

Biuletyn techniczny

Hydra

Wersja: 2017.0



Comarch ERP
XL

Copyright © 2016 COMARCH

Wszelkie prawa zastrzeżone.

Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie na nośniku filmowym, magnetycznym lub innym, powoduje naruszenie praw autorskich niniejszej publikacji.

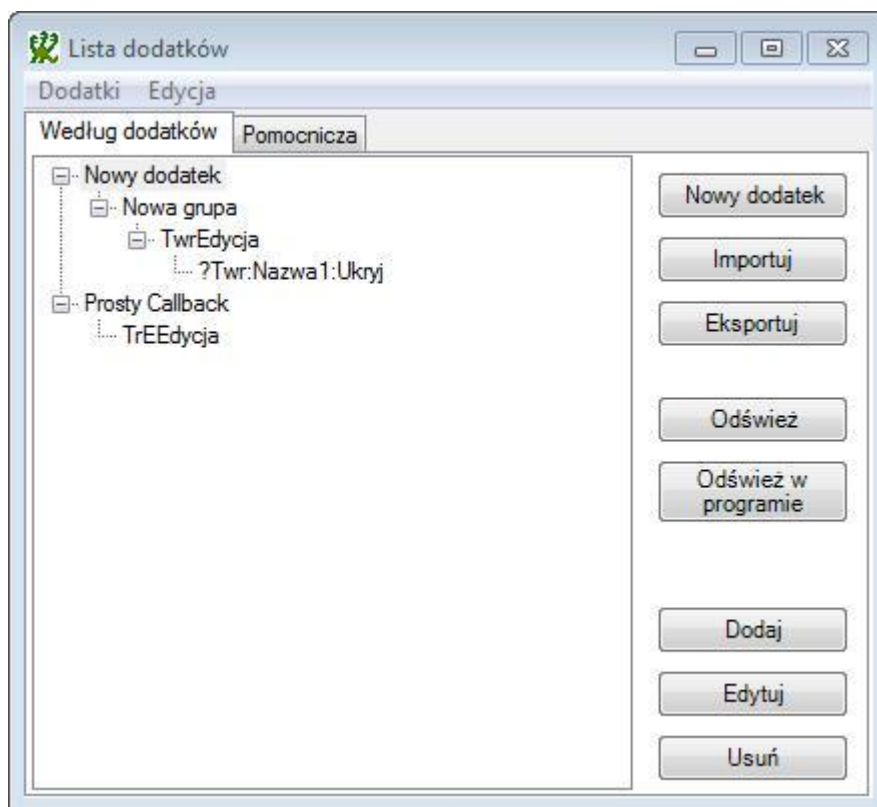
Spis treści

Spis treści.....	3
1 Mała Hydra	5
2 Duża Hydra - informacje ogólne.....	9
2.1 Callbacki	9
2.2 Przygotowanie Callbacka	9
2.3 Operacje na kontrolkach	12
2.4 Podstawowe kontrolki	12
2.4.1 Główne właściwości kontrolek.....	12
2.4.2 Box	13
2.4.3 Panel	13
2.4.4 Image	13
2.4.5 Button	13
2.4.6 Menu.....	14
2.4.7 Group.....	14
2.4.8 Text.....	15
2.4.9 List.....	15
2.4.10 Progress	15
2.4.11 Region.....	15
2.4.12 Spin	15
2.4.13 Check	16
2.4.14 DropCombo.....	16
2.4.15 Droplist	17
2.5 Wyświetlanie komunikatów (dotyczy wyłącznie wersji SE)	17
3 Ogólne wskazówki związane z tworzeniem callbacków	18
3.1 Odczytanie nazwy kontrolki	18
3.2 Bufor Tabeli	18
3.3 Identyfikator błędu w przypadku zdarzeń plikowych	18
3.4 Obsługa list z poziomu Hydry	18
3.5 Zmiana wartości w kontrolce Check	20
4 Przykłady	21
4.1 Zdefiniowanie połączenia do bazy bez potrzeby podawania ConnectionString.....	21
4.2 Wywołanie zapytania na bazie bez potrzeby otwierania nowego połączenia do bazy.	21
4.3 Aktualna data i godzina	21
4.4 Odczyt zaznaczonych pozycji	21
4.5 Odczyt wartości atrybutu	21
4.6 Zmiana wartości pola tekstowego	21
4.7 Odświeżanie zawartości okna	22
4.8 Podniesienie do edycji wybranego okna.....	22

4.9	Ukrycie kolumny na liście.....	22
4.10	Debugowanie kodu z poziomu Visual Studio	22
4.11	Reakcja na zamknięcie okna.....	23
4.12	Ctrl+ins na dokumencie.....	23
4.13	Zmiana kontrahenta na dokumencie	23
4.14	Przechodzenie między oknami w trybie zaawansowanym	24
4.15	Wykonanie wydruku	24
4.16	Odczyt naciśniętego klawisza	24
4.17	Modyfikacja menu kontekstowego.....	24
5	Znane problemy w Hydrze	27
5.1	Współpraca API z oknami .net'owymi (dotyczy wyłącznie wersji SE)	27
5.2	Wywoływanie funkcji API z poziomu danego dokumentu	27
5.3	Ograniczenia callbacków plikowych.....	27
5.4	Hydra i nowy interfejs	27
	Spis ilustracji.....	28

1 Mała Hydra

Przez Małą Hydę rozumiemy mechanizm, który pozwala na ukrycie, wyłączenie lub ustawienie wymagalności dla kontrolek na istniejących oknach. Callbacki zarówno Małej jak i Dużej Hydry są widoczne na Liście dodatków. Lista ta umożliwia importowanie i eksportowanie callbacków, zmianę ich stanu, dodawanie, usuwanie oraz edycję.



Rys. 1 Lista dodatków

Na liście dodatków znajdują się następujące przyciski:

Nowy dodatek – użycie funkcji rozpocznie proces dodawania nowego callbacka.

Importuj – funkcja umożliwia zaimportowanie zdefiniowanego callbacka ze wskazanej lokalizacji.

Eksportuj - funkcja umożliwia wyeksportowanie callbacka do wskazanej lokalizacji.

Odśwież – odświeżenie listy dodatków.

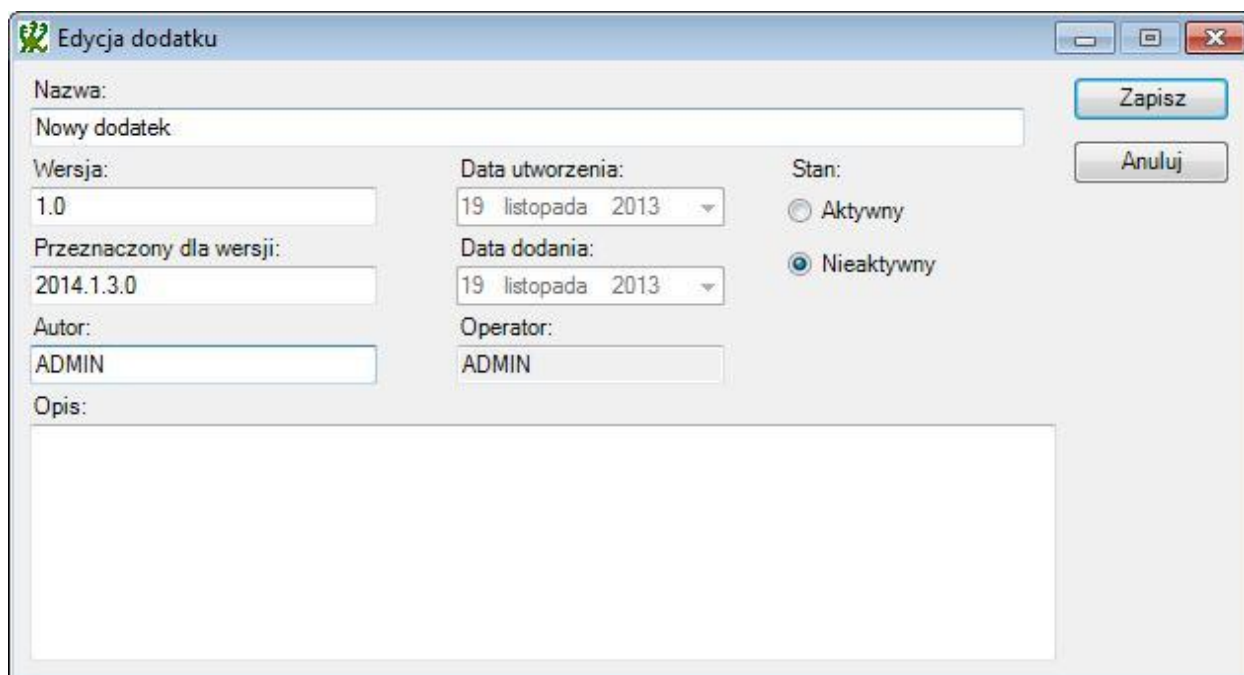
Odśwież w programie - funkcja odświeżania w całym programie, np. w przypadku usunięcia callbacka, zostanie on usunięty ze struktury firmy i innych obiektów w systemie dopiero po jej użyciu.

Dodaj - funkcja aktywna po dodaniu callbacka. Umożliwia jego szczegółowe zdefiniowanie.

Edytuj - umożliwia edycję zdefiniowanego callbacka.

Usuń – usuwa callbacka zaznaczonego na liście.

Aby dodać nowy dodatek Małej Hydry należy uruchomić funkcję Nowy dodatek. Zostanie podniesione okno Edycja dodatku.



Rys. 2 Edycja dodatku

Nazwa - w polu należy wprowadzić nazwę callbacka.

Wersja - wersja „Hydry”, w której został utworzony callback; pole wypełniane jest automatycznie.

Przeznaczony dla wersji - wersja systemu Comarch ERP XL, w której callback będzie funkcjonował; pole wypełniane automatycznie.

Autor - autor callbacka; pole wypełniane automatycznie - wartością wprowadzaną domyślnie będzie zalogowany operator.

Data utworzenia - data utworzenia callbacka; pole wypełniane automatycznie.

Data dodania - data dodania callbacka do systemu; pole wypełniane automatycznie.

Operator - operator dodający callback do systemu; pole wypełniane automatycznie.

Stan - zaznaczenie wybranej opcji zadecyduje o aktywności callbacka w systemie. Aby callback funkcjonował w systemie, musi mieć zaznaczoną opcję: Aktywny.

Opis - w polu można wprowadzić opis callbacka.

Aby skonfigurować callbacka należy zaznaczyć dodatek i uruchomić funkcję Dodaj. W efekcie zostanie podniesione okno Edycja grupy procedur. W tym miejscu istnieje możliwość zdefiniowania dodatkowego warunku, na zakładanym callbacku. Warunek musi być zdefiniowany zgodnie ze składnią języka C# i musi zwracać wartość logiczną. W przypadku zwrócenia wartości true callback zostanie wykonany, w przypadku zwrócenia wartości false działanie callbacka zostanie zablokowane. W programie możemy definiować dwa rodzaje złożonych warunków callbacka:

Warunek oparty o zmienne słownikowe – warunek można skonstruować w oparciu o zmienne zawarte w ConfigurationDictionary.



Przykład: Przykładowy warunek callbacka na zalogowanego operatora:

```
if (Runtime.ConfigurationDictionary.NumerOperatora == 1)
```

```
{ return false; }
```

```
else
```

```
{ return true; }
```

Warunek oparty o bufor tabeli – warunek może być oparty o dane zawarte w buforze tabeli.



Przykład: Przykładowy warunek na stan towaru:

```
if (TwrKarty.Twr_Typ==1)
```

```
{return false;}
```

```
else
```

```
{return true;}
```



Uwaga: Aby warunek oparty o bufor tabeli działał prawidłowo, należy wyłączyć na operatorze Otwieranie okien w trybie zaawansowanym. W przeciwnym wypadku operator będzie miał możliwość przechodzenia między obiektami (np. dokumentami) bez ponownego sprawdzania warunku callbacka.




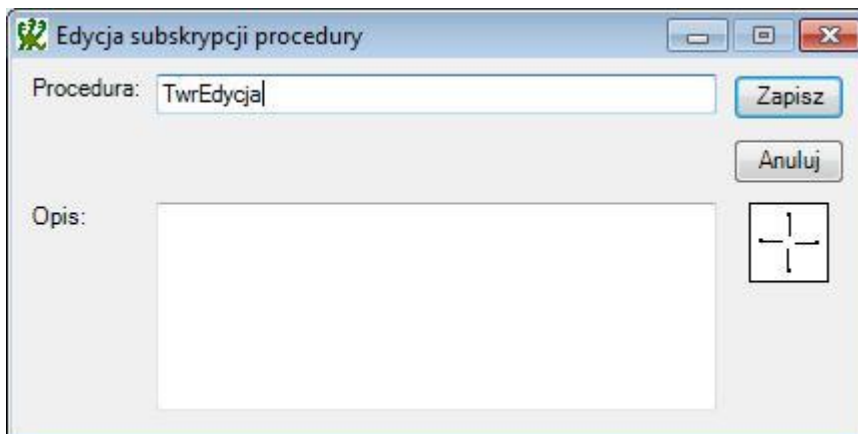
Uwaga: Warunek callbacka Małej Hydry jest sprawdzany tylko w momencie uruchamiania formatki.



Rys. 3 Edycja grupy procedur

W kolejnym kroku należy zaznaczyć na liście dodatków dodaną grupę i ponownie wywołać funkcję Dodaj. W efekcie zostanie podniesione okno: Edycja subskrypcji procedury. W tym miejscu należy podać nazwę procedury (okna), na którym będzie zdefiniowany dodatek. Aby odczytać nazwę procedury należy przeciągnąć

na dane okno znacznik .



Rys. 4 Edycja subskrypcji procedury

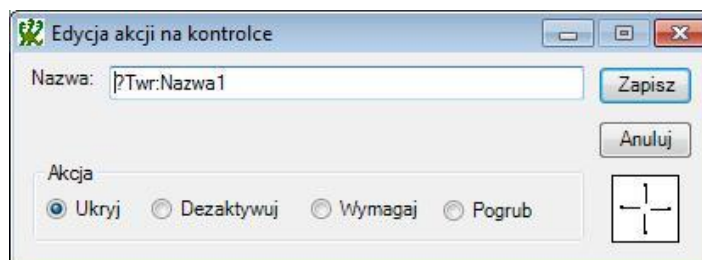
W ostatnim kroku należy zdefiniować zmienioną obsługę kontroltek. W tym celu należy na liście dodatków zaznaczyć dodaną procedurę i wywołać funkcję Dodaj. Zostanie podniesione okno Edycja akcji na kontrolce. W tym miejscu należy wskazać za pomocą znacznika odpowiednią kontrolkę na oknie, a następnie zdefiniować właściwą akcję:

Ukryj – ukrycie kontrolki na oknie

Dezaktywuj – dezaktywacja kontrolki na oknie

Wymagaj – ustawienie wymagalności kontrolki – nie będzie możliwe zamknięcie okna do czasu wprowadzenia wartości w kontrolce.

Pogrub – pogrubia czcionkę etykiety (pogrubiona czcionka zwykle służy do wizualnego oznaczenia pól wymaganych na formularzu)



Rys. 5 Edycja akcji na kontrolce



Uwaga: Akcja Wymagaj może być użyta tylko dla wybranych kontroltek. Dotyczy przede wszystkim kontroltek, dla których istnieje stan „bez wartości” np. kontroltek tekstowych. Akcja nie może być natomiast wykorzystana np. dla kontrolki check, gdyż kontrolka zawsze przyjmuje wartość (0 lub 1).

Aby dodatek był aktywny w systemie należy go dodać do właściwego centrum w strukturze praw. W tym celu należy uruchomić okno Edycja struktury firmy i dodać zdefiniowanego callbacka na zakładce Dodatki.

2 Duża Hydra - informacje ogólne

2.1 Callbacki

Callbacki Dużej Hydry są mechanizmem, który służy do dodawania nowej funkcjonalności, bez potrzeby modyfikacji binariów programu. Od strony technicznej callbacki są zrealizowane jako plik dll eksportujący określony zestaw funkcji, podzielonych na funkcje plikowe i okienkowe.

Funkcje plikowe są wywoływane przez program podczas uruchamiania lub zamykania formatki - wówczas każda akcja użytkownika powoduje wywołanie pary zdarzeń: poprzedzających akcję i następujących po niej. Callbacki z tej grupy są najczęściej wykorzystywane do kontroli poprawności zapisywanych danych oraz do zaawansowanej kontroli uprawnień operatora.

Funkcje okienkowe są wykorzystywane w celu modyfikacji wyglądu i zachowania formatek. Składają się na nie trzy funkcje: wywoływane podczas inicjalizacji okna, przy każdym zdarzeniu okna oraz przed zamknięciem okna. Callbacki z tej grupy są najczęściej wykorzystywane do ukrywania lub uniemożliwiania edycji niektórych pól na formatkach lub do dodawania elementów interfejsu.

Zdarzenia obu grup są bardzo często używane łącznie.

2.2 Przygotowanie Callbacka

Kod obsługujący zdarzenia należy umieścić w specjalnie przygotowanym module .NET'owym. Moduł taki musi zostać opisany za pomocą atrybutu `CallbackAssemblyDescription` w następujący sposób:

```
[assembly:CallbackAssemblyDescription("Nazwa callbacka","Opis callbacka","Autor callbacka","wersja callbacka","wersja systemu, dla której callback jest przeznaczony","data utworzenia callbacka w formacie dd-mm-rrrr")]
```

Aby obsłużyć zdarzenia plikowe i okienkowe, należy dodać do modułu publiczną klasę dziedziczącą po klasie `Callback`, zdefiniowaną w assembly `CdnHydra.dll`. Klasę taką należy opisać atrybutem `SubscribeProcedure` w następujący sposób:

```
[SubscribeProcedure(Procedura z enumeracji Procedures,"Opis subskrypcji")]
```

Klasa `Callback` jest abstrakcyjna, więc do skompilowania assembly będzie konieczne zdefiniowanie metod `Init` i `Cleanup`. Zdefiniowanie tych metod umożliwia użytkownikowi wykonanie kodu odpowiednio inicjującego i kończącego pracę z procedurą. Uwaga – nie należy umieszczać takiego kodu wewnątrz konstruktora ani metody `Dispose`, ponieważ jedna instancja klasy może być wykorzystywana wielokrotnie (każde kolejne uruchomienie przez użytkownika procedury będzie powodowało uruchomienie metody `Init`, ale nie konstruktora, podobnie z metodą `Cleanup`). W metodzie `Init` należy określić zainteresowanie zdarzeniami w procedurze. W przypadku zdarzeń plikowych subskrypcja powinna mieć postać:

```
AddFileActionSubscription(bool BeforeAction,  
                           FileActionTypes ActionType,  
                           FileActionDelegate ActionDelegate)
```

gdzie

BeforeAction – określa czy obsługa ma być wywoływana przed czy po zdarzeniu

FileActionTypes – określa typ zdarzenia plikowego, którego dotyczy subskrypcja

ActionDelegate – określa delegat wywoływany w odpowiedzi na zdarzenie

W przypadku zdarzeń okienkowych subskrypcja powinna mieć postać:

```
AddSubscription(bool BeforeHandling,
                 int ControlId,
                 Events EventId,
                 TakeEventDelegate EventDelegate)
```

gdzie

BeforeHandling – określa czy obsługa ma być wywoływana przed czy po zdarzeniu

ControlId – określa ID kontrolki (wartość 0 w przypadku ID okna)

EventId – określa zdarzenie, którego dotyczy subskrypcja

EventDelegate – określa delegat wywoływany w odpowiedzi na zdarzenie

Subskrypcje na zdarzenia mogą zostać usunięte za pomocą instrukcji:

```
RemoveFileActionSubscription(bool BeforeAction,
                             FileActionTypes ActionType,
                             FileActionDelegate ActionDelegate)
```

```
RemoveSubscription(bool BeforeHandling,
                   int ControlId,
                   Events EventId,
                   TakeEventDelegate EventDelegate)
```

Deklaracje delegatów dla zdarzeń plikowych i okienkowych powinny mieć odpowiednio postać:

```
public delegate bool FileActionDelegate (Procedures ProcedureId,
                                         GID GID,
                                         ADODB._Recordset Attributes);
```

gdzie

ProcedureId – id procedury, której dotyczy zdarzenie

GID – identyfikator rekordu, którego dotyczy zdarzenie

Attributes – zestaw atrybutów powiązanych z edytowanym rekordem

```
public delegate bool TakeEventDelegate (Procedures ProcedureId,
                                         int ControlId,
                                         Events EventId);
```

gdzie

ProcedureId – id procedury, której dotyczy zdarzenie

ControlId – id kontrolki, której dotyczy zdarzenie

EventId – id zdarzenia

Kod znajdujący się w callbacku ma dostęp do kontekstu, w którym został uruchomiony za pośrednictwem własności klasy ConfigurationManager. W ramach tej klasy są przechowywane następujące informacje:

- **Baza** string
- **BazaHurtowni** string
- **BazaOLAP** string
- **Centra** ADODB._Recordset
- **DomyslnyMagazynSprzedazy** string
- **DomyslnyMagazynZakupu** string

- **DomyslnyRejestrBankowy** string
- **DomyslnySkladCelny** string
- **IdCentrumKosztowego** int
- **IdCentrumPodleglosciowego** int
- **IdCentrumWlasciwego** int
- **IdFirmy** int
- **KasaTransakcjiGotowkowych** string
- **KatalogProgramu** string
- **KoniecOkresuObrachunkowego** datetime
- **KontoDostawcow** string
- **KontoOdbiorcow** string
- **LoginZintegrowany** bool
- **MagazynDocelowyKompletacji** string
- **Magazyny** ADODB._Recordset
- **MagazynZrodlowyKompletacji** string
- **NumerOperatora** int
- **PieczatkaEmail** string
- **PieczatkaMiasto** string
- **PieczatkaNazwa** string
- **PieczatkaNIP** string
- **PieczatkaNIPPrefiks** string
- **PoczatekOkresuObrachunkowego** datetime
- **RabatyOdCeny** bool
- **RejestryBankowe** ADODB._Recordset
- **SciezkaWStrukturze** string
- **Serie** ADODB._Recordset
- **SerieOddzialowe** ADODB._Recordset
- **Serwer** string
- **SerwerOLAP** string
- **TypNaliczaniaMarzy** Hydra.TypMarzy
- **TypyKlasAtrybutow** ADODB._Recordset
- **WalutaSystemowa** string
- **Waluty** ADODB._Recordset

Procedury HASPa:

- **HASPDaDataAplikacji** datetime
- **HASPDaDataDealera** datetime
- **HASPLicencje** ADODB._Recordset
- **HASPLiczbaBaz** int
- **HASPNazwaUzytkownika** string
- **HASPNumerBazyDanych** int
- **HASPNumerKlucza** int
- **HASPOpcje** int
- **HASPPlatforma** int
- **HASPStanKlucza** int

2.3 Operacje na kontrolkach

Kod znajdujący się w callbacku może wykonywać operacje na interfejsie (dodawać, przesuwać i ukrywać kontrolki, wypełniać je itp.) za pośrednictwem własności i metod klasy **ClWindow**. Instancję klasy **ClWindow** można uzyskać za pomocą metody **GetWindow** klasy **Callback** (w celu uzyskania głównego okna procedury) lub **GetControl** (w celu uzyskania kontrolki).

Klasa **ClWindow** jest wykorzystywana do operowania na oknach i kontrolkach Clarionowych. Każda instancja klasy **ClWindow** jest powiązana z kontrolką lub oknem, na którym operuje.

Kolekcja **AllChildren** przechowuje wszystkie kontrolki znajdujące się na oknie. Można ją wykorzystać do odnalezienia poszukiwanej kontrolki. Pozostałe własności klasy odpowiadają własnościom kontrollek Clarionowych, które zostały opisane w dalszej części biuletynu.

Zdarzenia do kontrolki mogą być wysyłane za pośrednictwem metody:

```
public void PostEvent(Events EventId)
```

gdzie:

EventId – identyfikator zdarzenia, które ma zostać wysłane do kontrolki

Bardzo często zdarzenia do kontrolki są wysyłane parami – po wysłaniu określonego zdarzenia wymagane jest wysłanie zdarzenia **Accepted** w celu zatwierdzenia wprowadzonej wartości.

Podstawowe zdarzenia dla kontrollek:

OpenWindow/JustAfterWindowOpening – otwarcie okna

ResizeWindow – zmiana wielkości okna

NewSelection – kliknięcie w spin, przejście do innej zakładki

Accepted – akceptacja wprowadzenia nowej wartości do pola (odpowiednik wyjścia z kontrolki za pomocą tabulatora), kliknięcie przycisku.

ScrollDown/ScrollUp – scrollowanie listy

2.4 Podstawowe kontrolki

2.4.1 Główne właściwości kontrollek

Visible – (true/false) – widoczność kontrolki

Disable – kontrolka nieaktywna

Skip – kontrolka jest pomijana podczas przechodzenia tabulatorem

TextRaw – tekst widoczny na kontrolce

Bounds – współrzędne kontrolki

FontSizeRaw – wielkość czcionki

FontColorRaw – kolor czcionki

BackgroundRaw – kolor tła



Uwaga: Lista możliwych kolorów została przedstawiona w pliku **kolory.txt** w załącznikach do biuletynu.

Przed przypisaniem wartości należy przekonwertować kody kolorów z notacji szesnastkowej do postaci dziesiętnej.

FontStyleRaw – format tekstu



Uwaga: Możliwe wartości dla wybranych formatów:

100 – cienki tekst, 400 – zwykły tekst, 700 – pogrubiony tekst, 4096 – kursywa, 8192 –

podkreślenie, 16384 – tekst przekreślony

LeftRaw/RightRaw/CenterRaw – wyrównywanie tekstu w kontrolce

TipRaw – tekst wyświetlany w tooltipie kontrolki

MsgRaw – opis wyświetlany w stopce okna podczas aktywności kontrolki

2.4.2 Box

Kontrolka typu Box definiuje ramkę na oknie (prostokątny obszar).

ColorRaw – kolor ramki

RoundRaw – zaokrąglenie ramki

FillRaw – kolor wypełnienia

LineWidthRaw – grubość ramki

2.4.3 Panel

Kontrolka typu Panel definiuje obszar na oknie.

BevelRaw – efekt 3D dla ramki

2.4.4 Image

Kontrolka typu Image pozwala umieścić rysunek na oknie.

TextRaw – ścieżka do pliku z rysunkiem

2.4.5 Button

Kontrolka typu Button dodaje przycisk na oknie

IconRaw – przypisanie ikony do przycisku np. "~EXPORT20.ICO"



Uwaga: W przypadku przypisywania ikony na podstawie pliku należy podać pełną ścieżkę do ikony w formacie: @\"C:\WINDOWS\msnms.ico\".

W przypadku przypisywania standardowej ikony ERP XL należy podać nazwę ikony w formacie: „~EXPORT20.ICO”. Pełna lista istniejących ikon nie jest dostępna na zewnątrz.

FlatRaw – określenie czy przycisk jest płaski (wymagane jest przypisanie ikony do przycisku)

ReqRaw – naciśnięcie przycisku spowoduje sprawdzenie czy wszystkie pola Entry na oknie, które mają ustawiony parametr Req są wypełnione. Jeżeli nie są wypełnione to kursor zostanie przeniesiony do odpowiedniego pola Entry.

ImmRaw – moment wyzwolenia akcji na kontrolce. W przypadku przekazania do właściwości wartości „1” akcja zostanie wywołana w momencie kliknięcia przycisku, ale przed zwolnieniem przycisku myszy. W przypadku przekazania wartości „0” akcja zostanie wykonana po naciśnięciu i zwolnieniu przycisku myszy.

TextRaw – etykieta przycisku



Uwaga: Jeżeli w etykiecie dodamy znak & to pojawi się możliwość wywołania przycisku z klawiatury zgodnie z kolejną literą. Przykładowo przycisk o etykiecie &nazwa pozwoli na wywołanie kliknięcia poprzez alt+n.

StdRaw – przypisanie do przycisku standardowej akcji



Uwaga: Lista możliwych akcji została przedstawiona w pliku akcje.txt w załącznikach do biuletynu.

2.4.6 Menu

Kontrolka dodaje dodatkową opcję w Menu okna. Aby zbudować menu należy posłużyć się dwoma typami kontroltek: Menu i Item. Poniżej zaprezentowano przykład działania Menu:



Przykład:

```
parent = GetWindow();

menu1 = parent.AllChildren.Add(ControlTypes.menu);

menu1.Visible = true;

menu1.TextRaw = "Nowe menu";

menu2 = menu1.AllChildren.Add(ControlTypes.menu);

menu2.Visible = true;

menu2.TextRaw = "Menu poziom 2";

item1 = menu1.AllChildren.Add(ControlTypes.item);

item1.Visible = true;

item1.TextRaw = "Element 1";

item2 = menu1.AllChildren.Add(ControlTypes.item);

item2.Visible = true;

item2.TextRaw = "Element 2";

item3 = menu2.AllChildren.Add(ControlTypes.item);

item3.Visible = true;

item3.TextRaw = "Element 3";
```

2.4.7 Group

Kontrolka pozwala zgrupować na oknie inne kontrolki. Możliwe jest narysowanie ramki dla grupy i nadanie jej nazwy.

BoxedRaw – czy ma być wyświetlana ramka dla grupy (0 – nie, 1 – tak)

TextRaw – nazwa grupy

2.4.8 Text

Kontrolka typu Text dodaje pole tekstowe na oknie.

ToolTipRaw – tekst w tooltipie

SkipRaw – pole jest ignorowane podczas przechodzenia między kontrolkami za pomocą przycisku

Tabulatora

ReadOnlyRaw – pole ma status „tylko do odczytu”

TextRaw – domyślna wartość w polu tekstowym

2.4.9 List

Kontrolka typu List dodaje listę na oknie.

FromRaw – wypełnienie listy wartościami gdzie kolejne rekordy powinny być oddzielone od siebie znakiem |. Jeżeli taki znak ma zostać wpisany jako wartość pozycji to należy wpisać ||. Pustą wartość można uzyskać wpisując |

VScrollRaw – umieszczenie pionowego paska przewijania na liście

FormatRaw – zdefiniowanie kolumn listy

SelectedRaw – numer zaznaczonego elementu na liście



Uwaga: Aby przejść do kolejnego elementu listy można wykorzystać zdarzenie ScrollDown:

```
Runtime.WindowController.PostEvent(idkontrolki, Events.ScrollDown);
```

Ewentualnie może zająć potrzeba kliknięcia się w daną listę za pomocą zdarzenia SelectControl:

```
Runtime.WindowController.SelectControl(i);
```

2.4.10 Progress

Kontrolka Progress definiuje pasek postępu. Określenie postępu dla kontrolki określa się za pomocą parametru **ProgressRaw**.

2.4.11 Region

Kontrolka typu Region dodaje „interaktywny” obszar (Panel) na oknie.

ImmRaw – ustawienie parametru pozwala na obsługę zdarzeń typu MouseIn/MouseOut. Ponieważ właściwość może mieć wpływ na prędkość otwierania formatki należy ją definiować jedynie w przypadku wykorzystania dodatkowych zdarzeń.

DragIDRaw/DropIDRaw – obsługa zdarzeń typu Drag i Drop na obszarze.

2.4.12 Spin

RangeLowRaw – minimalna wartość dla kontrolki

RangeHighRaw – maksymalna wartość dla kontrolki

ScreenTextRaw – domyślna wartość dla kontrolki

FromRaw – lista możliwych wartości dla kontrolki oddzielonych od siebie znakiem |

TextRaw – określenie formatu pola



Uwaga: Przykładowe ustawienia dla pól typu data zostały przedstawione w pliku `daty.txt` w załącznikach. Data powinna być przekazana w formacie Clariona, czas w formacie Timestamp.

ValueRaw – określenie wartości domyślnej w polu (wymagane ustawienie `UseRaw="1"`)

Zmiana wartości w kontrolce:

```
Runtime.WindowController.Change(kontrolka.Id, "wartość")
```



Przykład: Przykład na zmianę wartości rabatu na dokumencie ZS:

```
Runtime.WindowController.Change(Parent.AllChildren["?ZaN:Rabat"].Id, "50");
```

```
Runtime.WindowController.PostEvent(Parent.AllChildren["?ZaN:Rabat"].Id, Events.Accepted);
```

2.4.13 Check

Kontrolka typu Check pozwala na zwrócenie wartości z przedziału {0,1}.

CheckedRaw – domyślna wartość kontrolki (1 – zaznaczony, 0 – odznaczony)

UseRaw – wartość w kontrolce przy odczycie

2.4.14 DropCombo

Kontrolka typu DropCombo jest połączeniem dwóch kontrol: List oraz Entry. Kontrolka pozwala na wybranie wartości z listy jednocześnie umożliwiając dopisanie/zmianę wartości.

FromRaw – wypełnienie listy wartościami gdzie kolejne rekordy powinny być oddzielone od siebie znakiem |. Jeżeli taki znak ma zostać wpisany jako wartość pozycji to należy wpisać ||. Pustą wartość można uzyskać wpisując |

VScrollRaw – umieszczenie pionowego paska przewijania na liście

DropRaw – liczba wierszy wyświetlanych na liście.

VCRRaw – pasek szybkiego przechodzenia po liście wartości

UprRaw – wpisanie wartości do pola będzie skutkowało automatyczną zmianą tekstu na duże znaki

SelectedRaw – wybranie pozycji na liście wartości

ScreenTextRaw – wpisanie wartości do pola entry



Przykład: Przykładowy kod na obsłużenie kontrolki o typie dropcombo:

```
ClWindow combo = Parent.Children.Add(ControlTypes.dropcombo);
```

```
combo.Visible = true;
```

```
combo.Bounds = new Rectangle(200, 255, 70, 10);
```

```
combo.FromRaw = "wartosc1|wartosc2|wartosc3";
```

```
combo.SelectedRaw = "2";
```

```
combo.ScreenTextRaw = "wartosc2";
```

```
AddSubscription(true, combo.Id, Events.NewSelection, new TakeEventDelegate(komunikat));
```

2.4.15 Droplist

Kontrolka typu Droplist pozwala na wybór wartości z rozwijanej listy bez możliwości jej edycji lub wprowadzania nowych wartości. Właściwości kontrolki w większości pozostają analogiczne jak na kontrolce Dropcombo.

2.5 Wyświetlanie komunikatów (dotyczy wyłącznie wersji SE)

Komunikat można wyświetlić za pomocą polecenia `MessageBox.Show(string)` po uprzednim odblokowaniu wątku. Właściwa obsługa wątków jest również konieczna na przykład w przypadku wywoływania okna .net'owego.



Przykład:

```
Runtime.WindowController.UnlockThread();
MessageBox.Show("Tekst komunikatu");
Runtime.WindowController.LockThread();
```

3 Ogólne wskazówki związane z tworzeniem callbacków

3.1 Odczytanie nazwy kontrolki

Nazwę danej kontrolki można odczytać korzystając z konstruktora Małej Hydry. W przypadku modułu Detal nazwy procedur rozpoczynają się od słowa Detal, a nazwy kontrolerek w większości są zgodne z analogicznymi kontrolkami w innych modułach.

3.2 Bufor Tabeli

W wielu miejscach programu mamy możliwość odwołania się do wartości przechowywanych w buforze tabeli. Przykładowo z poziomu formatki dokumentu możemy odczytać informacje na temat jego nagłówka bez konieczności otwierania połączenia do bazy. Aby wykonać taki odczyt należy wskazać nazwę tabeli wraz z nazwą kolumny w bazie np. TraNag.TrN_GIDNumer.

Warto zwrócić uwagę że w buforze pliku jest przechowywany tylko jeden, aktualny rekord. Dodatkowo bufor plików mogą być wykorzystywane przez główny kod programu na różne sposoby – np. mogą być modyfikowane w odpowiedzi na zdarzenia na oknie. Dlatego warto ograniczyć poleganie na informacjach znajdujących się w buforach plików do niezbędnego minimum, wszelkie dodatkowe informacje odczytując samodzielnie np. przy użyciu ADO.NET. Modyfikowanie buforów plików jest możliwe, ale jest to działanie, które absolutnie odradzamy – może prowadzić do powstania trudnych do wykrycia i usunięcia błędów w bazie danych.

3.3 Identyfikator błędu w przypadku zdarzeń plikowych

W przypadku zdarzeń plikowych istnieje możliwość zareagowania na ewentualny błąd np. nieudaną aktualizację rekordu. W tym celu obok standardowej subskrypcji AddFileActionSubscription jest udostępniona funkcja AddFileActionErrSubscription. W ramach tej subskrypcji jest dodany delegat FileActionErrDelegate, do którego będzie przekazywany kod błędu. Delegat jest wołany tylko na zdarzeniach typu after.

Oba typy subskrypcji mogą działać równocześnie. Wówczas Hydra najpierw wywołuje delegata z przekazanym kodem błędu, a później stary delegat.

W materiałach dodatkowych zostały udostępnione callbacki przedstawiające wykorzystanie zdarzeń plikowych. Są to odpowiednio callback_plikowy.cs (dla AddFileActionSubscription) i callback_plikowy_err.cs (dla AddFileActionErrSubscription).

3.4 Obsługa list z poziomu Hydry

W przypadku tworzenia nowych list za pomocą Hydry, pierwszym etapem jest ustalenie prawidłowej definicji listy. Aby prawidłowo utworzyć listę należy się posłużyć dokumentacją Clariona, gdzie szczegółowo opisano działanie poszczególnych parametrów definicji listy. Jeżeli nie dysponujemy dokładnym opisem, to można odczytać i ewentualnie zmodyfikować format innej, istniejącej już listy w programie.



Uwaga: Dla wersji SE systemu Comarch ERP XL zalecamy dokumentację Clariona w wersji 5.5, natomiast dla wersji z nowym interfejsem w wersji 8).

Kolejnym etapem jest proces wypełniania listy danymi. Aby prawidłowo wypełnić listę należy określić parametr FormatRaw wcześniej zdefiniowanym łańcuchem znaków. Stopniowa inkrementacja wartości parametru w pętli nie jest w tym przypadku zalecana, gdyż powoduje znaczne wydłużenie procesu wypełniania kontrolki. Poniższy fragment kodu prezentuje w jaki sposób powinno wyglądać prawidłowe wypełnianie listy:



Przykład:

```
string tekst = "";
```

```
// ***** Zalecana konstrukcja: *****
```

```
int y = 0;

while (y < 5000)

{

    tekst += "wartosc " + y + "|";

    y++;

}

polelist.FromRaw = tekst;
```

```
//***** Niezalecana konstrukcja: *****
```

```
int y = 0;

while (y < 5000)

{

    polelist.FromRaw += "wartosc " + y + "|";

    y++;

}

}
```

W przypadku nowej listy istnieje możliwość określenia kolorów dla każdego rekordu. Aby pojawiła się taka możliwość należy dodać znak * w definicji danej listy. Poniżej zaprezentowano fragment kodu dodającego nową listę. Dla każdej wartości został zdefiniowany zestaw 4 kolorów oznaczających kolejno: kolor tekstu, kolor tła, kolor tekstu po podświetleniu wiersza, kolor tła po podświetleniu wiersza. W przypadku pozostawienia wartości -1 zostanie pobrany domyślny kolor zdefiniowany dla listy.



Przykład:

```
polelist = zakładka.AllChildren.Add(ControlTypes.list);

polelist.Visible = true;

polelist.Bounds = new Rectangle(20, 20, 260, 260);

polelist.FormatRaw = "160L*~Naglowek~@s40@#1#";

string tekst = "";

tekst = "wartosc1|32768|-1|-1|-1|";

tekst += "wartosc2|-1|32768|-1|-1|";

tekst += "wartosc3|-1|-1|32768|-1|";
```

```
tekst += "wartosc4|-1|-1|-1|32768";
```

```
polelist.FromRaw = tekst;
```

W przypadku list istniejących nie możemy odczytać kolejki danych wysyłanych do kontrolki i w związku z tym, nie mamy możliwości zmiany koloru poszczególnych pozycji.

3.5 Zmiana wartości w kontrolce Check

Poniżej zaprezentowano przykład na zmianę wartości kontrolki typu Check.



Przykład:

```
if (polecheck.UseRaw == "0")  
{  
    Runtime.WindowController.Change(polecheck.Id, "1");  
}  
else  
{  
    Runtime.WindowController.Change(polecheck.Id, "0");  
}  
  
Runtime.WindowController.PostEvent(polecheck.Id, Events.Accepted);
```

4 Przykłady

4.1 Zdefiniowanie połączenia do bazy bez potrzeby podawania ConnectionStringa.

Nie ma potrzeby definiowania Connection Stringa przy wywoływaniu połączenia do bazy danych. W to miejsce można zastosować instrukcję:

```
Runtime.ActiveRuntime.Repository.Connection;
```

która zwróci pełny ConnectionString dla połączenia XL-owego.

4.2 Wywołanie zapytania na bazie bez potrzeby otwierania nowego połączenia do bazy.

Do klasy Config została dodana metoda ExecSQL, która pozwala na wywołanie zapytania SQL bez potrzeby otwierania nowego połączenia do bazy. Wykonanie zapytania odbywa się w tym przypadku na połączeniu XLa i służy wykonaniu instrukcji bez zwracania jej wyniku. W związku z tym jest ona przeznaczona przede wszystkim do uruchamiania procedur SQL lub instrukcji insert/update/delete na bazie danych. Na chwilę obecną nie ma możliwości wykorzystania instrukcji w celu zwrócenia wartości select.

Instrukcja ma postać:

```
Runtime.Config.ExecSql (string Sql, bool Transakcja)
```

gdzie

parametr **Sql** - zapytanie SQL

parametr **Transakcja** (true lub false) - czy zapytanie ma być wykonane w transakcji (true) lub nie (false)

4.3 Aktualna data i godzina

Informacje na temat aktualnej daty i godziny można uzyskać korzystając z klasy ConfigurationDictionary:

```
Runtime.ConfigurationDictionary.Today()
```

```
Runtime.ConfigurationDictionary.TimeStamp()
```



Uwaga: Powyższe funkcje zwracają aktualną datę ustawioną w programie.

Jeżeli data zostanie zmieniona w programie, to również zmieniona zostanie zwrócona przez funkcję.

4.4 Odczyt zaznaczonych pozycji

Na listach została udostępniona możliwość odczytu zaznaczonych pozycji. Przykład tego rozwiązania można odnaleźć w materiałach dodatkowych w pliku listazaznaczonych.cs.

4.5 Odczyt wartości atrybutu

W materiałach dodatkowych w pliku atrybuty.cs można odnaleźć przykład odczytania wartości atrybutu na dokumencie FS. Odczytanie wartości nie jest możliwe za pomocą mechanizmu Hydry, więc w tym celu zostało wykorzystane odwołanie bezpośrednio do bazy danych. Aby callback zadziałał prawidłowo należy zdefiniować atrybut z opcją automatycznego zapisu wartości w bazie.

4.6 Zmiana wartości pola tekstowego

Aby zmienić wartość istniejącego pola tekstowego należy zmienić odpowiednią właściwość danej kontrolki, a następnie odświeżyć kontrolkę.

```
ClWindow Parent = GetWindow();
```

```
ClWindow miasto = Parent.AllChildren["?Knt:Miasto"];
```

```
miasto.ScreenTextRaw = "miasto";
Runtime.WindowController.PostEvent(Parent.AllChildren["?Knt:Miasto"].Id,Events.Accepted);
```

4.7 Odświeżanie zawartości okna

Jeżeli wykonujemy operacje dla całego okna, a nie konkretnej kontrolki to w miejsce id kontrolki powinna zostać ustawiona wartość 0. Przykładowo, aby odświeżyć zawartość okna należy wywołać zdarzenie FullRefresh bez wskazania na konkretną kontrolkę:

```
Runtime.WindowController.PostEvent(0, Events.FullRefresh);
```

4.8 Podniesienie do edycji wybranego okna

Istnieje możliwość wywołania funkcji XLUruchomFormatkeWgGID bez konieczności wykorzystania w tym celu zewnętrznych funkcji API. Należy w tym celu wykorzystać wbudowaną funkcję:

```
public Hydra.GID UruchomFormatkeWgGID(short GIDTyp, int GIDFirma, int GIDNumer, short GIDLp,Hydra.Request request)
```



Przykład: Przykładowe podniesienie formatki może mieć postać:

```
GID gid = Runtime.ConfigurationDictionary.UruchomFormatkeWgGID
```

```
(gidtyp, gidfirma, gidnumer, gidlp, Hydra.Request.ViewRecord);
```

gdzie:

gidtyp, gidfirma, gidnumer, gidlp – zmienne liczbowe przekazane z aplikacji

Hydra.Request – określenie trybu podniesienia formatki. Możliwe tryby to: ChangeRecord,

DeleteRecord, InsertRecord, SelectRecord, ViewRecord

W przypadku wystąpienia błędów do pola Typ klasy GID zostaje zwrócony numer błędu zgodny z analogiczną funkcją dostępną w API, a do pozostałych pól (Firma, Numer, Lp) zostaje zwrócona wartość -1.

4.9 Ukrycie kolumny na liście

W programie Comarch ERP XL nie ma możliwości, aby ukryć lub dodać kolumnę do listy z poziomu Hydry, gdyż wiązałoby się to z koniecznością zmiany sposobu wypełniania takiej listy danymi, nad czym z poziomu Hydry nie mamy kontroli. Aby ukryć kolumnę można za pomocą Hydry zmniejszyć jej szerokość tak, aby operator nie był w stanie zobaczyć jej zawartości. Efekt taki można osiągnąć modyfikując definicję danej listy poprzez zmianę właściwości FormatRaw. Przykładowy kod można odnaleźć w załącznikach w pliku UkrywanieKolumny.cs, który przedstawia sposób na ukrycie kolumny z ceną na liście towarów.



Uwaga: Załączony przykład ukrywa kolumnę na liście jednak może okazać się konieczne dodatkowe zabezpieczenie przed możliwością rozszerzenia kolumny przez operatora.

4.10 Debugowanie kodu z poziomu Visual Studio

Istnieje możliwość debugowania stworzonego callbacka. Aby przeprowadzić ten proces z poziomu Visual Studio należy wstawić w kodzie odpowiedni breakpoint, a następnie ustawić we właściwościach projektu, aby przy starcie została uruchomiona zewnętrzna aplikacja (w naszym przypadku moduł Comarch ERP XL). Właściwości projektu można ustawić z poziomu menu Project -> Properties. Konieczne jest również zaimportowanie aktualnego callbacka do ERP XL.

Przy powyższych ustawieniach w momencie uruchamiania debugowania zostanie uruchomiony moduł wskazany w linii 'start external program', a działanie programu zostanie przerwane przy dojściu do pierwszego breakpointa.



Uwaga: Funkcjonalność wywołania zewnętrznego programu w przedstawiony sposób podczas uruchamiania debugowania nie jest dostępna w Visual Studio w wersji Express.

4.11 Reakcja na zamknięcie okna

Aby zareagować na zamknięcie formatki najczęściej wykorzystuje się zdarzenie CloseWindow. Zdarzenie to ma jednak podstawowe ograniczenie, gdyż pozwala jedynie na reakcję przed zamknięciem formatki. Aby zareagować na moment „po zamknięciu” należy zadeklarować metodę Exit. Należy również zwrócić uwagę na brak blokowania wątku podczas wywoływania komunikatu.



Przykład:

```
public override void Exit()

{

    base.Exit();

    MessageBox.Show("komunikat");

}
```

4.12 Ctrl+ins na dokumencie

Comarch ERP XL oferuje możliwość dodania nowych obiektów do bazy jak również skopiowanie istniejących za pomocą kombinacji Ctrl+ins. Aby z poziomu Hydry wykryć w jaki sposób nowy obiekt został dodany do bazy należy zapisać się na zdarzenie plikowe before InsertInit. Jeżeli dodawany jest nowy obiekt do bazy, to w gidnumerze zostanie zwrócona wartość 0. W przypadku kopiowania obiektu zostanie zwrócony gidnumer kopiowanego elementu.

4.13 Zmiana kontrahenta na dokumencie

W przypadku zmiany kontrahenta lub towaru na dokumencie dostępne są dwie ścieżki postępowania – wartość może zostać wpisana w polu z klawiatury lub wybrana z podniesionej listy. Aby obsłużyć przedstawioną sytuację należy założyć dwie subskrypcje. Jedna z nich na wypełnienie wartości z klawiatury (zdarzenie before Accepted), druga na kliknięcie przycisku (zdarzenie after Accepted).



Przykład:

```
ClaWindow Kontrahent = ParentOkno.Children["?cKop_Knt:Akronim"];

ClaWindow Kontrahent2 = ParentOkno.Children["?Kontrahent:Button"];

AddSubscription(true, Kontrahent.Id, Events.Accepted, new TakeEventDelegate(
    OnOpenWindow2));

AddSubscription(false, Kontrahent2.Id, Events.Accepted, new TakeEventDelegate(
    OnOpenWindow2));
```

4.14 Przechodzenie między oknami w trybie zaawansowanym

Jeżeli na karcie operatora ustawiono otwieranie okien w trybie zaawansowanym, to pojawia się możliwość przechodzenia między dokumentami. Aby wykryć taką operację z poziomu Hydry należy zapisać subskrypcję na kliknięcie strzałki, a następnie poczekać na pierwsze wystąpienie zdarzenia FullRefresh do okna. Po wystąpieniu tego zdarzenia należy zdjąć subskrypcję na zdarzenie FullRefresh, aby uniknąć wyzwalania callbacka przy standardowym odświeżeniu danych.



Uwaga: W przypadku niektórych list należy uwzględnić sytuację, gdy przechodzenie następuje pomiędzy różnymi dokumentami – przykładowo dokumenty korekt mają inne IdProcedury niż oryginalne dokumenty.

4.15 Wykonanie wydruku

Z poziomu Hydry możliwe jest wykonanie wydruków. W tym celu w ConfigurationDictionary udostępniono metodę WykonajWydruk().

```
public int WykonajWydruk(int ProceduralID, int Kontekst, int Zrodlo, int Wydruk, int Format, string Zmienne,  
                        string FiltrTPS, string FiltrSQL, string Sortowanie, short Urzadzenie)
```

Przykład tego rozwiązania można odnaleźć w materiałach dodatkowych w pliku WykonajWydruk.cs.

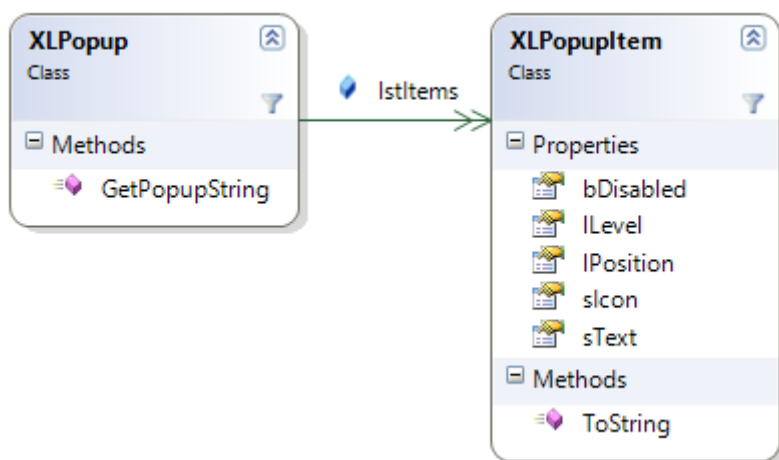
4.16 Odczyt naciśniętego klawisza

Od wersji 2014.1 jest możliwe odczytywanie za pomocą Hydry, jaki klawisz klawiatury został naciśnięty. W tym celu dodano metodę GetKeyCode do klasy WindowController.

Przykład tego rozwiązania można odnaleźć w materiałach dodatkowych w pliku OdczytKlawisza.cs.

4.17 Modyfikacja menu kontekstowego

Od wersji 2017.0 istnieje możliwość modyfikacji menu kontekstowych (tzw. PopupMenu) wyświetlanych w Comarch ERP XL. Służą do tego klasy XLPopup oraz XLPopupItem, które reprezentują odpowiednio menu oraz pojedynczą pozycję z menu.



W klasie WindowController dostępne są metody pomocnicze:

SetPopup – ustawia wygląd popup menu

SetPopupChoice – wybiera pozycję z popup menu związanym ze wskazaną kontrolką

GetPopup – zwraca definicję popup menu związanego z daną kontrolką

GetPopupChoice – zwraca numer pozycji ostatnio wybranej z popup menu

GetPopupItem – zwraca definicję pozycji ostatnio wybranej z popup menu

Aby modyfikować popup menu należy dodać subskrypcję przed zdarzeniem `Events.XLPopup` i w niej dokonać modyfikacji popup menu, natomiast na subskrypcji po tym zdarzeniu należy dodać obsługę wybranej pozycji.



Przykład:

```
public bool OpenWindow(Procedures ProclId, int ControlId, Events Event)
{
    AddSubscription(true, GetWindow().AllChildren["?ListaZam"].Id, Events.XLPopup, new
    TakeEventDelegate(BeforePopup));
    AddSubscription(false, GetWindow().AllChildren["?ListaZam"].Id, Events.XLPopup, new
    TakeEventDelegate(AfterPopup));
    return true;
}
public bool BeforePopup(Procedures ProclId, int ControlId, Events Event)
{
    XLPopup crPopup = Runtime.WindowController.GetPopup();
    //Dodanie nowej pozycji do menu na końcu
    INowaPozPos = crPopup.Items.Count;
    crPopup.Items.Add(new XLPopupItem() { sText = "-", IPosition = INowaPozPos, ILevel =
    0, slcon = "" }); //Separator
    crPopup.Items.Add(new XLPopupItem() { sText = "Nowa pozycja", IPosition =
    INowaPozPos+1, ILevel = 0, slcon = ".\\icons\\prod\\Wozek.ico" });
    //Dodanie ikony do istniejącej pozycji oraz deaktywacja usuwania
    try
    {
        crPopup.Items.Find(pi => pi.sText.Equals("Dodaj kolumny")).slcon =
        ".\\icons\\prod\\Narzedzie.ico";
        crPopup.Items.Find(pi => pi.sText.Equals("&Formaty listy")).slcon =
        ".\\icons\\prod\\Narzedzie.ico";
        crPopup.Items.Find(pi => pi.sText.Equals("Usuń")).bDisabled = true;
    } catch {}
    Runtime.WindowController.SetPopup(crPopup);
    return true;
}
public bool AfterPopup(Procedures ProclId, int ControlId, Events Event)
{
    XLPopupItem crPopuItem = Runtime.WindowController.GetPopuItem(ControlId);
    if (crPopuItem.IPosition == INowaPozPos+1)
    {
        MessageBox.Show(crPopuItem.ToString());
        return false;
    }
    return true;
}
```

Uwaga: Można dodawać i usuwać pozycje menu z dowolnego miejsca, ale jeżeli nie są to pozycje ostatnie to trzeba zadbać o przenieumerowanie wyboru użytkownika tak aby dla standardowych pozycji zwracać ich pierwotne numery.

5 Znane problemy w Hydrze

5.1 Współpraca API z oknami .net'owymi (dotyczy wyłącznie wersji SE)

Mechanizm Hydry umożliwia wywołanie okienek .net'owych co przyczynia się do znacznego zwiększenia jej możliwości, jednak oznacza to równocześnie próbę pogodzenia ze sobą dwóch różnych technologii. Dostyc często takie formatki pośredniczą pomiędzy XL-em, a funkcjami API – przykładowo zachodzi taka sytuacja gdy z poziomu programu zostaje zawołane okno .net'owe, a następnie na podstawie zebranych danych wywołujemy funkcje API. Przy klasycznej konstrukcji callbacka, gdy w ramach danego frame'a wywołujemy funkcje API dosyc często pojawiają się błędy obsługi pamięci. Jedynym rozwiązaniem jest wówczas wywołanie funkcji API poza oknem .net'owym. Najprościej uzyskać taki efekt tworząc osobną aplikację wywoływaną z poziomu linii poleceń z odpowiednimi parametrami. Dzięki temu udaje się ominąć problemy z obsługą pamięci.

5.2 Wywoływanie funkcji API z poziomu danego dokumentu

Niestety nie ma możliwości wywołania większości funkcji API z poziomu edytowanego dokumentu. Najczęściej stosuje się podobne próby w sytuacji, gdy podczas wystawiania jednego dokumentu zachodzi potrzeba wystawienia innego dokumentu. Z technicznego punktu widzenia taka sytuacja jest możliwa do obsłużenia, tzn. z poziomu jednego dokumentu możemy wywołać dodanie innego dokumentu. Najczęściej nie jest jednak wtedy możliwe prawidłowe zamknięcie pierwszego dokumentu, który pozostanie w bazie jako aktywny. Jest to spowodowane współdzieleniem przez obie sesje (XLa i API) informacji w tabeli ObiektyObce. Wówczas konieczne jest wywołanie nowego dokumentu dopiero po zamknięciu pierwszego.

5.3 Ograniczenia callbacków plikowych

Stosowanie callbackow plikowych ma swoje ograniczenia. Aby callback został wykonany niezbędne jest podniesienie danej formatki. Ma to istotne znaczenie zwłaszcza w przypadku obsługi menu kontekstowego i opcji pociągających za sobą update na elemencie. Przykładowo potwierdzenie lub anulowanie dokumentu handlowego nie spowoduje wyzwolenia callbacka zapiętego na akcję updateaction gdyż modyfikacja odbywa się bez otwierania danego okna. Inaczej natomiast wygląda sytuacja z usuwaniem takiego dokumentu z listy – w tym przypadku jest to obsłużone przez mechanizm Clariona, który podnosi okno w tle i dzięki temu callback zostanie wykonany.

5.4 Hydra i nowy interfejs

- Callbacki przygotowane dla wersji SE, nie muszą być ponownie kompilowane do wersji z nowym interfejsem.
- Jeżeli dodatek Hydry startuje nowy wątek i w tym nowym wątku występują odwołania do API bądź metod udostępnianych przez Hydrę (ogólnie odwołanie do XL-a np. przez wywołania funkcji z DLLImport), to należy w tym wątku, zaraz po jego uruchomieniu wywołać funkcję z biblioteki ClaRUN.dll AttachThreadToClarion(1).

```
[DllImport("ClaRUN.dll")]
```

```
static extern void AttachThreadToClarion(int _flag);
```

Spis ilustracji

RYS. 1 LISTA DODATKÓW	5
RYS. 2 EDYCJA DODATKU	6
RYS. 3 EDYCJA GRUPY PROCEDUR.....	7
RYS. 4 EDYCJA SUBSKRYPCJI PROCEDURY	8
RYS. 5 EDYCJA AKCJI NA KONTROLCE.....	8