BRAMKA KSEF

PODRĘCZNIK PROGRAMISTY

Autor: **BinSoft**

5 września 2023

SPIS TREŚCI

WSTĘP

W 2022 roku Ministerstwo Finansów wprowadziło tzw. Krajowy System e-Faktur. W chwili obecnej korzystanie z niego jest dobrowolne, jednak od 1 lipca 2024 roku przesyłanie do niego faktur będzie obowiązkowe.
Krajowy System e-Faktur zakłada, że każda poprawna faktura powinna mieć postać ustrukturyzowanego pliku w formacie XML i zgodnego z opublikowaną przez Ministerstwo schemą. W chwili pisania tego podręcznika dostępne są dwie wersje schemy: e-Faktura FA (1) i wchodząca właśnie w życie e-Faktura FA (2).

Bramka KSeF to aplikacja pozwalająca na przesyłanie i odbieranie faktur z Krajowego Systemu e-Faktur. Nie służy ona do wystawiania takich dokumentów, a jedynie do ich przesyłania. Oznacza to, że faktury powinniśmy pobrać lub wyeksportować z naszego aktualnego systemu sprzedażowego lub księgowego, a z bramki skorzystać by je łatwo przesłać do Ministerstwa.

Bramka KSeF posiada także szereg różnych dodatkowych usług (m.in. serwerów), które pozwalają na integrację tej aplikacji z innymi systemami i aplikacjami. Każdy programista może się komunikować z Bramką KSeF na wiele dostępnych sposobów i w ten sposób przesyłać lub odbierać dokumenty z MF.

BRAMKA KSEF DLA PROGRAMISTÓW

Program Bramka KSeF posiada bardzo podstawowy interfejs pozwalający na wysyłanie i odbieranie dokumentów wprost z głównego okna. Tryb ten przeznaczony jest dla klientów indywidualnych kupujących program w wersji podstawowej, bezpośrednio od producenta lub partnerów. Aplikacja posiada jednak szereg mechanizmów, które pozwalają innym programom na łączenie się do niego i za jego pomocą obsługę wspomnianych funkcji. Oznacza to, że zewnętrzni programiści mogą wyposażyć swoje aplikacje w odpowiednie mechanizmy współpracy z Bramką KSeF i w ten sposób wzbogacić swoje aplikacje o obsługę KSeF.

NA CO POZWALA BRAMKA KSEF?

Bramka KSeF pozwala na nawiązanie z nią połączenia w jeden z następujących sposobów:

- poprzez komunikację TCP/IP,
- poprzez komunikację WebSocket (WS + WSS),
- poprzez lokalny serwer HTTP/HTTPS,
- poprzez współdzielony folder lokalny,
- poprzez folder na serwerze FTP/SFTP,
- poprzez bramę BinSoft Gate;

W dowolnej aplikacji desktopowej czy mobilnej - programista może w prosty sposób nawiązywać połączenie TCP/IP, użyć WebSocket lub wywoływać określony adres URL by łączyć się z serwerem HTTP. Nie ma znaczenia z jakiego języka programowania korzysta, czy też pod jaki system operacyjny tworzy swoją aplikację. Może to być Windows, macOS, Linux, iOS, Android... Ważne jest jedynie, aby na jednym stanowisku klienta był uruchomiony komputer z systemem Windows lub Mac, a na nim Bramka

KSeF. Aplikacje z dowolnych urządzeń w sieci mogą się z nim połączyć i korzystać z jego usług.

DOSTOSOWANIE APLIKACJI DO POTRZEB UŻYTKOWNIKA

Bramka KSeF umożliwia dostosowanie jej według wytycznych programisty. Możliwe jest wyłączenie trybu podstawowego - tak by użytkownik widział jedynie panel logów, miał dostęp do ustawień programu i ewentualnie przygotowanych przez programistę wtyczek. Wreszcie, programista może ustalić informacje adresowe jakie mają się pojawić w oknie "O programie" oraz... ustalić dowolną nazwę dla aplikacji. W ten sposób deweloper może przygotować własną wersję Bramki KSeF, opatrzoną własną nazwą i wyglądem, i taki program oferować swoim klientom - jako np. opcja dodatkowo płatna lub wraz z paczką instalacyjną swojej aplikacji.

LICENCJONOWANIE

Istnieją trzy modele współpracy licencyjnej dla aplikacji Bramka KSeF.

Wariant 1

Użytkownik końcowy może kupić aplikację Bramka KSeF od producenta BinSoft lub jednego z Partnerów. Następnie może go odpowiednio skonfigurować lub doinstalować do niego wtyczkę dostarczoną od developera danego rozwiązania (Partnera) lub stworzyć własną wtyczkę dostosowującą program do jego aplikacji.

Wariant 2

Firma BinSoft może przygotować indywidualną wersję aplikacji Bramka KSeF. Może ona mieć zupełnie zmieniony wygląd, nazwę, stosować z góry ustaloną metodę komunikacji itp. Deweloper otrzymuje dostęp do panelu B2B i tam generuje klucze dla swoich Klientów. Użytkownik końcowy będzie musiał zarejestrować swój program otrzymanym kluczem. Po roku będzie musiał również przedłużyć swoje klucze u Partnera. Ceny za poszczególne klucze to standardowe ceny z aktualnego cennika pomniejszone o rabat Partnera. Fakturowanie za wygenerowane klucze odbywa się raz na koniec miesiąca rozliczeniowego.

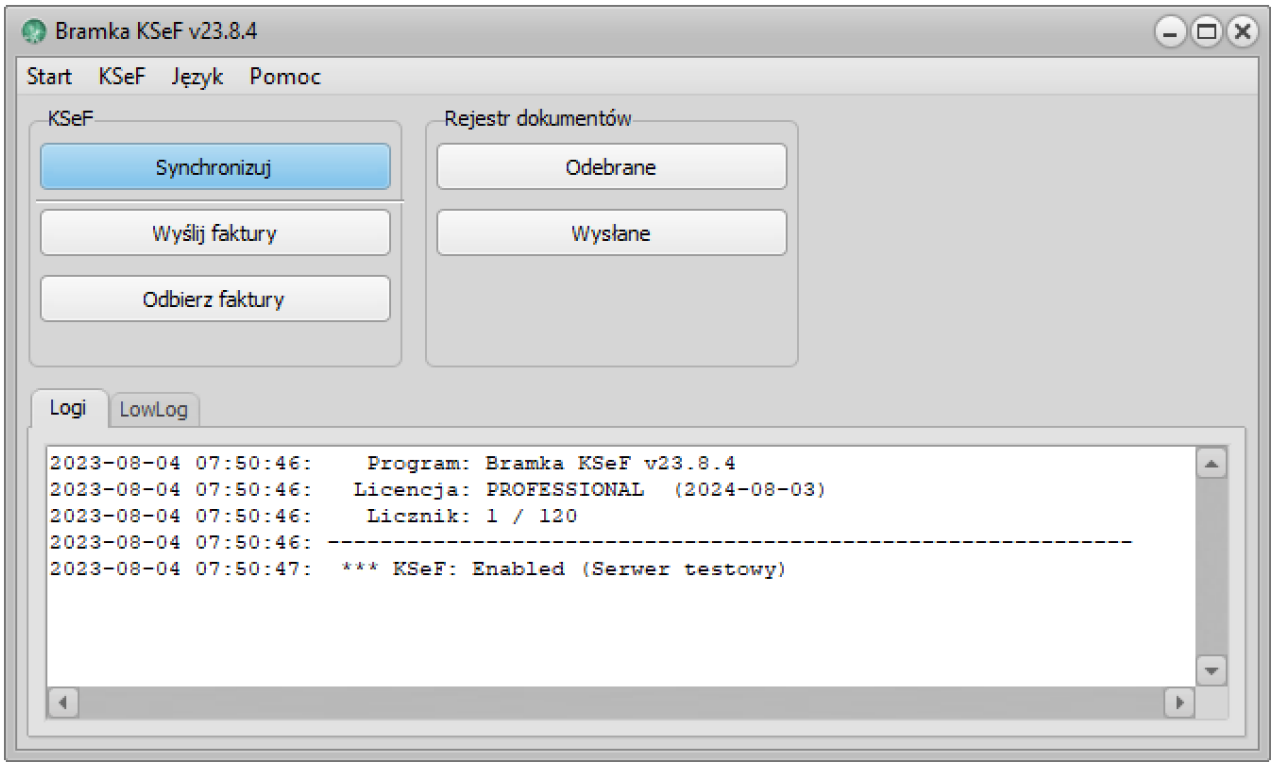
Wariant 3

Podobnie jak przy wariancie 2. firma BinSoft może przygotować indywidualną wersję aplikacji Bramka KSeF. Jest ona jednak pozbawiona zabezpieczenia kluczami. Użytkownik końcowy po jej instalacji nie musi rejestrować programu, przedłużać kluczy itp. Nie ma zatem żadnego ograniczenia co do liczby stanowisk czy ważności licencji. Koszt takiej wersji jest ustalany indywidualnie.

Aby uzyskać więcej szczegółów na temat licencjonowania, należy skontaktować się z producentem BinSoft pod adresem e-mail: kontakt@binsoft.pl i ustalić szczegóły współpracy.

INTERFEJS PROGRAMU

Po uruchomieniu aplikacji Bramka KSeF pojawi się okno podstawowe podobne do tego poniżej:



Okno to składa się z następujących elementów:

- *Pasek menu* - zawierający dostęp do wszystkich funkcji oferowanych przez program;
- *Przyciski* - zapewniające dostęp do najczęściej wykorzystywanych funkcji;
- *Okno logów* - prezentujące logi (dziennik zdarzeń) związane z działaniem aplikacji.

Developer ma możliwość modyfikacji tego okna, np. ukrywając wszystkie dostępne tu przyciski. Jeśli zatem przygotowuje własną aplikację, która będzie się komunikowała z Bramką KSeF poprzez jeden z dostępnych modeli, oraz planuje dołączyć Bramkę KSeF

do swojej wersji instalacyjnej, okno podstawowe programu może maksymalnie uprościć.

KONFIGURACJA PROGRAMU

Pierwszą czynnością jaką należy wykonać po zainstalowaniu i uruchomieniu aplikacji Bramka KSeF jest jej konfiguracja. W tym celu należy z menu **Start** wybrać **Ustawienia**. Okno konfiguracji składa się z dwóch zakładek: **KSeF** oraz **Monitorowanie**.

ZAKŁADKA KSEF

W zakładce **KSeF** możemy konfigurować integrację z usługą KSeF (Krajowy System e-Faktur). Poszczególne opcje mają następujące znaczenie:

- *Środowisko* - możliwość wyboru środowiska, w którym pracujemy. Jeśli chcemy przetestować integracje z KSeF możemy wybrać w tym miejscu Ś*rodowisko testowe*.
- *Włącz obsługę Krajowego Systemu e-Faktur (KSeF)* - włączenie integracji;
- *NIP Firmy* - NIP firmy, w kontekście której działa integracja;
- *Token* - token autoryzacyjny do usługi KSeF;
- *Import faktur* - w tej sekcji możemy określać, czy system ma również automatycznie pobierać wystawione nam faktury, a jeśli tak - od jakiej daty.

ZAKŁADKA MONITOROWANIE

Zakładka **Monitorowanie** zawiera podstawowe ustawienia programu. Ich znaczenie jest następujące:

- *Uruchamiaj program wraz ze startem systemu* - zaznaczenie tej opcji spowoduje, że aplikacja Bramka KSeF będzie automatycznie uruchamiana w momencie startu systemu Windows;
- *Minimalizuj przy zamykaniu* - zaznaczenie tej opcji spowoduje, że klikając ikonkę X w celu zamknięcia programu sprawimy, że zostanie on jedynie ukryty w zasobniku systemowym nadal będąc w pełni funkcjonalnym;

- *Serwer TCP/IP | Telnet | WebSocket* - zaznaczenie tej opcji spowoduje uruchomienie serwera TCP/IP. Dzięki temu inne aplikacje będą mogły połączyć się z Bramką KSeF poprzez owy protokół. Z obsługą serwera TCP/IP wiążą się dwie opcje: *Port* - określa port, na którym pracuje program oraz *Hasło* - określa hasło bezpieczeństwa.

  Po określeniu hasła bezpieczeństwa tylko aplikacje znające to hasło będą mogły podłączyć się do serwera.
- *Serwer HTTP / HTTPS* - zaznaczenie tej opcji spowoduje uruchomienie serwera HTTP. Dzięki temu będzie możliwe komunikowanie się z Bramką KSeF poprzez odpowiednie wywołania URL. Podobnie jak w przypadku serwera TCP/IP - należy określić port na jakim serwer będzie uruchomiony oraz hasło komunikacji.
- *Folder lokalny* - zaznaczenie opcji powoduje, że Bramka KSeF obserwuje zawartość folderu zdefiniowanego poniżej tej opcji, w poszukiwaniu zadań do wykonania. Poszukiwane są pliki z rozszerzeniem `.in` zawierający różne komendy oraz *.xml* zawierające e-Faktury.
- *Folder FTP/SFTP* - zaznaczenie tej opcji oraz podanie poniżej danych dostępowych spowoduje, że Bramka KSeF połączy się z wybranym serwerem FTP lub SFTP i będzie na nim poszukiwał plików *.xml*.
- *BinSoft Gate* - zaznaczenie tej opcji spowoduje wygenerowanie unikalnego klucza. Następnie zlecanie e-Faktur do wysłania może się odbywać poprzez API dostarczone przez bramkę BinSoft Gate. W API tym wystarczy posługiwać się tym kluczem przy zlecaniu faktur do wysłania do KSeF czy też uzyskiwaniu informacji o statusie danego dokumentu.
- *Klient WebSocket* - zaznaczenie tej opcji spowoduje, że Bramka KSeF będzie próbował połączyć się z wybranym serwerem WebSocket.

UWAGA! *Jeśli serwer TCP/IP i/lub HTTP w aplikacji Bramka KSeF jest uruchomiony, a mimo to zewnętrzne aplikacje nie chcą się z nim połączyć, należy sprawdzić czy port na którym pracuje jest odblokowany na firewall-u. Informacje o tym jak tego dokonać należy szukać w systemie pomocy Windows lub u administratora systemu.*

KOMUNIKACJA Z BRAMKA KSEF

Dostępnych jest kilka trybów współpracy zewnętrznej aplikacji z Bramką KSeF.

Nawiązanie połączenia poprzez TCP/IP (Telnet, WebSocket), odpytywanie serwera HTTP, skorzystanie z współdzielonego folderu (lokalnego lub folderu na serwerze FTP/SFTP) lub łączenie się z wybranym serwerem WebSocket.

KOMUNIKACJA POPRZEZ TCP/IP

Bramka KSeF uruchamia automatycznie **serwer TCP/IP**, do którego można nawiązać połączenie z dowolnej innej aplikacji. Domyślnie serwer nasłuchuje na porcie 7361, aczkolwiek port może zostać zmieniony w ustawieniach programu.

Zaimplementowany w Bramkę KSeF serwer TCP/IP obsługuje kilka pod-protokołów. Są to:

- protokół **Bramka KSeF** - opisany poniżej
- protokół **Telnet** (dzięki temu można połączyć się z Bramką KSeF z dowolnego klienta Telnet, np. Putty)
- protokół **WebSocket** (ws + wss)

PROTOKÓŁ BRAMKA KSEF

Serwer obsługuje komendy różnego typu. Znacznik typu to jednobajtowy kod wysyłany zawsze na początku komendy. W chwili obecnej programista może skorzystać tylko z typu podstawowego - tekstowego.

- **Typ tekstowy:** `[01]Ciąg znaków[13][10]`

W typie podstawowym, programista wysyła bajt o wartości ASCII `#01`, a następnie komendę w postaci ciągu znaków ASCII, zakodowanych w UTF8 i zakończonym bajtami `#13`, `#10`.

Odpowiedź serwera także może być różnego typu. Typ określony jest pierwszym bajtem odsyłanym przez serwer. Jeśli zatem bajt odpowiedzi będzie miał wartość `#01`, należy oczekiwać ciągu znaków, zakodowanego w UTF8, aż do napotkania bajtów `#13`, `#10`.

W dalszej części podręcznika będą omówione wszystkie komendy obsługiwane przez Bramkę KSeF. Są one bowiem wspólne w tej metodzie wymiany informacji i plikach `.in`, które Bramka KSeF może przetwarzać automatycznie.

Przykład. Chcąc uzyskać listę dostępnych portów szeregowych, należy wydać komendę `COMLIST`. Pełny ciąg znaków wysyłanych do serwera powinien mieć zatem postać:

```
[01]COMLIST[13][10]
```

Jeśli w ustawieniach programu użytkownik ustawi hasło bezpieczeństwa, wówczas pierwszą czynnością po nawiązaniu połączenia z serwerem powinno być "zalogowanie się" za pomocą komendy `PASSWORD`. Bez procesu uwierzytelniania większość komend nie będzie dostępna.

UWAGA! *Każda komenda wysyłana do Bramki KSeF i zwracana w odpowiedzi z Bramki KSeF zakończona jest dwuznakiem o kodach ASCII 13, 10. Oznacza to, że wewnątrz komendy nie można używać tych znaków, gdyż może to zaburzyć komunikację. Dlatego jeśli do jakiejś komendy chcemy przekazać jako parametr te symbole, należy w ich miejsce użyć ciągu znaków: ^13^ (dla oznaczenia kodu ASCII 13) i ^10^ (dla oznaczenia kodu ASCII 10).*

TELNET

Bramka KSeF obsługuje w ograniczonym zakresie komunikację wg. protokołu *Telnet*. Oznacza to, że można się z nim połączyć dowolną aplikacją obsługującą ten protokół i komunikować się prostymi komendami. Przy tym protokole przed komendami nie ma potrzeby wysyłania bajtu `#01`, lecz od razu wysyłamy ciąg znaków zakończony `#13#10`. Bramka KSeF również nie odpowiada bajtem początkowym `#01`, lecz ciągiem znaków.

Dzięki powyższej funkcji w prosty sposób można przetestować działanie aplikacji Bramka KSeF jak również z użyciem typowych narzędzi (np. Putty) się z nim połączyć.

WEBSOCKET

Bramka KSeF obsługuje także protokół WebSocket. Po nawiązaniu połączenia na wybranym porcie TCP/IP Bramka KSeF rozpozna, że nadawca komunikuje się według tego protokołu i odpowiednio przełączy swoją obsługę. Obsługiwany jest również WebSocket poprzez SSL (wss). Wówczas należy łączyć się na port TCP/IP + 1. Domyślnie Bramka KSeF używa certyfikatu samo-podpisanego, dlatego by takie połączenie zostało nawiązane może być konieczne dodanie tego certyfikatu do listy zaufanych.

Oto przykład prostej strony demonstrującej komunikację poprzez WebSocket.

```
<!DOCTYPE HTML>
<html>
   <head>
      <script type = "text/javascript">
      var ws = false;
      function show(s) {
        txtArea = document.getElementById("mm");
        txtArea.value += s + '\r\n';
      }

      function send(cmd) {
        if (!ws) { show('Brak połączenia!'); return; }
        show('<'+cmd);
        ws.binaryType = "blob";
```

```
        ws.send(cmd);
      }

      function sendClick() {
        cmd = document.getElementById('cmd').value;
        send(cmd);
        document.getElementById('cmd').value = '';
      }

      function sendFile() {
        if (!ws) { show('Brak połączenia!'); return; }
        var file = document.getElementById('filename').files[0];
        var reader = new FileReader();
        var rawData = new ArrayBuffer();

        reader.loadend = function() {
        }

        reader.onload = function(e) {
           rawData = e.target.result;
           ws.binaryType = "arraybuffer";
           ws.send(rawData);
        }
        reader.readAsArrayBuffer(file);
     }

     function WebSocketStart() {

        if ("WebSocket" in window) {
            show("Łączenie...");
            ws = new WebSocket("ws://localhost:7363/");
            ws.onopen = function() {
              show("Połączono!");
```

```
              send('HELLO')
            }
            ws.onmessage = function (evt) {
               var received_msg = evt.data;
               show('>' + received_msg);
            }
            ws.onclose = function() {
               show("Połączenie zamknięte!");
            }
            ws.onerror = function(err) {
               show("Błąd!");
            }
         } else {
           show("Twoja przeglądarka nie obsługuje WebSocket!");
         }
      }
      </script>
   </head>
   <body>
      <a href = "javascript:WebSocketStart()">Połącz z serwerem</a><hr />
   <input type="text" name="cmd" id="cmd" /> <a href =
"javascript:sendClick()">Wyślij</a><hr />
       <h2>Upload plików</h2>
       <input type="file" id="filename" />
       <a href="javascript:sendFile()">Wyślij plik</a>
       <hr />
       <textarea name="mm" id="mm" cols="100" rows="25"></textarea>
   </body>
</html>
```

SQL

Bramka KSeF prowadzi wewnętrzną bazę danych, w której przechowuje informacje o przetwarzanych dokumentach, UPO itp. Aplikacja pozwala na wydawanie dowolnych komend SQL w ramach połączenia TCP/IP i w ten sposób dowolne pobieranie i operowanie na tych danych. Wystarczy aby komenda wysłana do Bramki KSeF rozpoczynała się od `SELECT`, `UPDATE`, `DELETE`, `INSERT`, `ALTER` lub `TABLES` - a będzie ona kierowana do silnika SQL.

KOMUNIKACJA POPRZEZ SERWER HTTP/HTTPS

Bramka KSeF może uruchomić własny serwer HTTP na wybranym porcie (domyślnie 8001). Aby sprawdzić, czy serwer działa poprawnie można spróbować wejść z przeglądarki użytkownika pod adres http://localhost:8001. Powinna pojawić się strona informacyjna, że serwer działa poprawnie.

Chcąc korzystać z tej komunikacji, wystarczy wywoływać podany adres URL przekazując metodą GET lub POST dodatkowe informacje. Niezbędne parametry to:

- `cmd=KOMENDA`
- `password=HASŁO`

W zależności od komendy, mogą być wymagane dodatkowe parametry wywołania. Serwer odpowiada danymi zakodowanymi w formacie JSON.

Jeśli wystąpi błąd, serwer odpowie `{"error":"Komunikat"}`

Serwer HTTP programu Bramka KSeF przesyła nagłówek `Access-Control-Allow-Origin:*` dzięki czemu wywołanie tego adresu powinno być możliwe z innych adresów URL - w tym również z sieci.

Należy zwrócić uwagę, że jeśli strona WWW, z poziomu której będziemy próbować połączyć się do serwera Bramki KSeF wykorzystuje połączenie szyfrowane SSL (protokół HTTPS) - wywołanie adresu http://localhost będzie zapewne zablokowane przez przeglądarkę. Wynika to z mechanizmów bezpieczeństwa, które blokują odwołania do elementów w Sieci poprzez połączenia „niebezpieczne" (niezaszyfrowane) ze strony wykorzystującej „bezpieczne" (szyfrowane) połączenie. Aby powyższe ominąć, należy również uruchomić szyfrowane połączenia w Bramce KSeF. Jest to możliwe, potrzebujemy jednak do tego celu tzw. certyfikat SSL.

Jeśli posiadamy odpowiedni certyfikat, zapisujemy go w pliku o nazwie `cert.pem` i umieszczamy w folderze, razem z plikiem wykonywalnym Bramka KSeF. Następnie ponownie uruchamiamy aplikację. Kiedy Bramka KSeF wykryje powyższy plik w swoim katalogu, uruchomi nasłuchiwanie również na porcie 443, gdzie wykorzystywać będzie połączenia szyfrowane. Obsługiwała będzie także automatyczne rozpoznawanie połączeń, więc jeśli spróbujemy wejść w przeglądarce na adres https://localhost - Bramka KSeF powinna się automatycznie zgłosić.

Jeśli nie posiadamy odpowiedniego certyfikatu, możemy go sobie wygenerować samodzielnie. Będzie nam w tym celu potrzebne **OpenSSL**. Po jego pobraniu i zainstalowaniu wydajemy komendę w konsoli:

```
openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout
cert.pem -out cert.pem
```

i postępuje zgodnie z instrukcjami, które się ukażą. Powstanie plik `cert.pem`, który umieszczamy w folderze Bramki KSeF jak wspomniano wcześniej.

UWAGA! *Domyślnie Bramka KSeF posiada wgrany samo-podpisany certyfikat SSL właśnie dla localhost. W folderze głównym programu znajdziemy także plik RootCA.crt z certyfikatem „wydawcy" certyfikatu cert.pem. Możemy dodać ten certyfikat w swojej przeglądarce jako zaufany, wówczas wszystkie połączenia z Bramką KSeF będą nawiązywane poprawnie.*

Należy pamiętać, że powyżej wygenerowany certyfikat wystarczy do obsługi połączeń szyfrowanych SSL. Nie jest on jednak certyfikatem zaufanym, dlatego przeglądarki będą ostrzegały o takich połączeniach. Dodając jednak wyjątek dla `localhost` (czy 127.0.0.1) lub dodając certyfikat wydawcy jako zaufany, połączenia będą działać i będzie można z nich korzystać również z poziomu stron WWW wykorzystujących połączenia SSL zaufane.

UWAGA! *Istnieje możliwość zakupienia certyfikatu zaufanego dla danej domeny np. mojlocalsklep.pl. Następnie w pliku* ***\|system32\|drivers\|etc\|hosts*** *w systemie Windows określenie by domena ta, mojlocalsklep.pl wskazywała właśnie na localhost. Dzięki takiemu postępowaniu certyfikat będzie widoczny jako zaufany.*

Dostępne komendy przy połączeniach HTTP:

parsexml - Przetworzenie pliku XML

```
cmd=parsexml
password=hasło
data=XMLDATA
```

Powoduje przetworzenie przesłanego pliku XML. Plik powinien mieć format Faktura FA (1) lub Faktura FA (2).

Plik XML zostaje zapamiętany w wewnętrznej bazie danych i wywołanie to zwróci numer ID nadany temu wpisowi.

parsein - Przetworzenie pliku IN

```
cmd=parsein
password=hasło
data=INDATA
```

Powoduje przetworzenie pliku IN. Analogicznie jak powyżej, plik powinien mieć format opisany w dalszej części dokumentacji. Wywołanie zwróci ID nadane temu dokumentowi w wewnętrznej bazie danych.

gethistory - Pobranie historii

```
cmd=gethistory
password=hasło
type=files|ksef
[limit=po ile rekordów]
[start=od którego rekordu]
[id=ID]
[iddoc=ID]
```

Wywołanie powoduje zwrócenie listy rekordów lub pojedynczego rekordu.

Gdy `type=files`, zwrócona zostanie informacja o przesłanych plikach (XML/IN), gdy `type=ksef` - informacja o przetworzonych e-Fakturach KSeF itp. Jeśli podamy parametr `id=XX` - wówczas zwrócony zostanie jeden rekord z historii plików lub dokumentów. Jeśli podamy parametr `iddoc=ID` (dla `type=ksef`) - zwrócona zostanie lista dokumentów powiązanych z danym ID pliku. Poprzez parametry `start` i `limit` można listować rekordy. Wywołanie zwraca dane w kolejności od najnowszego.

#KOMENDA - Dowolna komenda

Umieszczając w zmiennej `cmd` instrukcję zaczynającą się od `#` wywołamy dowolną komendę opisaną w dalszej części podręcznika.

KOMUNIKACJA POPRZEZ WSPÓŁDZIELONY FOLDER

Kolejna z metod współpracy zewnętrznych programów z Bramką KSeF - to współdzielony folder. W ustawieniach programu można wskazać folder, który będzie "nasłuchiwany". Bramka KSeF będzie szukała w tym folderze plików z rozszerzeniem: `.in` lub `.xml`. Dotyczy to również serwera FTP, jeśli ustawione jest na nim nasłuchiwanie.

Po odnalezieniu takiego pliku, automatycznie zmieniane jest mu rozszerzenie na `.pr` (co oznacza, że plik został przetworzony) - a następnie wykonywane są komendy umieszczone w tym pliku. Na koniec tworzony jest plik z rozszerzeniem `.out` zawierający wynik działania przekazanych komend (dotyczy to plików `.in`).

Programista może zatem utworzyć we współdzielonym folderze plik tekstowy z rozszerzeniem `.in`, w którym zapisze instrukcje jakie ma wykonać Bramka KSeF. Następnie aplikacja programisty może oczekiwać na plik o tej samej nazwie, lecz rozszerzeniem `.out`. Kiedy się pojawi, może odczytać jego zawartość i na tej podstawie uzyskać informacje o wyniku wykonania przekazanych komend.

Wszystkie komendy w pliku tekstowym powinny być zapisywane w oddzielnych liniach. Plik powinien być zakodowany w ANSI (Windows 1250) lub UTF-8.

Ostatnia linia pliku (`.in`) powinna zawierać komendę `EXECUTE`. Jeśli w pliku nie będzie tej komendy, Bramka KSeF w ogóle nie będzie przetwarzała takiego pliku. (Jest to dodatkowe zabezpieczenie przed próbą przetworzenia pliku, zanim nie zostanie on do końca zapisany).

BINSOFT GATE

BinSoft Gate to kolejna metoda integracji z programem Bramka KSeF. Wykorzystuje pośrednictwo serwerów firmy BinSoft (lub partnera) w komunikacji pomiędzy aplikacji użytkownika a Bramką KSeF.

W programie Bramka KSeF wybierając z menu **Start** -> **Ustawienia**, odnajdziemy w zakładce **Monitorowanie** sekcję **BinSoft Gate**. Aby włączyć tę metodę komunikacji wystarczy zaznaczyć *Korzystaj z usługi BinSoft Gate*. W polu *Klucz* wygenerowany zostanie unikalny klucz.

Następnie aby zlecić przetworzone dokumentu wystarczy skorzystać z dostarczonego przez BinSoft bardzo prostego API.

Dane do API przekazywane są metodą GET lub POST zależnie od funkcji. Zwracane dane są zawsze zakodowane w JSON.

W odpowiedzi powinniśmy otrzymać pole `success` o wartości `true` - w przypadku powodzenia oraz dodatkowe pola zależnie od wybranej funkcji. Gdy wystąpi błąd, zamiast pola `success` otrzymamy pole `error` z opisem błędu i `errorCode` z kodem błędu.

ZLECANIE PRZETWORZENIA DOKUMENTU

Aby zlecić przetworzenie dokumentu należy wywołać adres:

https://api.bramka-ksef.pl/insert?key=KLUCZ

i metodą GET lub POST przekazać jeden parametr **data**. Umieszczamy w nim plik XML z e-Fakturą który chcemy przetworzyć. Faktura może być podana wprost, może być także zakodowana w Base64.

W odpowiedzi na to wywołanie otrzymamy w polu `id` nadany danemu dokumentowi numer ID.

To wszystko!

SPRAWDZENIE STATUSU DOKUMENTU

Chcąc sprawdzić status dokumentu należy wywołać adres:

https://api.bramka-ksef.pl/check?key=KLUCZ&id=ID

W parametrach podajemy klucz danej instancji Bramki KSeF oraz ID dokumentu, jaki otrzymaliśmy przy jego zlecaniu. W odpowiedzi otrzymamy odpowiedź pola: `id` - id danego wpisu, `ptype` - typ wpisu oraz zestaw dodatkowych pól, zależnych od tego z jakim dokumentem mamy do czynienia.

Dla `ptype` = FA (e-Faktura) otrzymamy pola:

- `status` - status dokumentu
- `nodoc` - numer dokumentu
- `nida` - numer KSeF
- `nidb` - numer referencyjny e-Faktury
- `upo` - UPO
- `ndate` - data przyjęcia do KSeF
- `nerrorcode` - kod błędu
- `nerrorstr` - opis błędu

Statusy mają następujące znaczenie:

- **0** - oczekuje na wysłanie do bsxPrinter
- **1** - odebrany przez bsxPrinter
- **2** - wysłano do KSeF
- **-3** - Odrzucony przez KSeF
- **4** - Trwa weryfikacja po stronie KSeF
- **5** - Potwierdzony przez KSeF
- **6** - Dostępne jest UPO

WŁASNE SERWERY DLA BINSOFT GATE

Domyślnie usługa BinSoft Gate korzysta z serwerów firmy BinSoft. Jednakże istnieje możliwość współpracy z partnerami, by ich indywidualne wersje programu Bramka KSeF korzystały z ich serwerów na potrzeby tej usługi. Dzięki temu wszystkie informacje przekazane do Bramki KSeF nie będą trafiały na serwery BinSoft lecz na serwery Partnerów.

Aby skorzystać z powyższej możliwości należy skontaktować się z nami w celu ustalenia szczegółów.

BinSoft udostępni kod źródłowy w języku PHP skryptu obsługującego BinSoft Gate i pomoże w przygotowaniu takiego rozwiązania.

KLIENT WEBSOCKET

Bramka KSeF umożliwia nawiązanie połączenia z dowolnym serwerem WebSocket. Wystarczy w jego konfiguracji wprowadzić odpowiedni adres URL. W adresie tym można zdefiniować port, na który ma być nawiązane połączenie po znaku dwukropka. Adres można poprzedzić również identyfikatorem protokołu `ws://` lub `wss://` (dla połączeń szyfrowanych), np.: `wss://www.adres-serwera-wss.pl/api:7654`

Jeśli w ustawieniach określimy nazwę użytkownika i/lub hasło, wówczas Bramka KSeF po nawiązaniu połączenia z serwerem automatycznie wyślę do niego linię:

```
#LOGIN#użytkownik#hasło
```

KOMENDY BRAMKI KSEF

System Bramka KSeF zawiera szereg komend, które możne wykonać - bądź to za pośrednictwem połączenia TCP/IP lub poprzez plik tekstowy umieszczony we współdzielonym katalogu (lub na serwerze FTP). Wszystkie komendy mogą być poprzedzone znakiem #. Nie jest to konieczne i występuje tylko dla zachowania kompatybilności ze starszymi wersjami programu. Jeśli zawierają jakieś dodatkowe parametry, te powinny być przekazywane po znaku #. Oto kilka przykładów:

- `#KOMENDA` - bez parametru
- `#KOMENDA#Parametr1` - jeden parametr
- `#KOMENDA#Parametr1#Parametr2` - dwa parametry

W odpowiedzi na każdą komendę Bramka KSeF odpowie ciągiem znaków. Będzie on rozpoczynał się od:

- `+OK[#...]` - jeśli komenda została wykonana poprawnie; po znaku `#` mogą być dodatkowe informacje związane z wykonaną komendą;
- `+ERR[#KOD BŁĘDU]#Opis błędu` - jeśli wystąpi błąd;

UWAGA! *Umieszone informacje w nawiasach kwadratowych oznaczają, że dane informacje (np. parametry) występują opcjonalnie.*

UWAGA! *Kolejne parametry oddzielone są znakiem #. Oznacza to, że chcąc podać kolejny parametr (np. trzeci), należy podać wszystkie poprzednie. Nie możemy również użyć symbolu # jako wartość jakiegoś parametru. Nie możemy więc w nazwie produktu wyświetlić tego symbolu.*

DOSTĘPNE KOMENDY PODSTAWOWE

- `#HELLO` - Bramka KSeF przedstawi się swoją nazwą oraz numerem wersji;
- `#DATE` - zwraca datę na serwerze Bramki KSeF;
- `#TIME` - zwraca czas na serwerze Bramki KSeF;
- `#DATETIME` - zwraca datę i czas na serwerze Bramki KSeF;
- `#PASSWORD#hasło` - autoryzacja użytkownika;
- `#CLOSE` - wyłączenie programu Bramka KSeF;
- `#CONNECTIONS` - zwraca listę wszystkich podłączonych do serwera połączeń;
- `#SERVERS` - zwraca status wszystkich serwerów i monitorów;
- `#DATABASE[#Value]` - zwraca informacje o aktualnej bazie danych, z której korzysta aplikacja. Jeśli przekazano wartość `Value` - ustawia nową bazę danych; Jako `Value` można wpisać: `default` - by użyta była domyślna baza danych lub ciąg reprezentujący dostęp do bazy danych np.: `MySQL#localhost#3306#root#bsx#bsxprinter` lub `MSSQL#192.168.188.25\SQLEXPRESS#1433#sa#sa#bsxprinter`
- `#LICENSE[#Name#E-mail#Phone#Key]` - zwraca informacje o aktualnej licencji lub pozwala na zarejestrowanie programu;
- `#LOGMONITOR#Value[#Name]` - ustawienie monitora logów; `Value` przyjmuje wartość 1 / 0 (włączony/wyłączony). Parametr `Name` gdy przyjmie wartość `low` oznacza logi niskopoziomowe, w przeciwnym wypadku logi standardowe;
- `#GETACTDOCUMENTS` - licznik aktualnie wydrukowanych dokumentów;
- `#GETLIMITDOCUMENTS` - pobranie limitu dokumentów;
- `#GETAVAILABLEDOCUMENTS` - pobranie liczby możliwych do wydrukowania jeszcze dokumentów;

OBSŁUGA KONFIGURACJI

- `#CONFIGFILE` - zwraca ścieżkę do pliku konfiguracyjnego;
- `#RELOADCONFIG` - przeładowuje plik konfiguracyjny;
- `#CONFIG` - zwraca zawartość pliku konfiguracyjnego;
- `#GET#Section#Name` - zwraca wartość z pliku konfiguracyjnego;
- `#SET#Section#Name#Value` - modyfikuje wpis w pliku konfiguracyjnym;

OBSŁUGA BAZY DANYCH I PLIKÓW XML

- `#XMLDATA#<xml>` zlecenie przetworzenia danych zapisanych w XML; w ten sposób można przekazać do Bramki KSeF cały dokument opisany w XML. Należy pamiętać by w kodzie XML nie używać znaku przejścia do nowej linii (#13, #10), gdyż te znaki kończą komendę. Jeśli bardzo nam na nich zależy, należy je zakodować jako ciągi: ^13^ i ^10^. Przekazując dane w XML Bramka KSeF zapisuje je do wewnętrznej bazy danych tzw. *bazy plików*, a następnie przetwarza ten wpis asynchronicznie. W efekcie przetworzenia w drugiej bazie (bazie *dokumentów*) tworzy zlecenie przetworzenia odpowiednich dokumentów itp.

  W odpowiedzi na poprawne wykonanie tej komendy zwrócone zostaną dane w postaci: `+OK#ID_PLIKU#ID_DOKUMENTU1;ID_DOKUMENTU2….`
  Zatem zwrócony zostanie ID pliku, który utworzył się w bazie *plików* i numery ID dokumentów utworzonych w bazie *dokumentów*, oddzielone średnikami.
- `#FILES` - zwraca listę plików z wewnętrznej bazy plików; Komenda zwraca maks. 100 ostatnich wpisów. Struktura zwracanych danych jest następująca: `+OK#ID_PLIKU|STATUS|KOD_BŁĘDU;.…`

- `#FILE#id` - zwraca opis pliku w bazie *plików*, o podanym numerze ID. Struktura zwracanych wartości: `+OK#ID_PLIKU|STATUS|KOD_BŁĘDU|OPIS_BŁĘDU`.

SQL

- `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `ALTER` - komendy SQL odnoszące się do wewnętrznej bazy danych; Pozwalają na wydobywanie dowolnych informacji z tej bazy, jak również dokonywanie zmian w bazie;
- `SHOW TABLES` - wyświetla listę wbudowanych tabel;

INNE KOMENDY

- `#RUNLOOP` - uruchamia przetwarzanie kolejek dla wszystkich integracji (XML, FTP, lokalny folder itp.);
- `#RUNPOOL` - uruchamia przetworzenie wewnętrznej kolejki dokumentów do przetworzenia;
- `#RUNFISCALPOOL` - uruchomienie przetworzenia wewnętrznej kolejki plików do przetworzenia;
- `#RUNONTIMER` - uruchomienie we wszystkich rozszerzeniach i wtyczkach funkcji `onTimer();`

STRUKTURA TABEL

Aplikacja Bramka KSeF, kiedy się uruchamia łączy się z bazą danych, w której przechowuje lokalne informacje o wysłany/odebranych dokumentach, UPO itp. Domyślnie tworzy w tym celu bazę SQLite, która znajduje się w jej folderze roboczym. Istnieje możliwość zmiany, aby bazą danych była dowolna baza wspierana przez aplikację. Obsługiwane są serwery: MySQL, MariaDB, MS SQL, Firebird, PostgreSQL, Interbase i właśnie SQLite. Aby zmienić użytą bazę danych, należy w pliku konfiguracyjnym `config.ini`, w sekcji `[Settings]` dodać wpis `DatabaseURL`. Wpis powinien mieć format:

```
RODZAJ_SERWERA#adres#port#login#hasło#baza danych
```

Np.:

```
DatabaseURL=MSSQL#192.168.188.25\SQLEXPRESS#1433#sa#pswd#bramkaksef
```

Bramka KSeF po połączeniu się do bazy danych tworzy w nim niezbędną strukturę tabel. Tabele mają przedrostek `bsx_`. W kontekście użytkownika najważniejsze są tabele:

- `bsx_ksef` - informacje o dokumentach;
- `bsx_ksefupo` - informacje o UPO;

Poszczególne kolumny w tabeli `bsx_ksef` mają następujące znaczenie:

- `ptype` - typ dokumentu;
- `add_time` - data dodania wpisu do bazy danych;
- `pdate_issue` - data wystawienia dokumentu;
- `nnodoc` - numer dokumentu;
- `ksefstatus` - status dokumentu;
- `ksefno` - numer KSeF dokumentu;
- `ksefdate` - data wysłania/odbioru z KSeF;
- `ksefxml` - kod XML dokumentu;
- `kseferror` - opis błędu (jeśli wystąpił);

- `ksefref` - numer referencyjny sesji;
- `kseferef` - numer referencyjny dokumentu;
- `pxmlsource` - identyfikator dokumentu w bramce BinSoft Gate;
- `psend` - znacznik, czy dokument został wysłany do bramki BinSoft Gate;
- `md5` - suma kontrolna dokumentu XML;
- `ksefstart` - data rozpoczęcia przetwarzania;

Statusy (`ksefstatus`) mają następujące znaczenie:

- 0 - wprowadzony;
- 1 - w kolejce do przetworzenia;
- 10 - w trakcie przetwarzania;
- 2 - wysłano do MF;
- 3 - odrzucono; w `kseferror` znajdzie się opis błędu;
- 4 - trwa weryfikacja;
- 5 - potwierdzono;
- 6 - pobrano z KSeF;

Algorytm w aplikacji przebiega następująco:

1. Dodając dokument do bramki przeciągając dokument do aplikacji, używając w aplikacji przycisku „Wyślij dokumenty”, poprzez metodę monitorowania folderu, bramkę BinSoft Gate, komunikacją TCP/IP - zawsze tworzy się wpis w tabeli `bsx_ksef` ze statusem `ksefstatus=1` (do wysyłki). Jeśli wpis pochodzi z bramki BinSoft Gate wypełniony będzie `pxmlsource` wskazujący ID tego dokumentu; W pozostałych przypadkach będzie on równy 0. W polu `ksefxml` umieszczony będzie cały dokument XML, a w polu `md5` - jest suma kontrolna; Uzupełniane są także pola `ndate_issue` i `nnodoc` - pobrane z tego dokumentu;
2. Bramka KSeF cyklicznie sprawdza, czy są w niej rekordy ze statusem `=1`. Jeśli tak rozpoczyna ich przetwarzania. Ustawia wówczas status `ksefstatus=10` (w trakcie

przetwarzania) zapamiętuje datę w zmiennej `ksefstart`. Następnie wysyła dokument do MF. Po wysłaniu zmienia status na `ksefstatus=2`. Po około 20 sekundach Bramka KSeF sprawdzi status dokumentu. Jeśli będzie już on pozytywny ustawi status na `ksefstatus=5`. W przypadku błędu ustawi status na `ksefstatus=3`, a w zmiennej `kseferror` umieści informacje o błędzie. Może się także zdarzyć, że usługa KSeF nie zdąży zweryfikować dokumenty w ciągu 20 sekund. Wówczas zmieni status na `ksefstatus=4` (w trakcie weryfikacji) i ponowi sprawdzenie statusu przy kolejnym cyklicznym sprawdzaniu kolejki;

3. W przypadku pozytywnego przesłania dokumentu wypełnione będą pola `ksefdate`, `ksefno`, `ksefref` i `kseferef`. W polu `kseferef` znajduje się numer referencyjny wysłanego dokumentu, natomiast w `ksefref` numer referencyjny sesji.

4. Przy kolejnej analizie kolejki, dla dokumentów które mają status pozytywny pobierany jest dokument UPO i zapamiętywany w tabeli `bsx_ksefupo`. Kluczem, który wiąże te UPO z dokumentami jest pole `ksefref`. Oznacza to, że wszystkie dokumenty przesłane w jednej sesji mają zbiorczy UPO.

5. Jeśli dokument został dodany do wysłania poprzez bramkę BinSoft Gate Bramka KSeF wysyła do bramki kolejne statusy przetwarzania. Wykorzystuje w tym celu pole `pxmlsource`. W polu `psend` oznacza wartością `=1` znacznik, że status końcowy dokumentu został do bramki wysłany - by nie był ponawiany przy kolejnym przetwarzaniu.

BRAMKA KSEF JAKO USŁUGA WINDOWS

Istnieje możliwość zainstalowania aplikacji Bramka KSeF jako usługa Windows. Aby tego dokonać należy uruchomić wiersz poleceń Windows (koniecznie z uprawnieniami Administratora), a następnie przejść do folderu, gdzie został zainstalowany program (domyślnie `c:\Program Files (x86)\Bramka KSeF`).

Należy wydać polecenie:

```
bsxServer.exe /install
```

Można też uruchomić instalację „po ciuchu" - wówczas:

```
bsxServer.exe /install /silent
```

Po zainstalowaniu usługi możemy ją uruchomić, wstrzymać, zatrzymać - korzystając z narzędzia „Usługi" w systemie Windows.

Chcąc odinstalować usługę wydajemy z wiersza poleceń komendę:

```
bsxServer.exe /uninstall
```

lub „po cichu"

```
bsxServer.exe /uninstall /silent
```

W Usługach Windows program Bramka KSeF będzie widoczny jako „`bsxServer`".

OBSŁUGA PROGRAMU

Usługi nie posiadają interfejsu, dlatego nie można ich w prosty sposób konfigurować i obsługiwać. W programie Bramka KSeF zawsze uruchamiany jest serwer lokalny TCP/IP działający domyślnie na porcie `7361`, oraz ma skonfigurowane hasło domyślne `BinSoftBSX`. Możemy się zatem w nim połączyć poprzez terminal (np. Z użyciem programu Putty) i w ten sposób nim sterować.

PLIKI KONFIGURACYJNE

Bramka KSeF przechowuje swoje pliki konfiguracyjne w katalogu: `c:\Users\Nazwa_użytkownika\AppData\Roaming\BramkaKSeF\data`. Jednak uruchomiony jako usługa korzysta z folderu, w którym jest zainstalowany, tj. w folderze `c:\Program Files (x86)\Bramka KSeF` stworzy folder `data` i tam zacznie tworzyć swoje pliki.

W katalogu tym odnajdziemy plik `config.ini`, w którym zapisana jest cała konfiguracja aplikacji. Możemy w ten sposób ustawiać konfigurację do aplikacji.

Warto tutaj podpowiedzieć, że jeśli uruchomimy Bramkę KSeF z prawami Administratora, ona również uruchomi się wskazując konfigurację, jak przy usłudze. Zatem istnieje możliwość, abyśmy uruchomili Bramkę KSeF z prawami administratora, a następnie ją skonfigurowali według uznania i wyłączyli. Wówczas zapisze się konfiguracja i już aplikacji w wersji usługowej będzie z niej korzystała.

Uwaga! Uruchamiając Bramkę KSeF może ona ustawić opcję automatycznego startu przy uruchamianiu Windows. Warto tę opcję wyłączyć, jeśli planujemy korzystać z wersji usługowej, tak by w tym samym czasie nie działały zarówno bsxServer jak i Bramka KSeF.

TWORZENIE WTYCZEK DO BRAMKI KSEF

Bramka KSeF pozwala na tworzenie tzw. wtyczek (ang. *plugins*). Wtyczki mogą rozszerzać funkcjonalność aplikacji, dodając do niej np. obsługę sklepów on-line, integrując się z systemami CRM, CMS itp. Tworzenie wtyczek polega na przygotowaniu odpowiednich plików XML oraz PAS.

BUDOWA WTYCZKI

Wtyczki do programu Bramka KSeF to odpowiednio przygotowane pliki z rozszerzeniem `.xml` oraz `.pas`. Każda wtyczka to folder, w którym powinny znajdować się owe pliki. Folder taki powinien być umieszczony w katalogu `APP\plugins` - gdzie APP - to folder, gdzie zainstalowany jest program Bramka KSeF. Na przykład:

- `APP\plugins\Wtyczka\plugin.xml`
- `APP\plugins\Wtyczka\plugin.pas`

Bramka KSeF w momencie uruchamiania przeszukuje wszystkie podfoldery folderu `plugins` i analizuje wszystkie pliki XML, które tam odnajdzie.

Każdy plik XML powinien mieć następujący format:

```
<plugin name="NazwaFirmy.NazwaPluginu" caption="Tytuł pluginu">
    <!-- dodatkowe sekcje -->
</plugin>
```

Każdy plugin możne zawierać dowolnie wiele okien dialogowych (formularzy).
Okna te definiuje się w znaczniku `<forms>`. Każde okno opisuje się znacznikiem `<form>`. Znacznik `<form>` posiada następujące atrybuty:

- `name` - nazwa danego okna,
- `priority` - priorytet okna,
- `inherited` - nazwa okna, po którym chcemy dziedziczyć,
- `width` - szerokość okna,
- `height` - wysokość okna,

- `caption` - tytuł okna.

W atrybucie name podajemy nazwę tworzonego formularza. Bramka KSeF uzupełni tę nazwę o nazwę samego pluginu i rozdzieli symbolem kropki. Jeśli zatem plugin ma `name="NazwaFirmy.NazwaPluginu"`, a formularz ma `name="Okno"`, to pełna nazwa tego okna (którą posługujemy się w innych miejscach programu) to `NazwaFirmy.NazwaPluginu.Okno`. Jeśli nie chcemy, aby system uzupełniał nazwę okna nazwą pluginu, powinniśmy ją poprzedzić symbolem `@`, np. `@Okno`.

Wewnątrz znacznika `<form>` umieszcza się znaczniki tworzące różne elementy interfejsu. Dostępne znaczniki to:

- `<field>` - pole ukryte;
- `<edit>` - pole tekstowe,
- `<editbtn>` - pole tekstowe z widocznym przyciskiem,
- `<memo>` - wielowierszowe pole tekstowe,
- `<combobox>` - pole z listą wyboru,
- `<listbox>` - lista wyboru,
- `<trackbar>` - pasek przesuwania,
- `<radio>` - pole typu *radio*,
- `<checkbox>` - pole typu *checkbox*,
- `<progressbar>` - pasek postępu,
- `<button>` - przycisk,
- `<label>` - etykieta,
- `<html>` - etykieta obsługująca podstawowe znaczniki HTML,
- `<groupbox>` - zgrupowanie obiektów,
- `<panel>` - panel z obiektami,
- `<tabs>` - panel z zakładkami,
- `<tab>` - pojedyncza zakładka,

- i wiele innych.

Znaczniki `<groupbox>` i `<panel>` mogą w swoim wnętrzu zawierać kolejne elementy, tworząc w ten sposób hierarchę drzewa. Wewnątrz znacznika `<tabs>` (zakładki) może znaleźć się tylko znacznik `<tab>` (zakładka), a w nim dowolne inne obiekty.

Umieszczanie kolejnych znaczników opisujących interfejs powoduje, że elementy te umieszczane są jeden pod drugim. Korzystając z atrybutów: `width` i `height` - można zmieniać wymiary tych obiektów; Natomiast atrybutami `top` i `left` - można zmieniać ich domyślne położenie.

Jeśli jako wartość parametru `top` wpiszemy `-1` - wówczas obiekt znajdzie się na wysokości obiektu poprzedniego, a jego `left` ustawi się automatycznie tak by nowy element był „obok" poprzedniego. Dzięki temu możemy w prosty sposób umieszczać kilka elementów obok siebie.

Każdy z obiektów posiada dodatkowo atrybuty:

- `name` - określający nazwę danego obiektu,
- `caption` - tytuł obiektu,
- `default` - domyślna zawartość,
- `visible` - widoczność obiektu,
- `enabled` - status dostępności obiektu,
- `anchors` - uchwyty obiektu,
- `align` - sposób wyrównania obiektu;

Właściwości `anchors` (uchwyty) nadajemy wartość tekstową, która może zawierać wyrazy: `left`, `right`, `top`, `bottom`. Każda z nazw odpowiada stworzeniu uchwytu odpowiednio lewego, prawego, górnego i dolnego. Utworzenie odpowiedniego uchwytu powoduje, że dana kontrolka "utrzymuje" stale odległość do danej krawędzi obiektu, w którym się znajduje, podczas zmiany wymiarów tego obiektu.

Właściwość `align` wpływa na sposób wyrównania obiektu. Możliwe wartości to:

- `top` - wyrównanie do góry,
- `left` - wyrównanie do lewej,

- `right` - wyrównanie do prawej,
- `bottom` - wyrównanie do dołu,
- `client` - wypełnienie;

Bramka KSeF uruchamiając się automatycznie ładuje formularz o nazwie `@Main`. Domyślnie wyświetli się formularz dostarczony wraz z programem. Jeśli jednak zrobimy samodzielnie formularz o tej nazwie i wyższym priorytecie, wówczas to on zostanie załadowany.

Oto przykład:

```
<plugin name="Firma.Plugin">
 <forms>
     <form name="@Main" caption="DrukSoft for v1.0" width="600" height="380"
priority="1">
      <button caption="Kliknij mnie" onclick="clickMe" />
      <script src="skrypt.pas" />
     </form>
  </forms>
</plugin>
```

W znaczniku `<script>`, w atrybucie `src` podajemy nazwę pliku z kodem w Pascalu, który ma być powiązany z danym formularzem. W naszym przykładzie jest to plik `skrypt.pas`. Może on mieć postać jak poniżej:

```
procedure clickMe;
begin
  ShowMessage('Witaj Świecie!');
end;
```

BUDOWA INTERFEJSU W JĘZYKU HTML

Budując formularz można użyć kontrolki `<html>`. Domyślnie wypełnia ona cały dostępny obszar, gdzie się znajduje (atrybut `align=client`). Wewnątrz tej kontrolki można używać znaczników języka HTML. Przykład:

```
<plugin name="Firma.Plugin">
 <forms>
     <form name="@Main" caption="DrukSoft for v1.0" width="600" height="380"
priority="1">
      <html>
       <style> b { color: red; } </style>
       <p>Akapit <b>tekstu</b></p>
       <button onclick="pascal:clickMe">Kliknij mnie</button>
      </html>
      <script src="skrypt.pas" />
     </form>
  </forms>
```

Powyższy przykład wyświetli jeden akapit tekstu (z czerwonym słowem tekstu) i przyciskiem. Kliknięcie w przycisk wywoła funkcje `clickMe` z Pascala. W przykładzie widzimy zatem, że by wywołać dowolną funkcję z podpiętego Pascala należy jej nazwę poprzedzić słowem: `pascal:`.

Znacznik `<html>` posiada atrybut `style`, któremu jako wartość możemy podać nazwę pliku ze stylami CSS. Posiada również wszystkie inne standardowe atrybuty, np.: `visible`, `align`, `width` itp.

ELEMENTY MENU

Z poziomu wtyczki można także dodawać nowe pozycje do menu głównego aplikacji. W tym celu należy umieścić wewnątrz formularza `@Main` znacznik `<menu>`.
Kolejne elementy (pozycje) menu opisuje się znacznikiem `<item>`. Znaczniki `<item>` można w sobie zagnieżdżać tworząc w ten sposób dowolnie zagłębione menu.
Oto przykład:

```
<menu>
  <item name="mnPlugins" caption="Wtyczki">
    <item caption="Konfiguracja" onclick="oknoKonfiguracjiClick" />
  </item>
</menu>
```

Ten kod XML spowoduje, że pokaże się nowa pozycja w menu programu zatytułowana *Wtyczki*. Wewnątrz niej będzie jedna pozycja o etykiecie *Konfiguracja*. Wybranie tej pozycji spowoduje uruchomienie procedury o nazwie `oknoKonfiguracjiClick` opisanej w powiązanym pliku `.pas`.

Jak zatem widać w przykładzie, wewnątrz `<item>` można używać atrybutów:

- `name` - nazwa elementu menu;
- `caption` - tytuł elementu menu;
- `onlick` - nazwa procedury, która ma zostać uruchomiona po wybraniu danej pozycji;
- `visible` - widoczność elementu;

Wszystkie elementy menu powinny posiadać nazwy. Jeśli użyjemy nazwy już istniejącej, wówczas nie stworzymy nowej pozycji, lecz "dołączymy" się do tej istniejącej.

Po uruchomieniu aplikacji Bramka KSeF w menu już znajdują się pewne elementy.
Ich nazwy są następujące:

- `mnStart` - *Start*
- `mnHelp` - *Pomoc*

Jeśli zatem użyjemy którejś z powyższych nazw w znaczniku `<item>`, możemy wpiąć swoje elementy menu w odpowiednią gałąź. Z tego mechanizmu można też skorzystać aby np. ukryć określone elementy menu. Przykład krótkiego pluginu:

```
<plugin name="Firma.Plugin">
 <forms>
     <form name="@Main" priority="1" inherited="@Main">
      <menu>
        <item name="mnHelp" visible="false" />
      </menu>
     </form>
  </forms>
</plugin>
```

JĘZYK SKRYPTOWY - PASCAL

Za pomocą plików XML możliwe jest budowanie formularzy, określanie elementów menu, wpływanie na wygląd aplikacji. Aby do niej dodać logikę, należy utworzyć odpowiednie pliki skryptowe. Są to pliki zapisane z rozszerzeniem .pas i powiązane z danym formularzem za pomocą znacznika `<script>`. W plikach tych stosuje się język programowania bazujący na Pascalu. Nie jest to dokładna implementacja oryginalnego Pascala, lecz pewna jego odmiana. W dalszej części tej instrukcji język ten jednak będziemy nazywać po prostu "Pascalem".

ELEMENTY SKŁADOWE PASCALA BRAMKI KSEF

W języku Pascal każda instrukcja powinna być zakończona średnikiem. Skrypt rozpoczyna się słowem kluczowym `begin` i kończy słowem `end`;. Kiedy Bramka KSeF wczytuje dany skrypt automatycznie uruchamia kod umieszczony w tym miejscu.
Oto przykład:

plugin.pas

```
begin
  ShowMessage('Witaj');
end;
```

Jeśli plik ten powiążemy z formularzem poprzez znacznik `<script>`, wówczas po wyświetleniu się tego formularza pokaże się komunikat *Witaj*.

W języku Pascal wielkość liter nie ma znaczenia. W przykładzie wywołano instrukcję `ShowMessage()`. Można ją było wywołać również poprzez nazwy `showmessage()`, czy `SHOWMESSAGE()`.

W Pascalu można tworzyć procedury oraz funkcje. Procedury i funkcje to podprogramy wykonujące określone czynności. Mogą one przyjmować argumenty, wówczas przekazujemy je w nawiasach okrągłych oddzielając od siebie przecinkami.
Funkcje - w przeciwieństwie do procedur - mogą też zwracać wartości.
Zwrócenie wartości polega na przypisaniu jej do wewnętrznej zmiennej `Result`.
Przykład:

```
procedure nazwaProcedury(Argument1,Argument2);
begin
  //kod procedury
end;
function nazwaFunkcji(Argument1):ZwracanyTyp;
begin
  //kod funkcji
  Result:='Wynik';
end;
```

Jeśli funkcja/procedura nie oczekuje żadnych argumentów wówczas nie umieszczamy części w nawiasach okrągłych. Na przykład:

```
procedure pobierzDane;
begin
  //kod procedury
end;
```

Oczywiście w obrębie procedury/funkcji można deklarować zmienne lokalne. Deklaruje się je po słowie kluczowym `var`, przed słowem `begin`.

```
procedure pobierzDane;
var x,y,z : String;
    a,b,c : Integer;
begin
  //kod procedury
end;
```

W języku Pascal do przypisywania wartości służy operator przypisania, tj. `:=`. Sam znak "`=`" jest operatorem porównywania. Inne operatory porównywania to: `<`, `>`, `<=`, `>=`, `<>`. Oprócz tego dostępne są operatory logiczne: `OR`, `AND`, `NOT`.

Instrukcja warunkowa w Pascalu ma postać:

```
if warunek then instrukcja;
```

Jeśli chcemy dodać klauzulę `else`, wówczas instrukcja ta ma postać:

```
if warunek then instrukcja1 else instrukcja2;
```

Należy zwrócić uwagę, że po `instrukcja1` nie ma średnika.

Pascal udostępnia tzw. instrukcję złożoną. Instrukcja złożona to dowolny blok instrukcji objęty słowami kluczowymi `begin` i `end`. Można z niej skorzystać wszędzie tam, gdzie składnia przewiduje użycie pojedynczej instrukcji. Zatem chcąc wykonać więcej linii kodu w instrukcji warunkowej może ona wyglądać tak:

```
if warunek then
begin
  instrukcja1;
  instrukcja2;
end else
begin
  instrukcja3;
  instrukcja4;
end;
```

W Pascalu można korzystać ze zmiennych. Aby móc użyć zmienną, należy ją najpierw zadeklarować. Używa się w tym celu słowa kluczowego `var`. Zmienne deklaruje się na samym początku pliku - wówczas są to zmienne globalne - lub też przed słowem `begin` procedur i funkcji - wówczas są to zmienne lokalne. Przykład:

```
var zmiennaGlobalna;

procedure test;
var zmienaLokalna1, zmiennaLokaln2;
begin
  //instrukcje
end;
```

W Pascalu można tworzyć pętle. Najbardziej popularna jest pętla `for`. Jej składnia jest następująca:

```
for licznik:=start to koniec do instrukcja;
```

Oczywiście w miejscu instrukcji może znaleźć się instrukcja złożona. Przykład:

```
for i:=1 to 10 do
begin
   ShowMessage(i);
end;
```

Aby zrobić pętlę odliczającą w dół należy użyć składni:

```
for licznik:=start downto koniec do instrukcja;
```

Istnieją jeszcze dwie inne formy pętli: `while` oraz `repeat`. Składnia instrukcji while jest następująca:

```
while warunek do instrukcja;
```

Pętla wykonywana jest tak długo, dopóki warunek jest spełniony.

Składnia instrukcji `repeat` jest następująca:

```
repeat
  instrukcja1;
  instrukcja2;
  itd.
until warunek;
```

Pętla jest wykonywana tak długo, aż warunek będzie spełniony.

W Pascalu występuje typowanie danych. Podczas deklaracji zmiennych można podać nazwę typu, jednak jest to z reguły robione tylko dla zachowania czytelności kodu. W praktyce nie ma potrzeby tego czynienia, gdyż typ definiuje się poprzez przypisanie do zmiennej wartości. Oznacza to, że jeśli do zmiennej przypiszemy wartość liczbową - będzie ona miała typ liczbowy. Jeśli przypisze się wartość tekstową - będzie miała typ tekstowy. (Wartości tekstowe umieszczamy zawsze w apostrofach!).
Nie można tworzyć wyrażeń składających się ze zmiennych różnych typów. Niedozwolony jestem np. taki zapis:

```
x:='Jan ma '+20+' lat';
```

W Pascalu można tworzyć tablicę. Robimy to w następujący sposób:

```
var T : Array[0..10] of String;
```

Można również:

```
var T : array;
```

Korzystając z symboli `[]` można tworzyć tablice dynamicznie, np.:

```
T:=['Jan','Piotr'];
```

Uwaga! W związku z tym, że symbole `[]` służą do tworzenia tablic, nie można ich używać do tworzenia zbiorów - jak ma to miejsce w klasycznym Pascalu. W tym celu należy używać funkcji `SetOf(array)`, który z tablicy tworzy zbiór.

PRZYDATNE FUNKCJE PODSTAWOWE

Pascal oferuje szereg funkcji podstawowych, z których z pewnością będziemy korzystać bardzo często. Są to:

- `ShowMessage(napis);` - wyświetlenie napisu w oknie dialogowym;
- `IntToStr(liczba);` - zamiana liczby na napis;
- `StrToInt(napis);` - zamiana napisu na liczbę;
- `StrToIntDef(napis,def);` - zamiana napisu na liczbę; jeśli się nie powiedzie zwróci wartość domyślną def;
- `Copy(napis,od,ilość);` - skopiowanie fragmentu ciągu;
- `Delete(napis,od,ilość);` - wycięcie z ciągu znaków określonego fragmentu;
- `Pos(ciąg,napis);` - szukanie ciągu wewnątrz napisu;
- `LowerCase(napis);` - zamiana liter na małe;
- `UpperCase(napis);` - zamiana liter na wielkie;
- `Length(napis);` - zwraca długość napisu;

- `Now;` - zwraca aktualną datę i godzinę (w postaci zmiennej typu `TDateTime`);
- `DateToStr(data)` - zamienia datę na napis;
- `DateTimeToStr(data)` - zamienia datę i godzinę na napis;
- `StrToDate(napis)` - zamienia napis na datę;
- `StrToDateTime(napis)` - zamienia napis na datę i godzinę;
- `SetOf(tablica)` - zamienia tablicę na zbiór;

Oprócz funkcji podstawowych Bramka KSeF oferuje szereg własnych funkcji. Są to:

- `APP.ShowForm(sender,nazwa,default,modal);` - wyświetlenie formularza o nazwie `nazwa`; Jako `Sender` przekazujemy obiekt, z którego następuje wywołanie. Można w to miejsce podać wartość `Self`. `Default` to lista domyślnych wartości w postaci: `nazwa=wartość;nazwa=wartość` itd. Jako `modal` podajemy wartość: `_BOOL_TRUE` - jeśli okno ma być modalne lub `_BOOL_FALSE` - w przeciwnym wypadku.
- `Home.GetValue(nazwa,domyślnie);` - pobranie wartości z obiektu `nazwa`. Jeśli obiekt nie zostanie odnaleziony zwrócona zostanie wartość domyślna;
- `Home.SetValue(nazwa,wartość);` - nadanie wartości dla wybranego obiektu;
- `Home.FindFieldControl(nazwa);` - pobranie bezpośredniego uchwytu do obiektu;

Bramka KSeF udostępnia także szereg klas oferujących różne rozszerzone funkcjonalności. Są to:

- `TRegistry` - obsługa rejestru Windows;
- `TIniFiles` - obsługa plików INI;
- `TStringList` - obsługa danych tekstowych;
- `TBinDatabase` - obsługa baz danych;
- `TBSXRest` - obsługa HTTP i REST;
- `TBSXUtils` - różne przydatne funkcje;
- `TXMLDocument` - obsługa XML;

- `TJSONPair`, `TJSONObject`, `TJSONArray`, `TJSONString`, `TJSONValue` - obsługa JSON;

KILKA PRZYKŁADÓW

Poniżej przedstawionych zostało kilka przykładowych fragmentów programu, które demonstrują sposób użycia niektórych funkcji.

PRZYKŁAD 1

```
procedure pobierzDane;

var L : TStringList;

begin

  L:=TStringList.Create;

  L.Text:=TBSXRest.SGET('http://127.0.0.1/
                         script.php','login=Janusz;pass=Hasło');

  L.SaveToFile('c:\1.htm');

  L.Free;

  ShowMessage('Przetworzono');

end;


begin

  Self.CreateTimer('Timer',60000,'pobierzDane');

end;
```

W przykładzie zastosowaną metodę statyczną klasy `TBSXRest` do pobierania plików ze zdalnych serwerów. Metoda przyjmuje dwa parametry: adres URL oraz listę parametrów przekazywanych metodą POST. Funkcja zwraca plik odebrany z serwera.

W przykładzie pokazano również obsługę timerów. Funkcja `CreateTimer` tworzy timery. Parametry przekazane tej funkcji to: nazwa timera, częstotliwość (w milisekundach) i nazwa funkcji/procedury, która ma być wywoływana cyklicznie z zadaną częstotliwością.

WSKAZÓWKI

Klasa `TStringList` pozwala na obsługę danych tekstowych. Najciekawsze jej metody i właściwości w kontekście obiektu `L : TStringList`:

- `L.Count` - zwraca liczbę linii tekstu, które przechowuje obiekt;
- `L.Add('test')` - dodaje kolejną linię tekstu;
- `L.Delete(numer)` - usuwa linię o podanym numerze;
- `L.Text` - daje dostęp do pełnej zawartości przechowywanego tekstu;
- `L.SaveToFile(nazwaPliku)` - zapisuje zapamiętany tekst do pliku;
- `L.LoadFromFile(nazwaPliku)` - wczytuje zawartość pliku;
- `L.Clear` - czyści przechowywany tekst;

PRZYKŁAD 2

```
procedure plikiINI;
var Ini : TIniFile;
begin
 Ini:=TIniFile.Create('c:\plik.ini','',false);
 Ini.WriteString('Sekcja','Nazwa','Wartosc');
 Ini.Free;
 ShowMessage('Zapisano');
end;
```

WSKAZÓWKI

Inne przydatne metody i właściwości w kontekście obiektu `Ini : TIniFile`:

- `Ini.WriteString('Sekcja','Nazwa','Wartość')` - zapamiętanie ciągu `wartosc` dla pola `nazwa` w sekcji `sekcja`;
- `Ini.WriteInteger('Sekcja','Nazwa',Wartość)` - metoda analogiczna do poprzedniej, jednak wartość jest typu Integer (liczba całkowita);
- `Ini.ReadString('Sekcja','Nazwa')` - pobiera wartość pola `nazwa` z sekcji `sekcja`;
- `Ini.ReadInteger('Sekcja','Nazwa')` - metoda analogiczna do poprzedniej, jednak zwraca wartość typu Integer (liczba całkowita);

PRZYKŁAD 3

```
procedure rejestrWindows;
var Reg : TRegistry;
begin
  Reg:=TRegistry.Create(KEY_ALL_ACCESS);
  Reg.RootKey:=HKEY_CURRENT_USER;
  if Reg.OpenKey('Software\BinSoft\BSX Printer', TRUE) then
  begin
    ShowMessage(Reg.ReadString('Folder'));
  end;
  Reg.Free;
end;
```

PRZYKŁAD 4

```
procedure bazaDanych;
var B : TBinDatabase;
    T : TBinTable;
    L : TStringList;
begin
  try

B:=TBinDatabase.CreateRemote('mysql','localhost',3306,'root','karol','binsoft',
'','',22,'','');
    L:=TStringList.Create;
    try
      B.Tables(L);
      ShowMessage('Tabel: '+IntToStr(L.Count));
      if L.Count>0 then
      begin
        T:=B.GetRow('SELECT count(*) FROM '+L[0]);
        if Assigned(T) then
        begin
          ShowMessage('W tabeli '+L[0]+' jest '+T.FieldValueS['count(*)']+'
rekordow.');
```

```
          T.Free;
        end;
      end;
    finally
      B.Free;
      L.Free;
    end;
  except
      ShowMessage('Nie dalo sie nawiazac polaczenia z baza danych. Sprawdz, czy
dane podano prawidlwo. '+_NL+'Komunikat: '+LastExceptionMessage);
  end;
end;
```

PRZYKŁAD 5

```
procedure plikiXML;
var XML,F,E;
begin
  XML := TXMLDocument.Create;
  try
    XML.LoadFromFile('c:\plugin.xml');
    XML.Active:=TRUE;
    if Assigned(XML) then
    begin
      F:=XML.DocumentElement.FindNodeS('forms');
      if Assigned(F) then
      begin
       E:=F.FindNodeS('form');
       while Assigned(E) do
       begin
         ShowMessage(E.getAttr('caption','Domyslne'));
         E:=E.Next;
       end;
      end;
    end;
```

```
  finally
    XML.Free;
  end;
end;
```

ZDARZENIA

Użytkownik ma możliwość tworzenia własnych procedur i funkcji wewnątrz plików skryptowych `.pas`. Tworząc te podprogramy, programista nadaje im własne nazwy. Jednakże jeśli nazwie on swoją procedurę w odpowiedni sposób, Bramka KSeF będzie ją wywoływała automatycznie, w określonych sytuacjach. Na przykład, jeśli w pliku znajdzie się procedura o nazwie `fOnFormCreate()` - wywołana zostanie zawsze automatycznie, w momencie tworzenia danego formularza. Funkcja o nazwie `fOnFormClose()` wywołana będzie w momencie zamykania okna. Poniżej znajduje się lista wszystkich podstawowych zdarzeń:

- `fOnFormCreate()` - wywoływana w momencie tworzenia okna;
- `fOnFormClose()` - wywoływana w momencie zamykania okna;
- `fOnTimer()` - funkcja wywoływana zgodnie z częstotliwością ustawioną w konfiguracji programu;

OBSŁUGA BAZ DANYCH

Z poziomu kodu Pascala mamy dostęp do klasy `TBinDatabase`. Klasa ta zapewnia obsługę wielu rodzajów baz danych, m.in.: SQLite, MySQL, MS SQL, Firebird, PostgreSQL.

Oto dostępne konstruktory:

- `CreateSQLite(nazwaPliku, Haslo)` - tworzy lub otwiera bazę danych SQLite zapisaną w pliku `nazwaPliku`. Jeśli baza jest zaszyfrowana (przy otwieraniu) lub chcemy by była zaszyfrowana (przy tworzeniu) należy podać hasło. W przeciwnym wypadku jako hasło wpisujemy ciąg pusty.
- `CreateFirebird(nazwaPliku)` - otwiera bazę danych Firebird z podanego pliku;
- `CreateRemote(protokół,host,port,login,pass,nazwaBazy,tunelHost,sshHost,sshPort,sshUser,sshPass)` - nawiązuje połączenie z dowolną bazą danych. Jako

protokół podajemy nazwę protokołu bazy danych, tzn. ciągi: MySQL, PostgreSQL, MSSQL, Firebird, SQLite. `Host` - to adres serwera, `port` - numer portu na których serwer nasłuchuje, `login` - nazwa użytkownika, `pass` - hasło dostępu i `nazwaBazy` - nazwa bazy danych (musi istnieć). Jeśli chcemy użyć tunelowania HTTP możemy podać adres do pliku `tunnel.php`. Chcąc skorzystać z tunelowania poprzez SSH - należy podać kolejne parametry dające dostęp do powłoki SSH.

Oto kilka najciekawszych metod i właściwości jakie ta klasa oferuje:

- `ExecSQL(zapytanie)` - wykonanie zapytania typu INSERT, UPDATE, DELETE;
- `GetRow(zapytanie)` - wykonanie zapytania typu SELECT, zwracające jeden wiersz odpowiedzi; Jeśli zapytanie nie zwróci wyników metoda zwróci wartość `NIL`, w przeciwnym wypadku zwróci obiekt typu `TBinTable`;
- `GetRows(zapytanie)` - wykonanie zapytania typu SELECT zwracające dowolnie wiele wierszy; Metoda zawsze zwróci obiekt typu `TBinTable`;
- `Start` - rozpoczęcie transakcji;
- `Commit` - zatwierdzenie transakcji;
- `Rollback` - cofnięcie transakcji;
- `AddSlashes(tekst)` - zwraca tekst poprzedzony symbolem \ przed niebezpiecznymi znakami;
- `AddDate(data)` - dodanie daty (typu `TDateTime`) to zapytania w odpowiedni dla danego typu bazy danych sposób;
- `AddDateTime(data)` - dodanie daty i godziny (ze zmiennej typu `TDateTime`) do zapytania w odpowiedni dla danego typu sposób;

Dane zwracane z bazy danych są obiektami typu `TBinTable`. Oto najważniejsze metody i właściwości obiektów tego typu:

- `Eof` - zwraca true jak skończą się wyniki odpowiedzi;
- `Next` - przejście do następnego wiersza odpowiedzi;
- `FieldValueS['nazwa']` - zwraca zawartość pola nazwa jako ciąg znaków;
- `FieldValueL['nazwa']` - zwraca zawartość pola nazwa jako liczbę całkowitą;

- `FieldValueD['nazwa']` - zwraca zawartość pola nazwa jako liczbę zmiennoprzecinkową;
- `FieldValueB['nazwa']` - zwraca zawartość pola nazwa jako zmienną typu `Boolean`;
- `Columns[i]` - zwraca nazwę i-tej kolumny;
- `Types[i]` - zwraca typ i-tej kolumny;
- `ColCount` - zwraca liczbę kolumn odpowiedzi;
- `RowCount` - zwraca liczbę wierszy odpowiedzi;
- `FieldTypeByName['nazwa']` - zwraca typ kolumny o podanej nazwie;
- `FieldExists['nazwa']` - zwraca true jak istnieje kolumna o podanej nazwie;

Chcąc dodać lub modyfikować dane w bazie danych można użyć odpowiedniego zapytania i metody `ExecSQL` obiektu `TBinDatabase`, ale można również skorzystać z dodatkowej metody `InsertUpdateRow` tej klasy. Jako parametr tej metody podajemy nazwę tabeli, na której chcemy wykonywać zmiany. Metoda zwraca obiekt typu `TBinInsertUpdateRow`. Ta klasa z kolei oferuje metody:

- `AddS(nazwa,typ,wartość)` - ustalenie wartości dla pola `nazwa`, typu `typ`. `Wartość` jest zmienną typu String;
- `AddL(nazwa,typ,wartość)` - ustalenie wartości dla pola `nazwa`, typu `typ`. `Wartość` jest zmienną typu Integer;
- `AddD(nazwa,typ,wartość)` - ustalenie wartości dla pola `nazwa`, typu `typ`. `Wartość` jest zmienną typu Double;
- `AddB(nazwa,typ,wartość)` - ustalenie wartości dla pola `nazwa`, typu `typ`. `Wartość` jest zmienną typu Boolean;
- `Execute(id)` - wykonanie zapytania; Jeśli podano `id` (Integer) - nastąpi modyfikacja rekordu (`UPDATE`). Jeśli jako `id` wprowadzono wartość 0 - zostanie dodany nowy rekord.

Przykład 1:

```
var B, T;
begin
  B:=TBinDatabase.CreateSQLite('nazwa.db','');

  T:=B.GetRow('SELECT * FROM tabela WHERE id=1');
  if T<>nil then
  begin
    ShowMessage(T.FieldValueS['imie']);
    T.Free;
  end;
  B.Free;
end;
```

Przykład 2:

```
var B, T;
begin

B:=TBinDatabase.CreateRemote('MSSQL','localhost\SQLEXPRESS',1433,'sa','karol','
mpfirma','','',0,'','');

  T:=B.GetRows('SELECT nnodoc FROM bs_invoices LIMIT 5');
  while not T.Eof do
  begin
    ShowMessage(T.FieldValueS['nnodoc']);
    T.Next;
  end;
  T.Free;
  B.Free;
end;
```

Przykład 3:

```
var B, E;
begin
```

```
  B:=TBinDatabase.CreateSQLite('nazwa.db','');


  E:=B.InsertUpdateRow('imiona');
  E.AddS('imie','varchar','Karol');
  E.AddS('wiek','int','32');
  E.Execute(0);
  E.Free;


  B.Free;
end;
```

FILTRY

Bramka KSeF posiada wbudowaną obsługę plików `.in` oraz `.xml` - zgodnie z opisem w tym podręczniku. Poprzez własną wtyczkę możemy tworzyć tzw. filtry i rozszerzać obsługę Bramki KSeF i inne typy plików.

UWAGA! *Przypominamy, że kiedy Bramka KSeF przetwarza plik (.xml lub .in) - przed tą operacją zmienia rozszerzenie tych plików na .pr - w celu zabezpieczenia przed powtórnym przetwarzaniem.*

Chcąc utworzyć filtr należy w pliku XML wtyczki umieścić sekcję `<filters>`, a w niej znaczniki `<filter>`. Znacznik ten może przyjąć atrybuty: `type` - rodzaj filtru oraz `src` - nazwa pliku ze skryptem, gdzie znajduje się procedura obsługująca dany filtr.

Oto dostępne typy filtrów:

- `xml` - przetwarzanie pliku XML;
- `file` - przetwarzanie innego typu pliku;
- `folder` - przetwarzanie folderu z plikami;

FILTR XML

Kiedy Bramka KSeF przetwarza plik XML, przed rozpoczęciem tej procedury możne wykonać się zdefiniowany przez nas filtr. Spójrzmy na przykład:

```
<filters>
 <filter type="xml" src="plugin.pas" />
</filters>
```

Kiedy teraz będzie miał być przetwarzany plik XML, Bramka KSeF załaduje plik `plugin.pas` i wywoła z niego funkcję `fOnConvert(lFileName : String; Code : String) : String`. Przekaże do tej funkcji nazwę przetwarzanego pliku (`lFileName`), a w parametrze `Code` - jego treść (kod XML). Funkcja ta może zwrócić „inną" treść pliku XML lub ciąg „`@false`" - jeśli nie chcemy by ten plik był przetwarzany. Spójrzmy na przykładową zawartość pliku plugin.pas.

```
function fOnConvert(lFileName : String; Code : String) : String;
var XML : TXMLDocument;
    E,E2 : IXMLNode;
    LP: Integer;
    R,lName,lPrice,lQuantity,lVAT,lNIP : String;
begin
  LP:=0;
  R:='<?xml version="1.0" encoding="utf-8"?>'+_NL+'<root>'+_NL+'
<receipts>'+_NL;
  XML:=TXMLDocument.Create;
  try
    XML.LoadFromCode(Code);
    try
      XML.Active:=TRUE;
    except
      Exit;
    end;
    if XML.DE=nil then Exit;

    if XML.DE.Name<>'fmdoc' then Exit;

    E:=XML.DE.FindNodeS('Documents');
    if E=nil then Exit;
```

```
    E:=E.FindNodeS('Document');
    while E<>nil do
    begin
      NIP:=E.GetElementValue('NIP','');

      E2:=E.FindNodeS('Pozycje');
      if E2=nil then Exit;
   R:=R+'  <receipt step="1" symbol="'+E.GetElementValue('Symbol','')+'"
vatid="'+lNIP+'">'+_NL;
      while E2<>nil do
      begin
          lName:=E2.GetElementValue('NM','');
          lPrice:=E2.GetElementValue('CN','');
          lQuantity:=E2.GetElementValue('QT','');
          lVAT:=E2.GetElementValue('VT','');

    R:=R+'   <item name="'+lName+'" price="'+CorrectS2S(lPrice)+'"
quantity="'+lQuantity+'" vatrate="'+lVAT+'" />'+_NL;

          E2:=E2.Next;
      end;
      R:=R+'  </receipt>'+_NL;

      E:=E.Next;
      Inc(LP);
    end;

  finally
    XML.Free;
  end;
  R:=R+' </receipts>'+_NL+'</root>';
  if LP>0 then Result:=R;
end;
```

Zadaniem tego filtra jest „konwersja" pliku XML do formatu obsługiwanego przez Bramkę KSeF.

FILTR FOLDER

Filtr typu `folder` pozwala na dodanie obsługi plików innego rodzaju. Standardowo kiedy Bramka KSeF analizuje zawartość folderu, wyszukuje plików z rozszerzeniami `.in` i `.xml`. Po tej operacji uruchamia filtry typu `folder`. Mogą one „dodać" do listy plików do przetworzenia kolejne. Spójrzmy na przykład:

```
<filters>
 <filter type="folder" src="plugin.pas" />
</filters>
```

Teraz w momencie przetwarzania folderu Bramka KSeF będzie ładowała skrypt `plugin.pas` i wywoływał z niego funkcję: `fOnFolder(lFolder : String) : String`. Do tej funkcji w parametrze przekazuje nazwę analizowanego folderu. Funkcja ta może zwrócić ciąg tekstowy, będący listą dodatkowych plików do przetworzenia (oddzielonych znakiem nowej linii). Na przykład, gdybyśmy chcieli dodać obsługę plików `.csv` funkcja ta by wyglądała następująco:

```
function fOnFolder(lFolder : String) : String;
var L : TStringList;
begin
  L:=TBSXUtils.DirectoryList(lFolder,'*.csv',nil);
  try
    Result:=L.Text;
  finally
    L.Free;
  end;
end;
```

UWAGA! *Jeśli wskażemy dla Bramki KSeF jaki inny plik ma być przetwarzany, jego rozszerzenie również będzie automatycznie zmieniane na .pr w momencie rozpoczęcia jego przetwarzania.*

FILTR FILE

Kiedy przetwarzany jest przez Bramkę KSeF plik inny niż `.xml` - moment ten możemy przechwycić właście filtrem typu `file`. W ten sposób możemy rozszerzyć obsługę Bramki KSeF o obsługę niemal dowolnych plików. Spójrzmy na przykład XML:

```
<filters>
<filter type="file" src="plugin.pas" />
</filters>
```

Wskazaliśmy w nim, że skryptem powiązanym z filtrem file jest `plugin.pas`. Gdy taki plik będzie analizowany Bramka KSeF automatycznie wywoła z niego funkcję: `fOnParseFile(lFileName,lOrgFileName,lPrinterName : String) : String`. Przekazane parametry to: nazwa przetwarzanego pliku (`lFileName` - jest to pliku z już zmienionym rozszerzeniem na `.pr`), oryginalna nazwa (`lOrgFileName`) oraz nazwa drukarki, do której kierowany jest wydruk (`lPrinterName`). Spójrzmy na przykład:

```
function fOnParseFile(lFileName,lOrgFileName,lPrinterName : String) : String;
var L,P : TStringList;
    E : TBinInsertUpdateRow;
    i : Integer;
    lNip : String;
begin
  if TBSXUtils.URLExtractFileExt(lOrgFileName)='.csv' then
  begin
     L:=TBSXUtils.LoadLinesFromFile(lFileName,nil);
     P:=TStringList.Create;
     try
       for i:=0 to L.Count-1 do
       begin
          if Trim(L[i])='' then Exit;
          TBSXUtils.ExplodeStrS(P,L[i],';',true);
          if P.Count>=5 then
          begin
            lNip:='';
            if P.Count>=6 then lNip:=P[5];
            E:=DB.InsertUpdateRow('bsx_receipts');
```

```
            try
              E.AddS('iddoc','int','0');
              E.AddS('psource','int','4');
              E.AddS('ptype','int','1');
              E.AddS('pstatus','int','0');
              E.AddS('perrorcode','int','0');
              E.AddS('perrorstr','varchar','');
              E.AddS('premoteid','varchar',Trim(P[0]));
              E.AddS('premotestep','varchar','1');
              E.AddS('pprinter','varchar',lPrinterName);
              E.AddS('pdata','text','<root><receipts><receipt id="'+Trim(P[0])
+'" step="1" symbol="'+Trim(P[0])+'" vatid="'+lNIP+'"
printer="'+lPrinterName+'"><item name="'+Trim(P[1])+'" price="'+Trim(P[4])+'"
quantity="'+Trim(P[3])+'" vatrate="'+Trim(P[2])+'" /></receipt></receipts></
root>');
              E.AddS('pdate','datetime',DateTimeToStr(Now));
              E.Execute(0);
            finally
              E.Free;
            end;
          end;
       end;
     finally
       P.Free;
     end;
  end;
end;
```

W ten sposób sprawiliśmy, że kiedy przetwarzany jest plik `.csv` - nasz filtr wczyta ten plik, a następnie przeanalizuje jego każdą linię. Na tej podstawie zrobi wpisy w wewnętrznej bazie danych (w tabeli `bsx_receipts`). To spowoduje ich automatyczne wydrukowanie.

WŁASNE KOMENDY

Łącząc się do Bramka KSeF poprzez TCP/IP, WebSocket, serwer HTTP lub tworząc pliki `.in` - możemy przekazywać różne komendy, jakie mają być przez niego wykonane. W podręczniku tym komendy te zostały opisane w poprzednich rozdziałach. Programista ma jednak możliwość rozszerzenia funkcjonalności Bramki KSeF w tym zakresie dodając obsługę własnych komend.

Aby tego dokonać tworzymy tzw. *moduły* do serwera. Robimy to poprzez stworzenie „wtyczki" składającej się z pliku XML oraz powiązanego z nim pliku PAS. W pliku XML tworzymy sekcję `<modservers>`, a w niej dowolnie wiele znaczników `<modserver/>` z jednym parametrem `src` wskazującym powiązany plik Pascal. W pliku tym powinna znaleźć się funkcja: `function onServerCommand(Sender, CMD, P, LINE, AIdent, AContext, AAuthorized) : String;`.

Mając tak utworzoną wtyczkę i zarejestrowany *moduł*, kiedy przyjdzie do Bramki KSeF jakakolwiek komenda - np. poprzez telnet, WebSocket czy z pliku `.in` - Bramka KSeF wywoła z każdego *modułu* funkcję `onServerCommand()` z dodatkowymi parametrami. Jeśli funkcja ta zwróci napis `„true"` - spowoduje przerwanie szukania obsługi danej komendy, gdyż znaczy to, że komenda została obsłużona w tym *module*. Parametry funkcji `onServerCommand` oznaczają: `Sender` - skąd przyszło wywołanie funkcji, `CMD` - komenda (pierwszy element do znaku #, zawsze zapisana wielkimi literami), `P` - parametry dodatkowe (po symbolu #), `LINE` - pełna linia tekstu komendy, `AIdent` - identyfikator „sesji", z której przyszła komenda, `AContext` - kontekst do sesji, z której przyszło polecenie, `AAuthorized` - wartość true/false informująca czy użytkownik wysyłający komendę jest uwierzytelniony.

Spójrzmy na przykład pliku XML:

```
<module name="BinSoft.Plugin">
<modservers>
 <modserver src="modServer.pas" />
</modservers>
</module>
```

Oraz plik `modServer.pas`:

```
function onServerCommand(Sender, CMD, P, LINE, AIdent, AContext, AAuthorized) :
String;

var k : Integer;

    lTo : String;

begin

  if CMD = '' then Exit;


  if CMD = 'TEST' then

  begin

    Main.SendText(AContext,'Działa!');

    Result := 'true';

  end;

end;
```

W pliku XML zdefiniowaliśmy jeden *moduł* z powiązanym plikiem `modServer.pas`, a w nim jest funkcja `onServerCommand()`. Kiedy do Bramki KSeF przyjdzie jakakolwiek komenda, funkcja ta będzie wywoływana automatycznie. Sprawdzamy w niej, czy komenda to: `TEST`. Jeśli tak - to jest to komenda obsługiwana przez nasz *moduł*, dlatego zwrócimy wartość tekstową „`true`”. W pozostałych przypadkach nic nie zwracamy, więc Bramka KSeF będzie dalej przetwarzała przychodzące komendy. W przypadku komendy `TEST` używamy funkcji `Main.SendText(kontekst, tekst)`. Pozwala ona na „wysłanie” odpowiedzi pod odpowiedni kontekst - w tym przypadku do osoby/miejsca, z którego przyszła komenda.

Uwaga! Kiedy ktoś łączy się do Bramką KSeF poprzez TCP/IP lub WebSocket tworzony jest dla niego tzw. kontekst i przypisany identyfikator. Za jego pośrednictwiem serwer wie, do kogo ma wysyłać odpowiedzi itp.

PYTANIA I ODPOWIEDZI

Poniżej prezentujemy listę często zadawanych pyta, jakie otrzymujemy na w dziale pomocy technicznej. Zanim więc zadasz nam pytanie - zerknij na listę poniżej.

CZY BRAMKA KSEF MOŻE PRZECHOWYWAĆ SWOJĄ BAZĘ W MYSQL?

Tak. Domyślnie Bramka KSeF tworzy lokalną bazę danych SQLite, w której przechowuje swoje informacje (listę przetworzonych dokumentów itp). Możemy jednak zmienić ten parametr i wykorzystać inne miejsce do tego celu. Wystarczy wyedytować plik `config.ini` i w sekcji `[Settings]` dopisać opcję `DatabaseURL` z parametrem opisującym bazę danych. Przykład:

```
DatabaseURL=MySQL#localhost#3306#root#bsx#ksefdb
```

Powyższa linia spowoduje, że Bramka KSeF podczas uruchamia połączy się z serwerem `MySQL` na komputerze pod adresem `localhost`, porcie `3306`. Użyje użytkownika `root`, hasła `bsx` i wybierze bazę danych o nazwie `ksefdb`. Stworzy tam niezbędne dla siebie tabele i w nich będzie przechowywała swoje informacje.

CZY MOŻNA BEZPOŚREDNIO MANIPULOWAĆ WEWNĘTRZNĄ BAZĄ DANYCH BRAMKI KSEF?

Tak. Użytkownik może np. samodzielnie dodawać wpisy do wewnętrznej tabeli `bsx_ksef`, a Bramka KSeF będzie te zmiany „widział" i będzie na nie reagował.

Możliwy scenariusz integracji z Bramką KSeF jest zatem na przykład taki, że w jego pliku konfiguracyjnym podajemy dane dostępowe do swojego serwera MySQL, na którym działa nasza aplikacja (np. sklep on-line). Po uruchomieniu Bramka KSeF połączy się do tej bazy i automatycznie stworzy w niej swoją strukturę tabel. Następnie programista z poziomu swojej aplikacji może dodawać wpisy do tabel Bramki KSeF i w ten sposób zlecać wydruk paragonu, lub też odczytywać status wydruku itp.

POMOC TECHNICZNA

W przypadku wszelkich problemów w funkcjonowaniu aplikacji Bramka KSeF lub w przypadku zaistnienia dodatkowych potrzeb jeśli chodzi o tę aplikację - prosimy pisać na adres: pomoc@binsoft.pl.