten drugi zawiera zmienna \$scc_switch_menu\$. Edytujemy więc szarp.cfg i inkludujemy drugi plik. Następnie dopisujemy w sekcji scc do zmiennej menu \$scc_switch_menu\$.np

```
:scc
```

```
menu = $scc_switch_menu$, \
...
```

Tak skonfigurowany komputer jest już gotowy.

15.4. Podstawy korzystania.

Aby zmienić typ regulatora musimy otworzyć scc menu i wybrać podmenu Zmień typ i wybrać szukany przez nas rodzaj regulatora. Następnie musimy z menu wybrać pole "Zmień nazwę" i w ukazanym okienku wybrać węzeł na którym znajduje się komputer przenośny, w przeciwnym wypadku wybrać pole "Dodaj nową nazwę". W przyszłości przewidziane jest że na wykresie Status Monitoringu zamiast liczb będą pojawiać się nazwy węzłów.

16. Tabela kodów ASCII

Tabela 1. Tablica kodów ASCII

ASCII	HEX	DEC
NUL (CTRL+@)	00	0
SOH (CTRL+A)	01	1
STX (CTRL+B)	02	2
ETX (CTRL+C)	03	3
EOT (CTRL+D)	04	4
ENQ (CTRL+E)	05	5
ACK (CTRL+F)	06	6
BEL (CTRL+G)	07	7
BS (CTRL+H)	08	8
HT (CTRL+I)	09	9
LF (CTRL+J)	0A	10
VT (CTRL+K)	0B	11
FF (CTRL+L)	0C	12
CR (CTRL+M)	0D	13
SO (CTRL+N)	0E	14
SI (CTRL+O)	0F	15
DLE (CTRL+P)	10	16
DC1 (CTRL+Q)	11	17
DC2 (CTRL+R)	12	18
DC3 (CTRL+S)	13	19
DC4 (CTRL+T)	14	20
NAK (CTRL+U)	15	21
SYN (CTRL+V)	16	22
ETB (CTRL+W)	17	23
CAN (CTRL+X)	18	24
EM (CTRL+Y)	19	25
SUB (CTRL+Z)	1A	26

ASCII	HEX	DEC
ESC (CTRL+])	1B	27
FS (CTRL+\)	1C	28
GS (CTRL+])	1D	29
RS (CTRL+^)	1E	30
US (CTRL+_)	1F	31
SPACE	00	0
!	21	33
"	22	34
#	23	35
\$	24	36
%	25	37
&	26	38
'	27	39
(28	40
)	29	41
*	2A	42
+	2B	43
,	2C	44
-	2D	45
.	2E	46
/	2F	47
0	30	48
1	31	49
2	32	50
3	33	51
4	34	52
5	35	53
6	36	54
7	37	55
8	38	56
9	39	57
:	3A	58
;	3B	59
<	3C	60
=	3D	61
>	3E	62
?	3F	63
@	40	64
A	41	65

ASCII	HEX	DEC
B	42	66
C	43	67
D	44	68
E	45	69
F	46	70
G	47	71
H	48	72
I	49	73
J	4A	74
K	4B	75
L	4C	76
M	4D	77
N	4E	78
O	4F	79
P	50	80
Q	51	81
R	52	82
S	53	83
T	54	84
U	55	85
V	56	86
W	57	87
X	58	88
Y	59	89
Z	5A	90
[5B	91
\	5C	92
]	5D	93
^	5E	94
_	5F	95
‘	60	96
a	61	97
b	62	98
c	63	99
d	64	100
e	65	101
f	66	102
g	67	103
h	68	104

ASCII	HEX	DEC
i	69	105
j	6A	106
k	6B	107
l	6C	108
m	6D	109
n	6E	110
o	6F	111
p	70	112
q	71	113
r	72	114
s	73	115
t	74	116
u	75	117
v	76	118
w	77	119
x	78	120
y	79	121
z	7A	122
{	7B	123
	7C	124
}	7D	125
~	7E	126
DEL	7F	127

17. Instalacja DDE Proxy w systemie Windows

17.1. Opis instalacji aplikacji DDE Proxy na platformie Windows

1. Rozpakować archiwum np do C:\ddeproxy 2. Uruchomić wiersz poleceń z prawami administratora: Start->Uruchom: runas /user:localhost/administrator cmd 3. W cmd: cd c:\ddeproxy ddeproxy.exe -install (to zainstaluje proxy jako usługę windows). 4. Start->Panel sterownia->Narzędzia administracyjne->Usługi (uwaga ta ścieżka do pod xp dla 2000 może być inna), prawym na DDE proxy service, i w zakładce Logowanie proszę zaznaczyć 'Zezwalaj na współdziałanie z pulpitem' oraz ustawić żeby usługa startowała automatycznie. 5. Aby zmiana ustawień z pkt 4 została uwzględniona należy zatrzymać usługę a następnie uruchomić ją ponownie. Usługa standardowo nasłuchuje połączeń na porcie 8080. Jeśli komputer ma uruchomioną zaporę albo zainstalowanego jakiegoś firewalla proszę odblokować połączenia przychodzące na ten port.