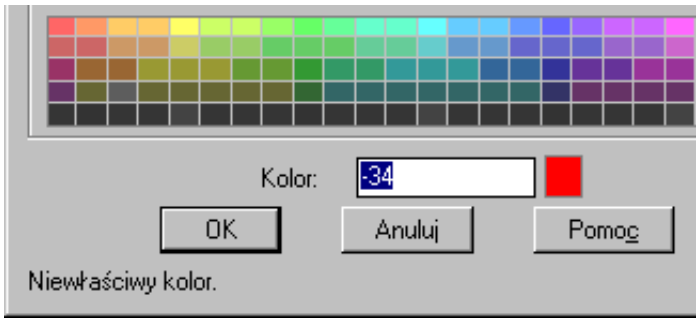
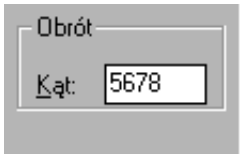
Okienka edycyjne - sprawdzanie poprawności danych

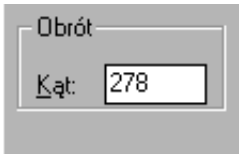
Jedną z niezaprzeczalnych zalet okien dialogowych jest łatwość wprowadzania danych potrzebnych do działania programu. Większość z nich polega na dokonaniu wyboru z listy, zaznaczeniu opcji lub wybraniu odpowiedniego przełącznika. W takich sytuacjach, programujący może stosunkowo łatwo kontrolować wprowadzane dane. Inaczej jest gdy dane przekazywane są do programu za pośrednictwem wycinka typu `edit_box`. AutoLISP nie posiada żadnych narzędzi do analizowania poprawności danych wprowadzonych do okienka edycyjnego, tak jak ma to miejsce przy wprowadzaniu danych w linii poleceń AutoCAD-a, gdzie funkcje typu `getxxx` (`getreal`, `getint` itd.) wraz z `initget` pozwalają na pobranie właściwych danych zarówno pod względem typu jak i wartości.
Okienka edycyjne są najbardziej narażone na przypadkowe lub celowe wprowadzenie niewłaściwych danych, skutkujących błędnym działaniem programu. W skrajnych przypadkach może dochodzić nawet do przerwania działania aplikacji. Dobrze napisany program, przewidujący potencjalne błędy i potrafiący im zapobiec, pozwala uniknąć takich sytuacji. W zależności od potrzeb program powinien pozwalać na wprowadzenie danych tylko takiego typu i tylko w takim formacie jaki jest dopuszczalny. Zapewnienie tego spoczywa na programującym w AutoLISP-ie.



Okienko edycyjne wyboru koloru AutoCAD-a. Dopuszczalnymi wartościami są nazwy kolorów podstawowych i logicznych (nazwy zlokalizowane), oraz liczby całkowite od 0 do 256. Liczby rzeczywiste, nie mieszczące się w zakresie lub inne łańcuchy tekstowe, oraz brak wartości powodują wyświetlenie komunikatu błędu.

Sprawdzenie poprawności danych oprócz wartości, musi uwzględniać również ich format. Powinien on odpowiadać aktualnym nastawom AutoCAD-a. Przykładowo, jeżeli w rysunku obowiązują np. cztery miejsca po przecinku dla formatu liczb rzeczywistych, dobrze napisana aplikacja będzie na bieżąco dokonywać odpowiedniego formatowania danych. I tak po wpisaniu w okienku edycyjnym dotyczącym współrzędnej punktu, liczby 10 zostanie ona być zamieniona na 10.0000.





Automatyczne formatowanie danych w okienku edycyjnym. Pomimo tego iż wprowadzona wartość kąta jako 5678 jest poprawna, wygląda nieco osobliwie i jest niejednoznaczna. Sprowadzenie tej wartości do zakresu kąta pełnego jest już bardziej zrozumiałe.

Do każdego wycinka będącego składnikiem okna dialogowego można przypisać reakcję. Uzyskuje się to przez użycie funkcji `action_tile` dla określonego wycinka, gdzie argumentem jest wyrażenie określające reakcję. Reakcja ta, to czynności jakie są podejmowane przez program w momencie wybrania przez użytkownika określonego wycinka (wtedy następuje uruchomienie `action_tile`). Dla wycinka typu `edit_box`, akcja ta podejmowana jest w momencie przejścia do innego wycinka, lub zaakceptowania wprowadzonej wartości klawiszem enter. Z reguły reakcją przypisywaną do okienek edycyjnych jest przypisanie określonej zmiennej AutoLISP-u, aktualnej wartości wycinka (uzyskiwaną przez odwołanie się do atrybutu `$value` wycinka). Wartość tej zmiennej wykorzystywana jest później przez program. Sprawdzenie poprawności danych wprowadzonych do `edit_box`, najogólniej mówiąc, polega na przypisaniu zmiennej wartości `$value` dopiero wtedy gdy spełnia ona określone warunki.

Innymi słowy, dalasze działanie pogramu zostaje wstrzymane tak długo, jak długo pewne wartości nie są poprawne. Proces sprawdzenia poprawności danych jest zatem częścią wyrażenia przypisanego do **action_tile**.

Ponieważ typ danej pobranej z wycinka **edit_box** jest zawsze łańcuchem tekstowym, w zależności od wymaganego przez program typu wprowadzonej wartości, sprawdzanie dotyczyć będzie osobno sprawdzenia poprawności łańcuchów tekstowych i liczb uzyskanych przez konwersję łańcucha tekstowego. Dane których typ może być zamiennie łańcuchem tekstowym lub liczbą (np. wartości kąta czy nazwy kolorów), wymagają napisania osobnych i specyficznych procedur sprawdzających.
Błędy wynikające z nieprawidłowo wprowadzonej wartości tekstowej, mogą mieć błahe następstwa, (np. „czeski błąd“ w opisie rysunku), lecz mogą również powodować niestabilne działanie programu (np. odwołanie do tablicy symboli przez nieistniejącą nazwę).

Łańcuchy tekstowe

W zależności od wymagań aplikacji sprawdzenie poprawności danych tekstowych dotyczyć będzie:
- możliwości wprowadzenia łańcucha pustego
- dopuszczalnej długości tekstu
- występowania spacji
- użycia małych i dużych liter
- prawidłowości znaków dla tablic symboli (nazwy warstw, bloków, stylów tekstu itp.)
- zgodności ze wzorcem
- poprawności nazw plików

W różnych sytuacjach sprawdzenie poprawności wprowadzonego tekstu może dotyczyć jednego z powyższych warunków, lub też ich kombinacji. Na przykład okienko edycyjne do wprowadzenia nazwy nowej warstwy może wymagać wprowadzenia łańcucha tekstowego gdzie dwa pierwsze znaki muszą być cyframi, trzeci znak jest podkreśleniem, a pozostała część łańcucha nie może być dłuższa niz 10 znaków. Rzecz jasna, tak utworzony łańcuch musi być zgodny z formatem wymaganym przez AutoCAD-a dla nazwy warstwy.

Dopuszczenie wprowadzenia łańcucha pustego

Programujący musi zdecydować czy brak wartości (łańcuch pusty) jest możliwy do zaakceptowania przez aplikację czy też nie. Przykładowo: jeżeli program wypełnia wartości atrybutu bloku, brak wartości można dopuścić, jednak w przypadku np. tworzenia nowego elementu tekstowego należy wymusić wprowadzenie wartości. Innym sposobem jest podstawienie jakiejś wartości domyślnej w przypadku gdy użytkownik nie podał żadnej wartości.

Długość tekstu

Aplikacja może wymagać wartości o określonej długości, lub wprowadzać górną granicę długości łańcucha. Ograniczenie długości wpisanego tekstu można określić, przez nadanie wartości atrybutu **edit_limit** (standardowo 132, maksymalnie 256 znaków). Jest to wartość stała i nie podlega zmianie podczas działania programu.

```
: edit_box { label = "&Data:";
             key = "data";
             edit_width = 10;
             edit_limit = 10;
           }
```

Wartość 10 atrybutu edit_limit wprowadza ograniczenie wpisania łańcucha tekstowego do 10 znaków. Przekroczenie zakresu nie jest możliwe, próba przekroczenia sygnalizowana jest dźwiękiem (Windows)

W sytuacjach gdy istnieje potrzeba dynamicznej zmiany ograniczenia długości łańcucha, należy ustawić **edit_limit** na maksymalną dopuszczalną (przez aplikację) wartość, i kazdorazowo sprawdzać długość łańcucha programowo. W zależności od potrzeb można odrzucić zbędne znaki, lub odrzucić dane w całości. Należy jednak powiadomić o tym fakcie użytkownika *(por. „Komunikaty błędów“).*
Może istnieć również potrzeba określenia dolnej i górnej granicy długości wprowadzonego łańcucha tekstowego. Dobrym rozwiązaniem jest zastosowanie własnych atrybutów użytkownika. Poniższy fragment pliku dcl posiada dwa własne atrybuty **mini** i **maxi**. Określają one dane potrzebne do sprawdzenia długości wprowadzonego tekstu.

```
: edit_box { label = "Podaj &hasło: ";
             password_char = "*";
             key = "text";
             edit_width = 20;
             mini = 10;
             maxi = 15;
           }
```

Atrybuty użytkownika definiują dolną i górną granicę dopuszczalnego zakresu długości wprowadzanego tekstu.

Fragment pliku lsp dokonującego sprawdzenia dopuszczalnego zakresu długości wprowadzonego łańcucha tekstowego.

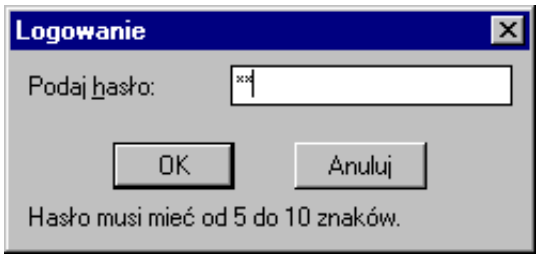
```
(setq Mini  (get_attr "text" "mini")
      Maxi  (get_attr "text" "maxi")
)
```

Odczytanie minimum i maksimum zakresu określonego w pliku dcl

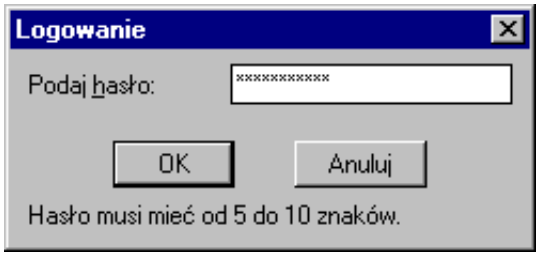
W dalszej części pliku lsp:

```
(action_tile "text"
  "(setq Test
     (if
       (or (< (strlen $value)(read Mini))
           (> (strlen $value)(read maxi))
       )
       (progn
         (set_tile \"error\"
           (strcat
             \"Hasło musi mieć od \" Mini \" do \" Maxi \" znaków.\"
           )
         )
         Nil
       )
       (progn
         (set_tile \"error\" \"\")
         $value
       )
     )
   )"
 )
```

Efekt działania można obserwować w oknie dialogowym:



Wprowadzenie mniej niż 5 znaków, powoduje wyświetlenie komunikatu błędu.



Podobnie wprowadzenie więcej niż 10 znaków:

Poprawna długość tekstu.
Uwaga:
Przedstawiony przykład ilustruje tylko sposób sprawdzenia zakresu długości wprowadzonego tekstu. Nie jest tutaj testowana poprawność wprowadzonego hasła.

Innym sposobem sprawdzenia dopuszczalnego zakresu długości wprowadzonego tekstu, jest użycie funkcji **wcmatch** *(por. „Zgodność ze wzorcem“).*

Spacje

Z różnych powodów znaki spacji w wartości przekazywanej z **edit_box** do dalszej części programu mogą być niepożądane. W przypadku ich wystąpienia, program analizujący poprawność danych, może je w całości odrzucić, lub automatycznie zamienić każde wystąpienie znaku spacji, innym poprawnym (z punktu widzenia aplikacji) znakiem. Przedstawiona poniżej funkcja AutoLISP realizuje właśnie to zadanie.

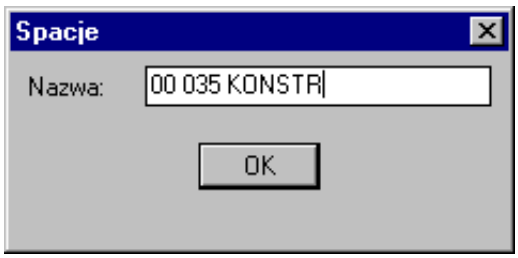
```
(defun dlg:Txt_DelSpace
  (X Char / Lst Tmp)
  (while (/= X "")
    (setq Tmp (substr X 1 1)
          X (substr X 2)
    )
    (setq Lst (append (list Tmp) Lst))
  )
  (setq Lst
    (reverse (subst Char (chr 32) Lst))
  )
  (while Lst
    (setq Tmp (car Lst)
          Lst (cdr Lst)
    )
    (setq X (strcat X Tmp))
  )
)
```

Funkcja dlg:Txt_DelSpace zastępuje każde wystąpienie spacji w łańcuchu tekstowym (argument X), znakiem (lub łańcuchem tekstowym) - (argument Char).
Funkcja zwraca zmodyfikowany tekst. Jeżeli spacje w łańcuchu nie występują, zwracany jest łańcuch bez zmian.

Poniższy fragment kodu ilustruje wykorzystanie tej funkcji w celu sprawdzenia i zmiany tekstu pobranego z **edit_box**. Po wykonaniu zmiany tekstu wyświetlany jest komunikat w wycinku **errtile**.

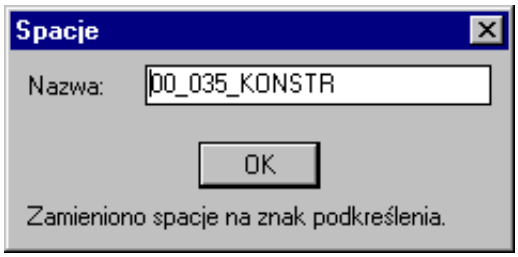
```
(setq Msg "Zamieniono spacje na znak podkreślenia.")
(action_tile "name"
        "(if
           (/= $value \"\")
           (setq Test (dlg:Txt_DelSpace $value \"_\"))
         )
         (if Test
           (if (/= Test $value)
             (progn
               (set_tile $key Test)
               (set_tile \"error\" Msg)
             )
             (set_tile \"error\" \"\")
           )
         )"
 )
```

Okno dialogowe testujace powyższą instrukcję wygląda tak:



Po wpisaniu dowolnej wartości tekstowej zawierającej spacje i akceptacji w edit_box...

... wartość zostaje zmieniona i wyświetlany jest komunikat w wycinku errtile.
Jeżeli tekst nie zawierał spacji dane są przyjmowane, bez wyświetlania komunikatu.



Oczywiście programujący musi jeszcze zdecydować czy tak sformatowany tekst nadaje się do dalszego zastosowania (przypadek gdy wprowadzono same spacje, lub czy tekst może być nazwą symbolu). W zależności od wyniku wartość może zostać zaakceptowana lub odrzucona.

Małe i duże litery

Aplikacja może wymagać aby tekst wprowadzony do **edit_box** używał w całości lub w części małych bądź dużych liter. Przykładowo np. jeżeli należy wprowadzić tekst który ma reprezentować imię i nazwisko, możnaby go automatycznie formatować, tak aby pierwsze litery wyrazów były zamieniane na duże litery pozostałe zaś na małe, bez względu na postać tekstu wprowadzonego przez użytkownika. Podobnie np. format numeru rysunku może wymagać użycia dużych liter zamiast małych. Ilość możliwych sytuacji, jest zależna od wymagań programu, programujący musi określić, czy sprawdzenie formatu pod tym kątem jest konieczne czy nie. Jeżeli istnieje potrzeba zastosowania tego typu testu, powinno się tak oprogramować zdarzenie, aby program dokonywał sam przekształcenia tekstu, zamiast informować użytkownika o niepoprawności danych. Rozważając wcześniej wspomniany przykład, automatyczne formatowanie imienia i nazwiska, będzie bardziej optymalnym rozwiązaniem, zamiast wysyłania komunikatów o błędzie. Jeżeli jest to uzasadnione, warto również rozważyć możliwość sprawdzania (i przekształcania) podanego łańcucha tekstowego, w pewnych sytuacjach, gdzie wprowadzona wartość jest wprawdzie poprawna lecz wygląda w oknie dialogowym dziwacznie (np. zamienić łańcuch „jakwaRSTWA“ na „jakwarstwa“ lub „JAKWARSTWA“). Do zamiany znaków na małe lub duże litery służy funkcja **strcase**.

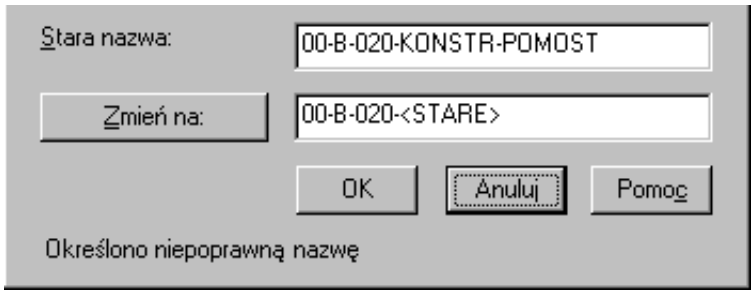
Sprawdzenie nazw tablic symboli

AutoCAD narzuca pewne ograniczenia w używaniu znaków w nazwach tablicy symboli (nazwy warstw, bloków, widoków, rodzajów linii itp.) W AutoCAD-zie 2000 jest to łańcuch o długości 255 znaków. Tekst może posiadać spacje, zabronione jest użycie następujących znaków: **< > / \ " : ? * | , = `**. We wcześniejszych wersjach (i w AutoCAD 2000 jeżeli zmienna systemowa EXTNAMES jest równa 0), wymagany łańcuch posiadał długość do 31 znaków. Mógł składać się ze znaków alfanumerycznych, oraz znaku dolara ($), podkreślenia (_), oraz myślnika (-), a niedopuszczalne było użycie znaku spacji. Jeżeli aplikacja wymaga podania takiej nazwy (w celu utworzenia lub zmiany nazwy), należy bezwzględnie sprawdzać nową wartość za pomocą funkcji AutoLISP-u **snvalid**. Funkcja zwróci **T** jeżeli nazwa jest poprawna lub w przeciwnym wypadku **nil**. Dopiero po pozytywnie przeprowadzeniu testu poprawności nazwy, można pozwolić na wykonywanie dalszej części programu.

```
(defun dlg:SNvalid (X Msg?)
  (if
    (and X (= (type X) 'STR))
    (if
      (snvalid X)
      (progn
        (if Msg? (set_tile "error" "")) X)
      (progn
        (if Msg?
          (set_tile "error" "Niepoprawna nazwa.")
        ) Nil
      )
    )
  )
)
```

Funkcja sprawdzająca łańcuch tekstowy pod względem poprawności znaków dla nazw tablic symboli.
Funkcja zwraca nazwę jeżeli jest ona poprawna w przeciwnym wypadku zwraca nil. Jeżeli argument Msg? jest różny od nil, wyświetlany jest komunikat błędu w wycinku errtile.

```
(action_tile "name"
  "(setq Test (dlg:Snvalid $value 'T))"
)
```



Fragment kodu pliku lsp. Reakcja przypisana do action_tile sprawdza poprawność nazwy w wycinku zmiany nazwy. Komunikat błędu pojawia się w wycinku errtile okna dialogowego.

W przypadku konieczności wprowadzenia nazwy istniejącego symbolu, lepszym rozwiązaniem jest zrezygnowanie z **edit_box**, utworzenie listy symboli i umożliwieniu dokonywania wyboru z listy rozwijalnej (wycinek **popup_list**) lub listy wybieralnej (wycinek **list_box**). Rozwiązanie to nie wymaga sprawdzania poprawności danych.

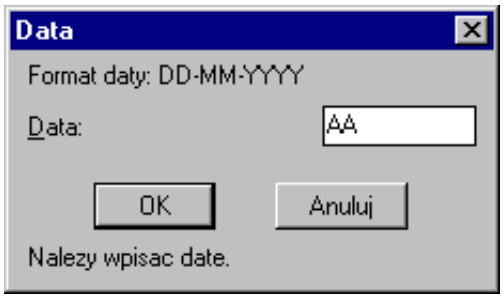
Zgodność ze wzorcem

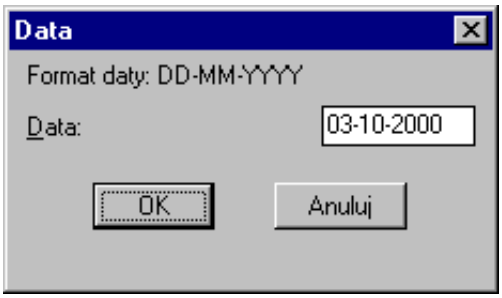
Ze względu na specyfikę programu, może istnieć potrzeba wprowadzenia danych w ściśle określonym formacie. Aplikacja może narzucać w całości lub tylko w pewnych granicach ograniczenia wprowadzanej wartości. Do porównania wartości uzyskanej z **edit_box** ze wzorcem można wykorzystać funkcję **wcmatch**. Poniżej prezentowany fragment kodu AutoLISP pozwala tylko na wprowadzenie daty w formacie DD-MM-YYYY. Każda inna wartość jest odrzucana jako niepoprawna.

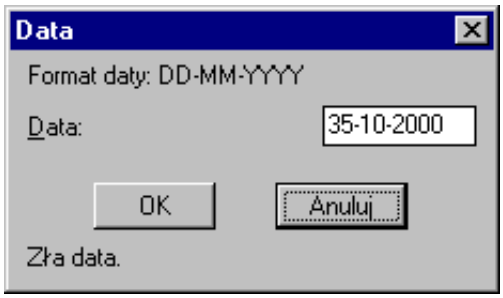
```
(defun dlg:Date? (X Msg? / dd mm yy)
  (if
    (and X (= (type X) 'STR))
    (if
      (wcmatch X „##-##-####")
      (progn
        (setq dd (substr X 1 2)
              mm (substr X 4 2)
              yy (substr X 7)
        )
        (if
          (and
            (dlg:Num_Min-Max 0 dd 1 31 Nil)
            (dlg:Num_Min-Max 0 mm 1 12 Nil)
            (dlg:Num_Min-Max 0 yy 1000 9999 Nil)
          )
          (progn
            (if Msg? (set_tile "error" ""))
            X
          )
          (progn
            (if Msg? (set_tile "error" "Zła data."))
            Nil
          )
        )
      )
      (progn
        (if Msg? (set_tile "error" "Nalezy wpisac date."))
        Nil
      )
    )
  )
```

```
      (progn
        (if Msg? (set_tile "error" "To nie jest poprawna data."))
        Nil
      )
    )
)
```

Poniżej przedstawione są przykłady okna dialogowego testującego prezentowaną funkcję:







Przykłady okna programu testującego. Powyżej: wprowadzenie łańcucha niezgodenego ze wzorcem, powoduje wyświetlenie komunikatu błędu. Przekroczenie zakresu 31 dni skutkuje również odrzuceniem danej, pomimo formatu zgodnego ze wzorcem.

Obok: poprawnie wprowadzona data.

Stosowanie funkcji **wcmatch** w większości przypadków, jest niezawodnym (i często wystarczającym) sposobem sprawdzenia łańcucha tekstowego wprowadzonego w **edit_box**. Warunki jakim mają sprostać dane zależą od indywidualnych wymagań programu.

Nazwy plików i ścieżki dostępu

Ze względu na możliwość powstawania błędów, dość skomplikowanej i rozbudowanej procedurze sprawdzającej poprawność, należy unikać wykorzystywania okienek edycyjnych do podawania nazw plików lub wskazywania ścieżek dostępu. Znacznie łatwiej i bardziej niezawodnie jest wykorzystywanie do tego celu standardowej funkcji **getfiled** wywoływanej np. przyciskiem. Przedstawienie nazwy pliku wraz ze ścieżką w wycinku tekstowym (a nie **edit_box**) zabezpiecza przed przypadkową zmianą:

W AutoCAD 2000 (i zainstalowanym ExpressTools) do wskazywania folderów (katalogów) można wykorzystać funkcję zdefiniowaną w pliku **acetutil.arx** o nazwie **acet-ui-pickdir***.
Funkcja **acet-ui-pickdir** wywoływana jest z trzema (opcjonalnymi) argumentami. Pierwszy z nich to tekst jaki pojawi się w górnej części okna, drugi to ścieżka w której rozpoczyna się poszukiwanie, a trzeci to tytuł okna dialogowego. Funkcja zwraca nazwę wskazanego folderu lub **nil**.
Uwaga
W wersji pliku **acetutil.arx** niższej niż 1.2 funkcję **acet-ui-pickdir** należy wywoływać bez argumentów! W celu sprawdzenia wersji pliku należy wywołać funkcję:

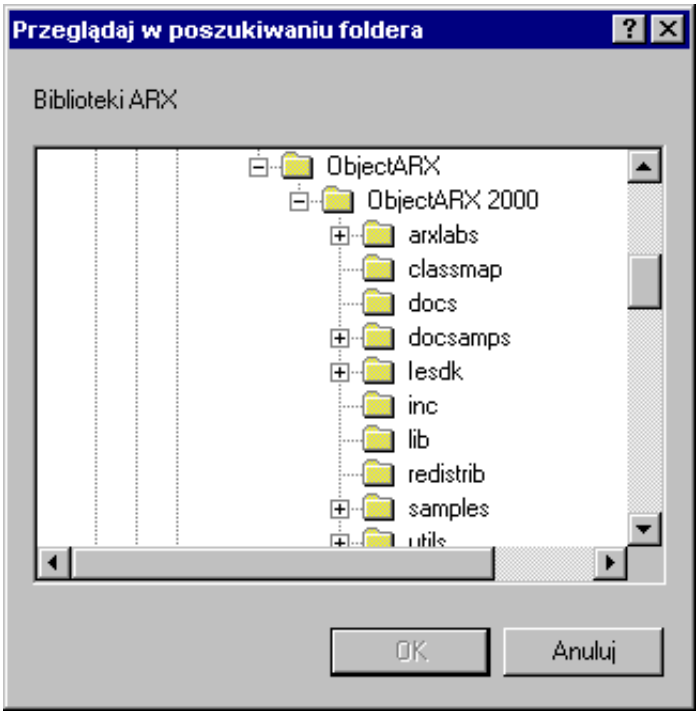
```
(acet-util-ver)
```

która zwraca liczbę rzeczywistą określającą numer wersji.
Następujące wywołanie:

```
(acet-ui-pickdir "Biblioteki ARX" "E:\\AutoCAD\\Ac2000\\Arx" "")
```

* Przed wywołaniem funkcji należy upewnić się czy aplikacja acetutil.arx jest załadowana (wywołanie funkcji (arx) zwraca listę aktualnie załadowanych aplikacji arx).

wyświetli okno dialogowe, w którym można dokonać wyboru katalogu:



Jeżeli zachodzi potrzeba wprowadzenia nazwy pliku do wycinka **edit_box**, co do którego istnieje pewność że powinien się on znajdować w bibliotecznej ścieżce poszukiwań do sprawdzenia poprawności można wykorzystać nieudokumentowaną funkcję AutoLISP-u, o nazwie **fnsplitl**. Pozwala ona w prosty sposób wyodrębnić ścieżkę dostępu do pliku i jego rozszerzenie. Poniższy fragment kodu w przypadku znalezienia pliku przypisze zmiennej **Test** listę składającą się z łańcuchów tekstowych reprezentujących ścieżkę, nazwę i rozszerzenie pliku. Gdy plik nie zostanie znaleziony (plik nie istnieje, lub podano błędną nazwę) zmienna **Test** nie ma przypisanej żadnej wartości (jest **nil**). Reakcja przypisana do przycisku zamykającego ("**accept**") pozwala na zamknięcie okna tylko w przypadku gdy **Test** jest różne od **nil** .

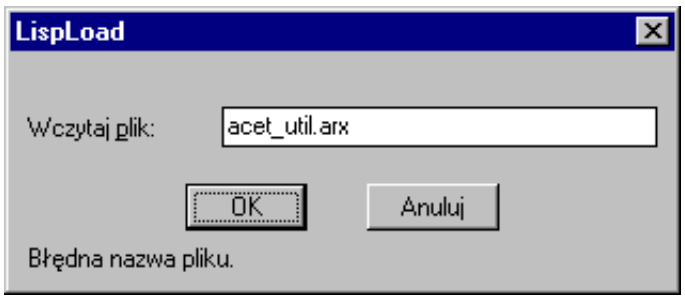
```
  ;;; reakcja przypisana do edit_box:
(action_tile "name"
  "(setq Test
     (if
       (findfile $value)(fnsplitl (findfile $value)) nil)
     )
     (if Test
       (progn
         (set_tile \"error\" \"\")
         (set_tile \"path\" (strcat \"Ścieżka: \" (car Test)))
       )
       (progn (set_tile \"error\" \"Błędna nazwa pliku.\")
              (set_tile \"path\" \"\")
       )
     )
  "
)
  ;;; Reakcja przypisana do przycisku "accept" :
(action_tile "accept"
  "(if Test
     (progn (done_dialog 1) Test)
     (set_tile \"error\" \"Należy podać nazwę pliku.\")
   )"
 )
```

Okna związane z przedstawionym kodem wyglądają następująco:

Jeżeli plik istnieje wyświetlana jest ścieżka dostępu.

Po naciśnięciu przycisku OK. zwracana jest taka lista:

```
("D:\\AutoCAD\\R15\\Express\\" "acetutil" ".arx")
```

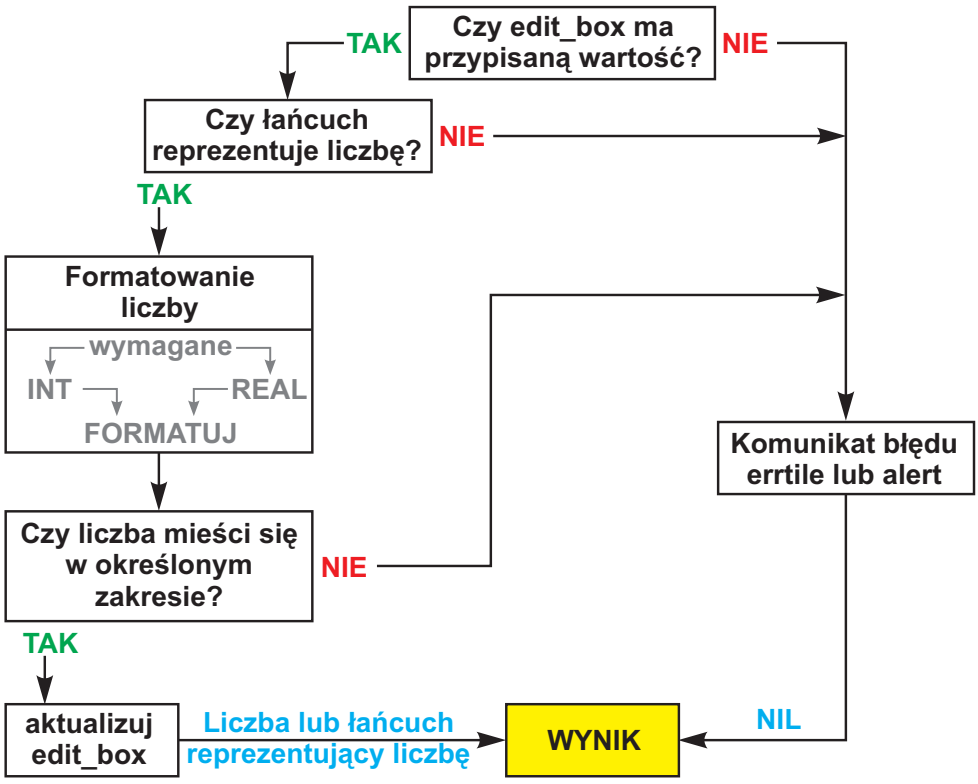


Gdy plik nie istnieje wyświetlany jest komunikat błędu, okno można zamknąć tylko przyciskiem Anuluj.

Zastosowanie **fnsplitl** pozwala w łatwy sposób wyodrębnić ścieżkę dostępu do pliku, którą można wyświetlić w oknie dialogowym. W prezentowanym przykładzie dodatkowo, akcja podejmowana przez program uzależniona jest od rozszerzenia pliku.

Liczby

Poprawność danych wprowadzonych do `edit_box` jako liczby może mieć krytyczne znaczenie dla działania całego programu. Wprowadzenie błędnej (z punktu widzenia programu) wartości, bez sprawdzenia jej poprawności, z pewnością doprowadzi do nieprawidłowego działania dalszej części programu. Dlatego też już na poziomie wprowadzania danych (w oknie dialogowym), należy zadbać o jej poprawność, oraz dokonać jej odpowiedniego formatowania. W różnych sytuacjach (w zależności od wymagań programu) sprawdzenie poprawności danej może odbywać się według takiego schematu:



Sprawdzenie bazujące na takim (lub podobnym) modelu pozwoli wprowadzić do **edit_box**, prawidłowe dane, odpowiednio sformatowane, a co za tym idzie, zapobiegające powstawaniu błędów. Procedura sprawdzajaca w zależności od potrzeb może obejmować wszystkie prezentowane kroki, lub ich część.

Sprawdzenie czy wprowadzono liczbę

Dla wycinka typu **edit_box** wymagającego wprowadzenia liczby, jest to pierwsze (oprócz sprawdzenia czy w ogóle jest wartość) sprawdzenie poprawności danych. Dopiero jego wynik określa, czy dalsze testy są wykonywane, lub czy wartość jest odrzucana. W wielu przypadkach (gdy wynik jest pozytywny), następuje już tylko odpowiednie sformatowanie danej, dalsze sprawdzanie nie jest już potrzebne. Dotyczy to np. składowych współrzędnej punktu - jeżeli jest to liczba, trzeba ją tylko odpowiednio sformatować jako liczbę rzeczywistą, sprawdzenie zakresu nie jest już konieczne. Prezentowana poniżej funkcja sprawdza czy łańcuch reprezentuje liczbę.

```
(defun dlg:Num? (X)
  (if
    (and X (= (type X) 'STR))
    (if
      (=
        (type (distof X))
        'REAL
      )
      T
      Nil
    )
  )
)
```

Funkcja sprawdza czy łańcuch tekstowy (argument X) reprezentuje liczbę (całkowitą lub rzeczywistą). Jeżeli tak zwraca T w przeciwnym wypadku zwraca nil. Funkcja ta wykorzystywana jest w innych funkcjach prezentowanych dalej.

Liczby całkowite

Jeżeli wartością wymaganą jest liczba całkowita, w większości przypadków, po pozytywnym wyniku sprawdzenia czy podano liczbę, gdy jest to liczba rzeczywista wystarczy przkształcić ją w liczbę całkowitą, przez „odcięcie“, części dziesiętnej używając funkcji **itoa** i **atoi** np.

`(itoa (atoi "10.97"))` zwraca: `"10"`

Jeżeli nie jest to specjalnie uzasadnione, trzeba zastanowić się, czy w przypadku wprowadzenia liczby rzeczywistej należy ją zaokrąglać do najbliższej liczby całkowitej. W wielu sytuacjach może być to niezrozumiałe dla użytkownika. Warto wtedy wyświetlić odpowiedni komunikat.

Liczby rzeczywiste

W przypadku wymaganych liczb rzeczywistych, po sprawdzeniu czy jest to liczba, w każdym przypadku, (liczbę całkowitą trzeba przekonwertować na rzeczywistą), należy dokonać odpowiedniego formatowania liczby zgodnie z aktualnymi nastawami AutoCAD-a. Pamiętać trzeba, że za pomijanie zer podczas konwersji liczby funkcją rtos, odpowiedzialna jest zmienna systemowa DIMZIN.

```
(defun dlg:Format_Real (X Msg? / DMZ Res)
  (if
    (dlg:Num? X)
    (progn
      (setq DMZ (getvar "DIMZIN"))
      (setvar "DIMZIN" 0)
      (setq Res (rtos (distof X)))
      (if Msg? (set_tile "error" ""))
      (setvar "DIMZIN" DMZ)
    )
    (progn
      (if Msg?
        (set_tile "error" "To nie jest liczba rzeczywista.")
```

```
      )
    )
  )
  (if Res Res)
)
```

Prezentowana powyżej funkcja dokonuje formatowania łańcucha tekstowego reprezentującego liczbę (całkowitą lub rzeczywistą) na łańcuch tekstowy reprezentujacy liczbę rzeczywistą zgodnie z aktualnymi nastawami AutoCAD-a.
Liczby rzeczywiste reprezentujące wartości kąta omówione są osobno w części „Liczby określające wartość kąta".

Określenie dolnej i górnej granicy zakresu liczb

Często samo wprowadzenie odpowiedniego typu wartości, nie wystarcza by wartość tą uznać za poprawną. Potrzebne jest również sprawdzenie czy mieści się ona w akceptowanym przez program dopuszczalnym zakresie. Przykładowo numery kolorów AutoCAD-a mogą być liczbami całkowitymi z zakresu od 0 do 256. Do sprawdzenia czy liczba mieści się w określonym zakresie można wykorzystać funkcję prezentowaną poniżej. Funkcja posiada pięć argumentów. Argument **Num** może mieć wartość 0 lub 1. Jeżeli ma wartość 0 testowana jest liczba całkowita, jeśli 1 testowana jest liczba rzeczywista. Drugi argument (**X**) jest łańcuchem tekstowym reprezentującym liczbę. Trzeci i czwarty argument to liczby określające zakres liczb. Mogą być typu **INT** lub **REAL**. Jeżeli piąty argument (**Msg?**) ma wartość różną od **nil**, wyświetlane są komunikaty błędu w wycinku **errtile**. Funkcja zwraca **T** lub **nil**. Ważne - przed wywołaniem funkcji należy upewnić się czy argument **X** jest łańcuchem reprezentującym liczbę.

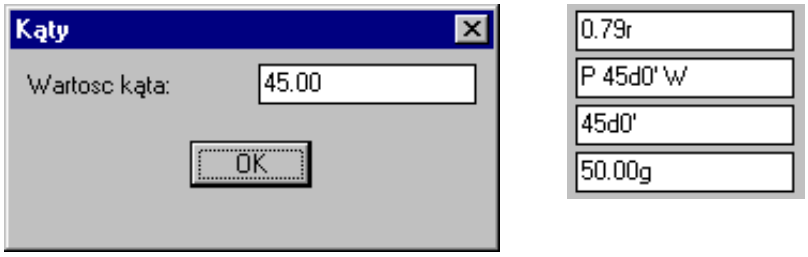
```
(defun dlg:Num_Min-Max
  (Num X Mini Maxi Msg? / Test Res)
  (if
    (= Num 0)
    (setq Test (atoi X))
    (setq Test (distof X))
  )
  (if
    (< Test Mini)
    (progn
      (setq Res Nil)
      (if Msg? (set_tile "error" "Liczba jest za mała."))
    )
    (progn
      (if
        (> Test Maxi)
        (progn
          (setq Res Nil)
          (if Msg? (set_tile "error" "Liczba jest za duza."))
        )
        (progn
          (setq Res T)
          (if Msg? (set_tile "error" ""))
        )
      )
    )
  )
  (if Res Res)
)
```

Liczby określające wartość kąta

Okienka edycyjne wymagające wprowadzenia wartości kąta, należy traktować nieco inaczej niż dotyczące liczb. Ze względu na szeroką gamę dostępnych formatów opisujacych wartości kąta w AutoCAD-zie (stopnie dziesiętne, radiany, jednostki geodezyjne itd.), aplikacja powinna być przygotowana, na przyjęcie każdego z tych formatów z aktualną dokładności z jaką wyświetlane są wartości kąta w AutoCAD-zie. Biorąc pod uwagę że wartości kąta mogą być zarówno łańcuchami tekstowymi, jak i łańcuchami tekstowymi reprezentującymi liczby, do sprawdzania poprawności wpisanego kąta należy używać funkcji konwersji dotyczących jednostek kątowych **angtos** i **angtof**. Poniżej funkcja sprawdzająca poprawność wprowadzenia wartości kąta. Łańcuch wprowadzony do **edit_box** może być wprowadzony w dowolnym formacie zapisu kąta. Po sprawdzeniu poprawności, w okienku edycyjnym wyświetlana jest wartość kąta w aktualnie obowiązujących jednostkach kąta.

```
(defun dlg:format_Angle (X Msg? / DMZ Res)
  (if
    (angtof X)
    (progn
      (if Msg? (set_tile "error" ""))
      (setq Res
        (angtos (angtof X))
      )
    )
    (if Msg?
      (set_tile "error"
        "Błąd. Zła wartosc kąta.")
      )
  )
  (if Res Res)
)
```

Jeżeli argument Msg? jest różny od nil, w przypadku wystąpienia błędu wyświetlany jest komunikat błędu.



Okno programu testującego wprowadzanie wartości kąta do edit_box. Dane akceptowane są we wszystkich formatach zapisu kąta dostępnych w AutoCAD-dzie.

W sytuacjach gdy program narzuca ograniczenia wartości kąta tylko do określonych wielokrotności pewnego kąta np. 0, 15, 30, 45 itd. jest lepiej (i łatwiej) wykorzystać do tego celu możliwość wyboru z listy (wycinek typu **popup_list**), niż sprawdzać poprawność liczby w **edit_box**. W podobny sposób można zastosować wycinek **slider**. Rozwiązania takie nie wymagają sprawdzania poprawności zwróconej wartości.
W sytuacjach gdy wymagane jest określenie dolnej i górnej granicy wartości kąta, można tego dokonać przez konwersję wartości kąta na liczbę (wartość kąta w radianach lub stopniach) i sprawdzeniu jej zakresu tak jak dla liczb.

Łańcuchy tekstowe i liczby

Niektóre okienka edycyjne mogą przyjmować dane które są łańcuchami tekstowymi lub łańcuchami tekstowymi reprezentującymi liczby. Przykładem takiego okienka edycyjnego może być pole w które można wpisać nazwę koloru (np. czerwony, zielony, jakwarstwa itd.), lub jego numer (np. 1, 40, 256). Sprawdzenie tego typu wartości musi odbywać się dwustopniowo. Na początku sprawdzany jest łańcuch czy reprezentuje liczbę (tutaj całkowitą). Jeżeli tak, należy spawdzić czy liczba nie jest mniejsza od zera i nie większa od 256. W wypadku gdy wartość sprawdzana nie jest liczbą, trzeba sprawdzić czy dany łańcuch znajduje się na liście kolorów. Gdy żaden z warunków nie jest spełniony, wpisana wartość jest

błędna i zostaje odrzucana. Jako przykład, poniżej funkcja realizująca takie sprawdzenie. Funkcja sprawdza łańcuch tekstowy czy jest on poprawną nazwą lub numerem koloru AutoCAD-a. Zwraca nazwę lub numer koloru, w przypadku powodzenia lub `nil`, gdy wartość jest niepoprawna. Jeżeli argument `Msg?` jest różny od `nil`, wyświetlany jest komunikat błędu w wycinku `errtile`. Dla kolorów nazwanych funkcja zwraca zawsze nazwę koloru (np. wpisano "`1`" funkcja zwaca "`czerwony`").

```
(defun dlg:Color? (X Msg? / Test)
  (setq Test
  (if X
    (progn
      (cond
        ( (= (type (read X)) 'REAL)
          (if Msg? (set_tile "error" "Niewłaściwy kolor."))
          Nil
        )
        ( (= (type (read X)) 'INT)
        (if
          (not (dlg:Num_Min-Max 0 X 1 256 Nil))
          (progn
            (if Msg? (set_tile "error"
                               "Kolor może być liczbą od 0 do 256."
                     )
            )
            nil
          )
          (progn (if Msg? (set_tile "error" "")) X)
        )
      )
      ('t
        (if
          (not
            (member (strcase X)
                    '("CZERWONY" "ŻÓŁTY" "ZIELONY"
                      "BŁĘKITNY" "NIEBIESKI" "FIOLETOWY"
                      "BIAŁY" "JAKWARSTA" "JAKBLOK"
                    )
            )
          )
          (progn
            (if Msg? (set_tile "error" "Niewłaściwy kolor."))
            Nil
          )
          (progn (if Msg? (set_tile "error" "")) X)
        )
      )
    )
  )
  Nil
  ))
  (if Test (dlg:Color->Str X) Nil)
)
```

Formatowania nazwy lub numeru koloru dokonuje funkcja:

```
(defun dlg:Color->Str (Color / Res)
  (setq Res
    (cdr
      (assoc
        (strcase Color)
```

```
            (list '("CZERWONY" . "czerwony")
                  '("ŻÓŁTY" . "żółty")
                  '("ZIELONY" . "zielony")
                  '("BŁĘKITNY" . "błękitny")
                  '("NIEBIESKI" . "niebieski")
                  '("FIOLETOWY" . "fioletowy")
                  '("BIAŁY" . "biały")
                  '("JAKWARSTA" . "JakWarstwa")
                  '("JAKBLOK" . "JakBlok")
                  '("1" . "czerwony")
                  '("2" . "żółty")
                  '("3" . "zielony")
                  '("4" . "błękitny")
                  '("5" . "niebieski")
                  '("6" . "fioletowy")
                  '("7" . "biały")
                  '("256" . "JakWarstwa")
                  '("0" . "JakBlok")
            )
          )
        )
      )
      (if (not Res) Color Res)
)
```

Uwaga: Przedstawione funkcje dokonują formatowania nazwy koloru w polskiej wersji językowej AutoCAD-a.

Reakcja przypisana do przycisku zamykającego

Podczas aktywności okna dialogowego użytkownik przejmuję pełną kontrolę nad dalszym wykonywaniem programu. W tym czasie może wprowadzać dane, zmieniać parametry programu itp. Przyciski zamykające okno dialogowe kończą ten proces, i w zależności od typu wybranego przycisku zamykającego (zwykle OK i Anuluj) wykonują odpowiednie operacje[*] w rysunku. Dokonując sprawdzenia poprawności danych wprowadzanych do **edit_box**, należy uwzględnić, reakcje przycisków zamykających okno dialogowe, w przypadku wprowadzenia niewłaściwych danych. Jest to konieczne ze względu na możliwość zignorowania, przez użytkownika, komunikatu o błędzie generowanego przez funkcję sprawdzajacą poprawność wprowadzonej wartości. Ostatnim elementem chroniącym program przed powstaniem poważnego błędu (spowodowanym wykorzystaniem błędnie wprowadzonej danej, lub jej braku), jest reakcja związana z przyciskiem zamykającym. Z reguły jest to niepodejmowanie żadnej akcji (oprócz wyświetlenia komunikatu błędu) przez przycisk akceptujący tak długo, dopóki nie jest wprowadzona poprawna wartość. Należy jednak pozostawić użytkownikowi możliwość opuszczenia okna dialogowego przez wybranie przycisku Anuluj.

Przycisk akceptujący

Jak to wcześniej wspomniano przycisk akceptujący nie może pozwolić na opuszczenie okna (a więc dalsze wykonywanie programu), tak długo dopóki nie są wprowadzone wszystkie wymagane dane w poprawnej formie. Programujący musi zatem tak skonstruować zdarzenie przypisane do przycisku akceptujacego, aby w wypadku braku poprawnych danych program nie podejmował żadnej akcji. Ponieważ przedstawione i omawiane tutaj funkcje sprawdzające poprawność danych są tak skonstruowane, aby zwracały **nil** w przypadku negatywnego wyniku testu, najprostszą metodą jest zastosowanie wyrażenia warunkowego **if**.
Poniższy przykład ilustruje wywołanie funkcji **done_dialog** z argumentem 1 (standardowo dla przycisku OK), w przypadku przypisania wartości do zmiennej **Test** (będącej wynikiem sprawdzenia poprawności danej w **edit_box**).

[*] Jest to standardowe zachowanie programów AutoLISP. W sytuacjach użycia metod ActiveX Automation, możliwe jest wykonywanie operacji na obiektach rysunkowych podczas aktywności okna dialogowego. Zostało to omówione w *„AutoLISP i DCL bez tajemnic 4 - DCL i ActiveX Automation“*.

```
(action_tile "accept" "(if Test (done_dialog 1))")
```

Konstrukcja taka przypisuje akcję tylko w przypadku gdy `Test` nie jest `nil`. Można ją łatwo rozbudować, dodając polecenie wyświetlenia komunikatu o błędzie, w wycinku `errtile`:

```
(action_tile "accept"
  "(if Test
     (progn (done_dialog 1) Test)
     (set_tile \"error\" \"Należy podać nazwę pliku.\")
  )"
)
```

Okno nie zostanie zamknięte przyciskiem akceptujacym dopóki uzytkownik nie poda poprawnej wartości. Podobnie należy postępować gdy w oknie dialogowym jest więcej wycinków typu `edit_box`, które były testowane na poprawność wprowadzanych danych.

```
(action_tile "accept"
  "(if
     (and Test1 Test2 Test3)
     (progn (done_dialog 1)(list Test1 Test2 Test3))
     (set_tile \"error\" \"Błąd. Złe parametry\")
  )"
)
```

W przypadku gdy wartości wprowadzone są bezbłędnie okno zostanie zamknięte, i zostanie zwrócona lista z wartościami zmiennych `Test1` `Test2` i `Test3` (potrzebna do dalszego wykonywania programu), w przeciwnym wypadku (gdy choć jedna wartość jest niepoprawna) zostanie wyświetlony komunikat błędu.
Dobrym zwyczajem jest takie oprogramowanie zdarzenia, aby użytkownik został powiadomiony, który z wycinków jest przyczyną powstania błędu (w wyżej przedstawionym przykładzie użytkownik, wie tylko że wprowadzone parametry są błędne - nie wie natomiast które). Poniższy fragment kodu, ilustruje, jeden ze sposobów wyznaczenia odpowiedniego komunikatu dla trzech wycinków. Zakładając że procedura sprawdzenia każdego wycinka zwraca `nil`, w przypadku błędnie wprowadzonej wartości, reakcja przypisana do wycinka `accept` wyświetli odpowiedni komunikat. Gdy choćby jedna zmienna jest `nil`, wykonywane jest kolejno sprawdzenie każdej zmiennej (czy ma wartość). Komunikat zostanie wyświetlony w przypadku napotkania pierwszej zmiennej która jest `nil`.

```
(action_tile "accept"
  "(if
     (and Test1 Test2 Test3)
     (progn (done_dialog 1)(list Test1 Test2 Test3))
     (if Test1
       (if Test2
         (if (not Test3)
           (set_tile \"error\" \"Błędna wartość Test3\")
         )
         (set_tile \"error\" \"Błędna wartość Test2\")
       )
       (set_tile \"error\" \"Błędna wartość Test1\")
     )
  )"
)
```

Zamiennie zamiast zagnieżdżonych funkcji `if`, można zastosować funkcję `cond`. Innym sposobem zapewnienia odpowiedniej treści wyświetlanego komunikatu błędu dla konkretnego wycinka, jest zastosowanie atrybutów użytkownika w pliku dcl. Można przypisać do każdego wycinka własny atrybut,

gdzie wartością będzie treść komunikatu błędu[*] . Odczytać go można używając funkcji **get_attr**, a wartość przekazać do **errtile**, w momencie powstania błędu (lub wybrania przycisku OK).

Przycisk anulujący

Jeżeli nie ma specjalnych powodów aby nie stosować przycisku anulującego, powinien się on znaleźć w każdym oknie dialogowym w którym dokonuje się ustawień programu. Zawsze należy umożliwić użytkownikowi opuszczenie okna bez wykonania dalszej części programu. W takich sytuacjach nie powinno się zapamiętywać ustawień dokonanych przez użytkownika w oknie dialogowym. Z reguły jedyną reakcją przypisywaną do przycisku jest operacja zamknięcia okna, natomiast czynności związane z zakończeniem czy przerwaniem działania programu, są wywoływane osobno (jako skutek anulowania okna). W szczególnych sytuacjach, gdy okno dialogowe jest wyświetlane automatycznie, jako kolejna część, jakiegoś procesu, należy zastanowić się czy zamknięcie okna przyciskiem Aunluj, ma na celu rezygnację z dokonywania ustawień w oknie, czy też anulowanie całości polecenia. W takich sytuacjach warto rozważyć możliwość wywołania okna potwierdzającego (zob. Okna ostrzegawcze).

Komunikaty błędów

Końcowym elementem procesu sprawdzenia poprawności wartości wprowadzanych do **edit_box**, jest przekazanie użytkownikowi informacji, w wypadku negatywnego wyniku testu. Użytkownik powinien mieć możliwość tylko takiego wprowadzenia danych jaki jest akceptowany przez program, zatem każde uchybienie powinno być zasygnalizowane. Informacja taka pozwoli użytkownikowi szybko, skorygować błędnie wprowadzoną wartość. W większości przypadków wystarczy wyświetlenie komunikatu w wycinku **errtile**, w innych sytuacjach można stosować różnego rodzaju okna ostrzegawcze i informacyjne.

Wycinek errtile

Natychmistowo po wrowadzeniu wartości do **edit_box**, powinna być wywoływana procedura sprawdzająca poprawność danych. W każdym przypadku gdy wartość nie spełnia określonych wymagań, należy powiadomić o tym fakcie użytkownika. Najlepszym sposobem jest stosowanie, we wszystkich oknach w którym istnieje prawdopodobieństwo wystąpienia wprowadzenia błędnej wartości, standardowego wycinka **errtile**. Jest to wycinek tekstowy o długości 35 znaków (dłuższe komunikaty zostaną obcięte). Przyjęto że wycinek **errtile** umieszcza się w dolnej części, jako ostatni składnik definicji okna dialogowego. Komunikaty powinny być krótkie i zwięzłe, powinny też wyjaśniać istotę powstania błędu. Komunikat typu “Należy wpisać liczbę całkowitą” jest bardziej zrozumiały niż np. “Błędna wartość.”. Pamiętać trzeba o tym aby “wyczyścić” wycinek **errtile** po skorygowaniu błędu, lub przejściu do innego wycinka (nawet gdy błąd nie został poprawiony). Działanie takie zapewnia łączność komunikatu błędu z konkretnym wycinkiem. Pozostawienie komunikatu, przy przejściu do innego wycinka, i tam wprowadzenie poprawnej danej, powodowałoby niejednoznaczności. Omawiana wcześniej reakcja związana z przyciskiem akceptującym, i tak wyświetli komunikat, w przypadku stwierdzenia niepoprawności wprowadzonej danej, i próbie zamknięcia okna przyciskiem OK.

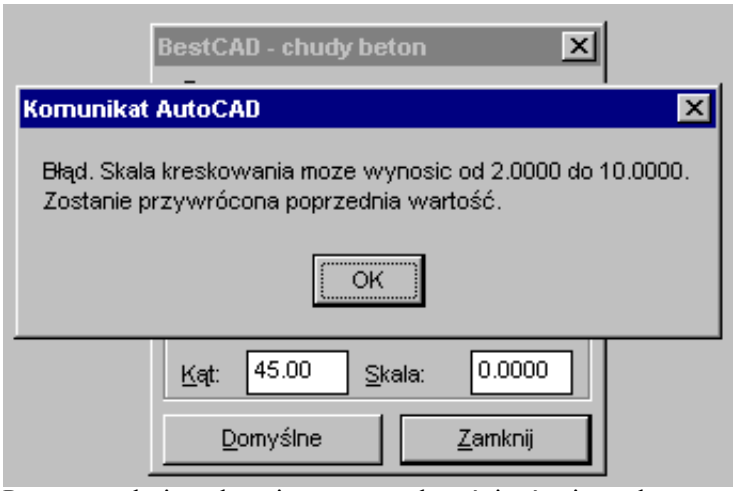
Okna ostrzegawcze[**]

W uzasadnionych przypadkach (gdy okno jest za wąskie by zmieścić wycinek **errtile**), do wyświetlenia komunikatów można wykorzystać standardową funkcję **alert**. Wyświetla ona ostrzegawcze okno dialogowe z komunikatem (łańcuch tekstowy będący argumentem funkcji **alert**). Trzeba zdawać sobie jednak sprawę, iż nadużywanie tej funkcji, często może być dla użytkownika zbyt irytujące. Lepiej jest używać **errtile**, do informowania użytkownika o wystąpieniu błędów, które można szybko skorygować, **alert** zaś używać w sytuacjach wystąpienia błędów o większym znaczeniu, lub jeśli wybranie jakiejś opcji skutkuje poważnymi następstwami.

---

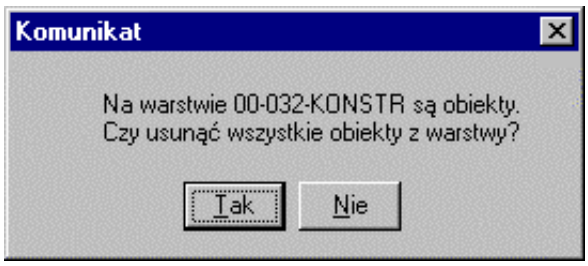
[*] Sposób użycia atrybutów użytkownika w pliku dcl, został zasygnalizowany podczas omawiania sprawdzenia dolnego i górnego zakresu liczb. Więcej o atrybutach można przeczytać w *„AutoLISP i DCL bez tajemnic 6 - Atrybuty użytkownika“*.

[**] Część ta zawiera tylko podstawowe informacje na temat zastosowań okien ostrzegawczych. Pełny opis zasad tworzenia i stosowania okien ostrzegawczych i informacyjnych w programach AutoLISP, przy wykorzystaniu DCL, VB i ARX znajduje się w *„AutoLISP i DCL bez tajemnic 3 - Okna informacyjne, ostrzeżeń i komunikatów“*.



Gdy okno dialogowe jest za małe, aby zmieścić wycinek errtile, komunikaty błędów można wyświetlać za pomocą funkcji alert.

Pewnego rodzaju nadużyciem może wydawać się również wykorzystanie funkcji **alert**, do wyświetlenia treści informujących. Nagłówek okna dialogowego "Komunikat AutoCAD-a", może wprowadzać w błąd jeżeli treść okna ma charakter informacyjny.
W sytuacjach gdy wprowadzenie wartości, może mieć nieodwracalne skutki (usunięcie lub nadpisanie pliku, przedefiniowanie bloku itp.) warto powiadomić o tym użytkownika, żądając jednocześnie możliwość powtórnego potwierdzenia, lub zrezygnowania z działania. Wyświetlenie okna informującego z dwoma przyciskami (anulującym i potwierdzającym), pozwoli użytkownikowi zaakceptować lub anulować, podjętą decyzję.



Okno ostrzegawcze, pozwalające przerwać wykonywanie programu.

Do wyświetlania różnego typu okienek ostrzegawczych czy informacyjnych można wykorzystać funkcję **acet-ui-message** z pliku **acetutil.arx**. Plik ten jest dostępny w AutoCAD-zie 2000, z zainstalowanym Express Tools.
Wywołanie funkcji ma postać:

```
(acet-ui-message message [caption [type]])
```

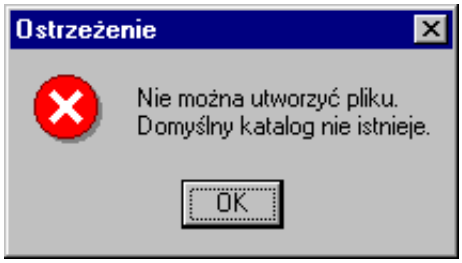
gdzie:

message - <STR> tekst komunikatu. Aby określić nową linię trzeba użyć znaku **"\n"**

caption - <STR> opcjonalnie, określa tytuł okna, domyślnie "Błąd" ("Error").

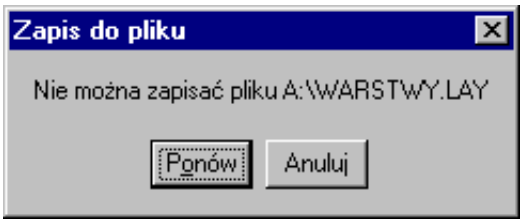
type - <INT> opcjonalnie, określa typ przycisku i/lub ikonę charakteryzującą typ okna. Domyślnie 0.

Poniżej przedstawine są przykłady wywołania funkcji **acet-ui-message** :




```
(acet-ui-message
  "Nie można utworzyć
   pliku.\nDomyślny
   katalog nie istnieje."
  "Ostrzeżenie"
  16
)
```

```
(acet-ui-message "Nie można
  zapisać pliku A:\\WARSTWY.LAY"
  "Zapis do pliku"
  5
)
```



Jak wcześniej wspomniano, podobnie jak funkcji **alert**, nie należy nadużywać **acet-ui-message**, do wyświetlania informacji które z punktu widzenia programu nie mają wielkiego znaczenia.

Wartości domyślne

Należy unikać projektowania okien gdzie użytkownik zmuszony jest do wypełniania większości lub wszystkich wycinków typu **edit_box** (wyjątkiem mogą być np. programy dokonujące edycji atrybutów bloku itp.). O ile to możliwe, okno powinno posiadać standardowe wartości, które wyświetlone w oknie dialogowym, mogą znacznie skrócić czas potrzebny do wprowadzenia danych. W wielu sytuacjach użytkownik dokonuje tylko modyfikacji pewnych wartości, inne zaproponowane przez program, pozostawia bez zmian. Ponadto wartości domyślne, sugerują typ i format danych wymaganych dla każdego z wycinków. Należy liczyć się również z tym, że użytkownik może zamknąć okno dialogowe przyciskiem akceptującym, bez modyfikowania wyświetlonych wartości.
Pierwsze (w aktualnej sesji lub rysunku) wyświetlenie okna dialogowego programu, powinno zaproponować użytkownikowi, rozsądne wartości większości (lub wszystkich) wycinków.
Wyświetlanie poprzednich wartości wycinków okna dialogowego, podczas ponownego uruchomienia programu, może znacznie przyśpieszyć wprowadzanie danych, pomimo tego iż prawdopodobne jest, że zostaną one zmienione. Sposób w jaki są one zapamiętywane zależy od programującego. Przed ich wyświetleniem powinno się je sformatować zgodnie z aktualnymi nastawami AutoCAD-a. Przykładowo: użytkownik pomiędzy kolejnymi wywołaniami okna dialogowego programu, może zmieć np. typ i format zapisu jednostek długości i kąta. Jeżeli wartości tego typu są wyświetlane w oknie dialogowym, zmiany te należy uwzględnić. Jeżeli po zmianach dokonanych przez użytkownika, okno dialogowe zostało anulowane, ponowne wyświetlenia okna nie powinno uwzględniać tych zmian.

Podsumowanie

Ilość możliwych sytuacji w których są wykorzystywane okienka edycyjne i wartości jakie mogą przyjmować jest tak duża że nie sposób omówić ich wszystkich. Sprawdzenie poprawności tych danych może być różne i indywidualne dla każdej aplikacji. Z tego powodu omówione zostały tutaj zagadnienia które uznałem za najbardziej charakterystyczne. Wiele z przykładowych rozwiązań nadaje się do użycia bez zmian, inne można dostosować do własnych potrzeb, lub na ich podstawie stworzyć nowe narzędzia analizujące poprawność danych. Stosowanie ich z pewnością pozwoli budować przyjazny i odporny na błędy graficzny interfejs użytkownika w AutoCAD-zie.
Liczę że, przedstawione tutaj uwagi, wskazówki i metody działania, ułatwią rozwiązywanie problemów związanych z kontrolowaniem danych wprowadzanych do **edit_box**, użytkownikom o różnym stopniu zaawansowania w programowaniu AutoLISP i DCL.

Jacek Kożuszek

Wszelkie uwagi można kierować:
jako@wr.onet.pl
http://www.kojacek.cad.pl [http://republika.pl./kojacek]