МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

МЕТОДИЧНІ ВКАЗІВКИ

до самостійної роботи студентів з дисципліни

“Основи САКР оптико-електронного приладобудування”

“Керування DCL діалогом у пакеті AUTOCAD засобами мови LISP ”

для студентів спеціальностей «Оптико-електронне приладобудування», «Фотоніка та оптоінформатика»

Рекомендовано Вченою Радою
приладобудівного факультету НТУУ «КПІ»
Протокол № 2/13 від 25.02. 2013р.

Розглянуто та ухвалено методичною
комісією приладобудівного факультету
НТУУ «КПІ»

Київ
НТУУ «КПІ»
2013

Методичні вказівки до самостійної роботи студентів з дисципліні «Основи САКР оптико-електронного приладобудування» «Керування DCL діалогом у пакеті AUTOCAD засобами мови LISP» для студентів спеціальностей «Оптико-електронне приладобудування», «Фотоніка та оптоінформатика». Електронне видання / Уклад.: Кравченко І.В. – К.: НТУУ “КПІ”, 2013. – 43 с.

Методичні вказівки призначено для самостійної роботи студентів спеціальностей «Оптико-електронне приладобудування», «Фотоніка та оптоінформатика» з вбудованими в систему AUTOCAD засобами організації обміну інформацією між користувачем на системою. Описано засоби програмного керування обміном даних між системою AUTOCAD та користувачем при розробці елементів САКР користувача.

Призначено для використання під час виконання курсової роботи з дисципліни. Може бути застосовано для підготовки до практичних занять, самостійного вивчення засобів автоматизації конструкторських робіт із застосуванням системи AUTOCAD.

Навчальне видання

Методичні вказівки до самостійної роботи студентів з дисципліні «Основи САКР оптико-електронного приладобудування» «Керування DCL діалогом у пакеті AUTOCAD засобами мови LISP» для студентів спеціальностей «Оптико-електронне приладобудування», «Фотоніка та оптоінформатика». Електронне видання

Укладач *Кравченко Ігор Володимирович*, старш. викладач.

Рецензент *Шевченко Вадим Володимирович* , канд. техн. наук, доцент.

За редакцією укладача

Зміст

1. Схема викликів функцій керування....5
1.1. Функції, заборонені під час дії діалогового вікна....7
2. Функції відкриття й закриття DCL-файлів....8
2.1. Функція «load_dialog»....8
2.2. Функція «unload_dialog»....9
3. Відкриття й закриття діалогових вікон....10
3.1. Функція «new_dialog»....10
3.2. Функція «start_dialog»....11
3.3. Функція «done_dialog»....12
3.4. Функція «term_dialog»....13
4. Ініціалізація виразів дії й функцій виклику з поверненням....15
4.1. Функція «action_tile»....16
4.2. Змінні дії....17
5. Ініціалізація режимів і значень....20
5.1. Функція «mode_tile»....20
6. Обробка полів і атрибутів....22
6.1. Функція «get_attr»....22
6.2. Функція «set_tile»....22
6.3. Функція «get_tile»....23
6.4. Функція «client_data_tile»....23
7. Обробка груп вибору....24
8. Функції, що працюють із списками....25
8.1. Функція «start_list»....25
8.2. Функція «add_list»....26
8.3. Функція «end_list»....27
9. Функції, що працюють із графічними полями....28
9.1. Функція «start_image»....29
9.2. Функція «end_image»....29
9.3. Функція «fill_image»....29
9.4. Функція «slide_image»....30
9.4. Функція «vector_image»....31

9.5. Функція «dimy_tile» («dimx_tile») 32
9.6. Створення зображень 32
9.7. Клавіші зображення 34
10. Обробка ковзних шкал 35
11. Поля текстового введення 36
12. Додаткові прийоми програмування діалогових вікон. 36
12.1. Тимчасовий вихід із діалогового вікна 36
12.2. Вкладені діалогові вікна 38
12.3. Дані, що обумовлені додатком 41
Рекомендована література 43

Опис діалогового вікна мовою DCL є статичною картиною форми вікна. Усі дії під час діалогу й після закриття вікна визначаються керуючою програмою, що викликає діалогові вікна, і обробляє результати діалогу. Такі керуючі програми можуть містити не тільки рядки керування діалогом, але й виконувати визначені прикладні задачі користувача. Однак більш доцільно функції керування діалоговими вікнами оформляти як окремі підпрограми, що викликаються із прикладної задачі. У цьому випадку полегшується режим налагодження діалогу й з'являється можливість використання таких функцій в інших прикладних програмах.

1. Схема викликів функцій керування

Приклад 1.1.

Припустимо, що у файлі "hello.dcl" міститься опис діалогового вікна, який наведено на рис. 1.1.

```
hello : dialog {
label = "Приклад діалогового вікна";
: text { label = "Усім привіт";}
   ok_only;}
```

Рис. 1.1. Опис діалогу приклада 1.1.

Керуюча появою цього вікна функція Автоліспа може мати наступний вигляд:

```
(defun show ( / dcld)
(setq dcld (load_gialog "hello.dcl")) ; Завантажити DCL-файл
(if (not (new_dialog "hello" dcld))(exit)) ; Ініціювати діалог,
```

вийти, якщо не працює

```
(action_tile "accept" (done_dialog)")
;Задати дію клавіші  "Так" – завершити діалог
(start_dialog )   ; Вивести діалогове вікно
(unload_dialog dcld) ;Вивантажити DCL-файл
```

Пояснення:

Після виклику функції «*start_dialog»* діалогове вікно стає активним, поки користувач не підсвітить поле (зазвичай клавішу), що зв'язано з викликом функції «*done_dialog»*. Виклик функції «*action_tile»* установлює зв'язок між полем (у наведеному прикладі з клавішею "Так" ключ якої має значення «*accept»*) і виразом дії. Саме через це виклик функції «*done_dialog»* з'являється всередині виклику функції «*action_tile»* і перед викликом функції «*start_dialog»*. Усі дії до функції «*start_dialog»* є налагоджувальними. Більш складні вікна можуть уміщувати більше число настроювань і додаткових операторів між функціями «*start_dialog»* і «*unload_dialog»*, але послідовність викликів буде подібною.

Приклад 1.1 демонструє звичайну послідовність викликів функцій, яка наведена нижче.

Рекомендована послідовність викликів функцій керування діалогом:

1. Завантаження DCL-файлу функцією «*load_dialog»*.
2. Виклик функції «*new_dialog»* для підготовки виводу діалогового вікна на графічний екран Автокада. Важливо перевіряти значення, що повертається функцією «*new_dialog»*. Виклик функції «*start_dialog»*, якщо функція «*new_dialog»* повернула помилку, може привести до непередбачених результатів.
3. Ініціалізація діалогового вікна й установка значень полів, списків

і зображень, якщо це необхідно. На цьому етапі зазвичай викликаються функції:

- «*set_tile»* і «*mode_tile»* для установки стану полів і їхніх значень;
- *«start_list»*, «*add_list»*, «*end_list»* для полів списків; «*start_image»*, «*vector_image»*, «*fill_image»,* «*slide_image»*, «*end_image»* для зображень. На цьому етапі зазвичай записується функція «*action_tile»* для установки виразу дії та для зв'язку визначених додатком даних з діалоговим вікном і його компонентами.

4. Виклик функції «*start_dialog»,* яка передає керування діалоговому вікну, у яке користувач може вводити дані.
5. Процес введення даних користувачем (виклики з поверненням). На цьому етапі використовуються функції «*get_tile»*, «*get_attr»*, «*set_tile»*, «*mode_tile»*.
6. Виклик функції «*done_dialig»* кнопкою дії.
7. Вивантаження DCL-файлу функцією «*unload_dialog»*.

1.1. Функції, заборонені під час дії діалогового вікна.

При відкритому діалоговому вікні (функція «*start_dialog»* викликана) не можна виконувати деякі функції Автоліспа. Це функції, що змінюють екран, або вимагають введення даних, які не відносяться до вікна, користувачем.

Якщо необхідно ввести дані в режимі графічного екрана (наприклад, задати точку або примітив), необхідно тимчасово закрити діалогове вікно, викликавши функцію «*done_dialog».* Графічний екран стане доступний. Після виконання вибору треба виконати повторний виклик вікна.

При відкритому діалоговому вікні заборонено застосування наступних функцій:

1. Запити і команди Автокада: *«command», «osnap».*
2. Функції введення користувачем: «*getint», «getreal», «getstring», «getpoint», «getcorner», «getdist», «getangle», «getorient», «getkword».*
3. Функції керування екраном: *«prompt», «menucmd», «redraw», «graphscr», «textscr», «textpage».*
4. Графічні функції низького рівня: *«grclear», «grdraw», «grread», «grtext», «grvacs».*
5. Функція набору вибору «*ssget».*
6. Функції керування примітивами: «*entmod», «entmake», «entdel», «entsel», «nentsel», «entupd».*

2. Функції відкриття й закриття DCL-файлів

2.1. Функція «load_dialog»

Функція завантажує DCL-файл у пам'ять.

Формат:

`(load_dialog <файл>)` , де

<*файл*> - текстовий рядок з ім'ям DCL-файлу, що завантажується. В ім'я файлу можна не уставляти розширення "dcl". Якщо ім'я файлу в аргументі зазначено без повного шляху, то система AutoCAD намагається знайти файл по стандартному алгоритмі пошуку файлів підтримки.

Значення, що повертається - ціле число (від'ємне, якщо завантаження не виконалося; позитивне - внутрішній номер завантаженого файлу в пам'яті),

використовуване в наступних викликах функцій "*new_dialog"* і "*unload_dialog".*

Приклад 2.1.

Запис `load_dialog "paktinfo.dcl")` може повернути від'ємне число у випадку помилки (файл не виявлений) або позитивне число (порядковий номер завантаженого DCL-файлу в даному сеансі роботи із системою AutoCAD)

2.2. Функція «unload_dialog»

Функція вивантажує DCL-файл із пам'яті. DCL-файли після закриття описаних у них діалогових вікон повинні вивантажуватися, звільняючи пам'ять.

Формат:

`(unload_dialog <номер>)`, де

<*номер*> - позитивне ціле число з номером завантаженого DCL-файлу (номер був повернутий попереднім викликом функції «*load_dialog»).*

Значення, що повертається — nil.

Приклад 2.2.

Запис `(unload_dialog dcl_id)` вивантажує вікно з номером, що зберігається в **dcl_id,** та повертає nil.

3. Відкриття й закриття діалогових вікон

3.1. Функція «new_dialog»

Функція виводить на екрані діалогове вікно, якщо його діалог уже завантажений у пам'ять у складі DCL-файлу функцією «*load_dialog»*. Починає керування діалоговим вікном, виводить його на екран і може визначити дію за замовчуванням.

Формат:

```
(new_dialog <ім'я вікна> <номер> [[<дія>] <точка>])
```

,де

< *ім'я вікна* > - рядок ім'ям діалогу з опису файлу діалогу.

<*номер*> - внутрішній номер вікна, що визначається викликом функції *«load_dialog».*

<*дія*> - рядок з LISP-виразом, виконуваним за замовчуванням. Якщо ніякого LISP-виразу виконувати не треба, як значення задається порожній рядок "". Дія за замовчуванням виконується після вказівки користувачем активного поля, що не має привласненого функцією *action_tile* або визначеного в DCL-файлі дії або функції виклику з поверненням. Аргумент дія, повинен бути заданий, якщо визначено аргумент <точка>.

<*точка*> - список із двох чисел, що дає в пікселах положення лівого верхнього кута діалогового вікна. Лівий верхній кут екрана має координати (0,0). Якщо задати значення (-1 -1), то вікно розташовується в центрі екрана, що є положенням за замовчуванням.

Повертається "true", якщо діалогове вікно відкрилося, і nil — якщо виникла помилка відкриття.

Слід завжди перевіряти повернуте функцією «*new_dialog*» значення. Виклик функції «*start_dialog*», якщо функція «*new_dialog*» повернула значення помилки, може привести до непередбачених наслідків.

Приклад 3.1.

Запис `(new_dialog "pakt" dcl_id "" (list 222 114))` відкриває діалогове вікно з діалогом **pakt**, та розташовує його верхній кут у точці з координатами (222,114). Повертає true

Додаток повинен викликати функцію «*new_dialog*» перед викликом «*start_dialog*». Усі установки: присвоєння полям значень, створення зображень або списків і зв'язування дій із полями (через виклики функції «*action_tile*»), - повинні відбуватися після виклику функції «*new_dialog*» і перед викликом «*start_dialog*.

3.2. Функція «start_dialog»

Активізує діалогове вікно й готує систему AutoCAD до прийому дій користувача в цьому вікні.

Формат:

```
(start_dialog)
```

Починає діалог у діалоговому вікні. Діалогове вікно повинне бути попередньо ініціалізовано попереднім викликом функції «*new_dialog*». Вікно залишається активним, поки вираз дії або функція виклику з

поверненням не викликають функцію «*done_dialog»*. Зазвичай виклик «*done_dialog»* зв'язаний із полем, що має ключ "accept" (клавіша "Так" - "ОК" або "Скасування" - "Cancel").

Функція «*start_dialog»* не має аргументів. Вона повертає аргумент <*стан*>, заданий у функції «*done_dialog».* Значенням за замовчуванням, що повертаються є:

1, якщо користувач указав клавішу Так;

0 - якщо користувач указав клавішу "Скасування";

-1, якщо вікна були закриті викликом функції «*term_dialog*».

Якщо функція «*done_dialog*» має явно заданий аргумент більше 1, функція «*start_dialog*» поверне значення, зміст якого визначається додатком.

3.3. Функція «done_dialog»

Функція завершує діалог у діалоговому вікні й прибирає його з екрана. Функція повинна викликатися з виразу дії або з функції виклику з поверненням.

Формат:

(done_dialog <стан>), де

<*стан*> не є обов'язковим. Якщо він визначений, то повинен бути позитивним цілим.

Функція повертає поточне положення (X Y) діалогового вікна.

Значення, що повертається - список із двох цілих чисел із координатами лівого верхнього кута діалогового вікна в момент його закриття. Цей список можна використовувати в наступному виклику функції *«new_dialog»* для повторного відкриття діалогового вікна в

місці, обраному користувачем.

Виклик функції "*done_dialog*" проводиться автоматично, коли у вікні діалогу застосовано стандартні кнопки виходу з вікна (OK, Cancel). Коли користувач хоче завершити роботу з діалоговим вікном самостійно виклик функції "*done_dialog*" є обов'язковим. Якщо цього не зробити, користувач не зможе вийти з діалогового вікна. Крім того, аргумент <*стан*> повинен мати в такому випадку значення 1, інакше функція (start_dialog) поверне значення за замовчуванням 0, що аналогічно скасуванню діалогового вікна.

Примітка.

Функція *«done_dialog»* використовується тільки всередині LISP виразів функції «*action_tile*» або в атрибуті «*action*».

Приклад 3.2.

Запис `(done_dialog 92)` може, наприклад, повернути список (229 160), а функція «*start_dialog»* при цьому поверне 92

3.4. Функція «term_dialog»

Функція завершує діалог у всіх активних на даний момент діалогових вікнах, як у випадку переривання користувачем за допомогою кнопки Скасування (Cancel). Функція використовується в основному для одночасного закриття вкладених діалогових вікон.

Формат:

```
(term_dialog)
```

Функція «*term_dialog»* завжди повертає nil.

Приклад 3.3.

Нижче наведено текст функції користувача, що відкриває діалогове вікно й закриває його, коли користувач натискає кнопку *ОК*.

```
(defun pinfo ( / dcid)
 (if (< (setq dcid (load dialog "info.dcl"))0)(exit))
     (if   (not   (new_dialog "p" dcid))    (exit))
     (start_dialog)
     (unload_dialog dcid)
     (princ) )
```

Функція «***pinfo***» працює з діалогом з DCL-файлу "i*nfo.dcl*". Функція має одну локальну змінну *dcid,* у якій зберігається внутрішній номер діалогового вікна, що відкривається. У другому рядку програми виконується завантаження файлу "*info.dcl*" і, якщо завантаження не виконалося (тобто перемінна *dcid* одержала від'ємне значення, яке повернуто функцією «*load_dialog»*), то виконується функція «*exit»*, що перериває роботу додатка.

У третьому рядку перевіряється, чи мається в завантаженому DCL-файлі діалог с ім'ям "p". Якщо ні, то робота програми знову завершується за допомогою функції «*exit».*

У четвертому рядку функція «*start_dialog»* відкриває діалогове вікно й чекає натискання користувачем кнопки "ОК".

У програмі немає рядка з функцією «*done_diaiog»* для закриття діалогового вікна бо кнопка "ОК" використана як ok_button. Це стандартна кнопка виходу, що і забезпечує неявне використання функції «*done_dialog»*. Наприкінці програми функція «*unload_diaiog»* вивантажує DCL-файл із пам'яті, а функція «*princ»* забезпечує тихий вихід (тобто без значення, що повертається,).

4. Ініціалізація виразів дії й функцій виклику з поверненням

Для більшої частини полів діалогового вікна, що є елементами керування, повинні бути описані дії, виконувані системою AutoCAD при виборі поля. Під вибором поля мається на увазі натискання клавіш. <Enter> або щиглик мишею для поля «*button»*, редагування вмісту для поля «*edit_box»*, вибір елемента (елементів) поля «*list_box»* або «*popup_list»*, рух шкали для поля «*slider»* і т.п.

Запрограмовані дії виконуються, коли після операції з полем (редагування, вибору елемента списку і т.п.) пристрій позиціювання перейде до іншого поля або користувач вийде з діалогового вікна за допомогою кнопки виходу.

Така дія для кожного поля може бути задане за допомогою атрибута «*action»* DCL-файлу. Значенням атрибута може бути рядок із Лісп виразом. Інший варіант завдання дії - указівка LISP виразу в рядку аргументу <*вираз*> функції *«action_tile»,* звертання до якої в Лісп програмі повинне бути записане між викликами функцій «*new_dialog»* та «*start_dialog».*

Дія, яка задана за допомогою аргументу <*вираз*>, заміщає дію, що задано значенням атрибута «*action»* цього поля в DCL-файлі.

Дія завжди задається через код поля, до якого воно відноситься. Код поля - це значення атрибута «*key»* цього поля. Саме цей код і передає функції «*action_tile»* як аргумент <*ключ*>.

LISP-вираз у функції «*action_tile»* може використовувати значення зарезервованих глобальних змінних із наступними іменами (див. табл. 1):

Таблиця 1. Змінні дії

Назва	Значення
$value	поточне значення поля
$key	код поля
$ reason	тип дії, виконаної над полем
$data	дані поля, задані функцією *«client_data_tile»* у додатку користувача
$x і $y	координати зазначеної користувачем точки усередині зображення (тільки для полів із зображеннями).

4.1. Функція «action_tile»

Задає за допомогою LISP-виразу дію, яку потрібно виконати системі AutoCAD при виборі поля або редагуванні його вмісту.

Формат:

(action__tile <ключ> <вираз>), де

<*ключ*> - рядок із кодом поля, Аргумент <ключ> представляє ім'я поля, що буде викликати дію.

<*вираз*> - рядок із LISP- виразом.

Значення, що повертається - true.

Приклад 4.1.

```
(action_tile  "accept"  "(done_dialog)")
(action_tile  "Drtl"
(strcat
"(if   (<=   (atoi  $value)   0)"
"(alert  \ "Негативні числа не допускаються! \")
)" ))
```

У прикладі 4.1. довгий LISP-вираз, що задається як дія для поля з

кодом "Drtl", отримано за допомогою зчеплення двох рядків. Внутрішні лапки, необхідні у виразі для виклику функції «*alert*», застосовуються з попереднім символом «\».

4.2. Змінні дії

Під час роботи діалогового вікна система AutoCAD формує значення змінних *$key, $value, $reason, $data, $x, $y* (вони називаються з*мінними дії),* зберігаючи в них деякі додаткові дані про стан цього вікна і виконаних у ньому діях користувача. Змінні використовуються в LISP-виразах, що використовуються функцією «*action_tile»* (**поза функцією action_tile значення змінні дії не визначені).**

Користувальницька програма не може змінювати значення цих змінних і може використовувати їх тільки для читання. Значення змінних дії наведено в табл. 3.

Таблиця 2. Значення змінної $reason.

Код	Зміст дії
1	Звичайний варіант дії з полем: користувач вибрав поле; натиснута клавіша <Enter>, а поле має значення *true* атрибута *is_default.*
2	Для поля «*edit_box»*: користувач вийшов із поля, але не завершив редагування (натиснув клавішу <Tab> або вибрав інше поле).
3	Для поля «*slider»*: користувач змінив значення поля за допомогою покажчика шкали, але не завершив уведення.
4	Для полів «*list_box»* і «*image_button»*: користувач виконав подвійного щиглика (якщо multiple_selection=false, то це повинно означати остаточний вибір, якщо multiple_selection=true, те це повинно оброблятися як звичайний вибір (див. значення 1).

Таблиця 3. Значення змінних дії

Змінна	Поле	Застосування
$key	Атрибут «*key*» обраного поля	Застосовна до всіх дій поля
$value	Рядок поточного значення поля. Для поля «*edit_box*» - уведений рядок; для «*toggle*» - "l" (включений) або "O" (виключений), для «*list__box*» і «*popup_list*» - рядок із номером обраного елемента списку	Застосовна до всіх дій. Якщо поле є полем списку (або списку, що розкривається,) і не містить жодного пункту, змінна $value буде мати значення nil
$data	Дані, що задані відразу після виклику «*new_dialog*» через виклик «*client_data_tile*» і керовані додатком	Застосовна до всіх дій. Не має значення доти, поки додаток не установить його викликом функції «*client_data_tile*»
$reason	Код причини, що повідомляє, який вибір користувача викликав дію. Використовується в полях «*edit_box*», «*list_box*», «*image_button*» і «*slider*»	Визначає причину дії (див табл.2)
$x $y	Координати X, Y вибору «*image_button*».	Представляють координати X і Y точки, у якій користувач вибрав поле «*image_button*». Для інших цілей ці перемінні не мають значення. Координата X знаходиться в інтервалі, що повертається функцією «*dimx_tile*», координата Y -в інтервалі, що повертається функцією «*dimy_tile*».

Приклад 4.2.

А) `(action_tile "fam"`

`"(setq nam (strcat \"Користувач: \" $value))"`

Для текстового поля *«fam»* вираз дії у функції *«action_tile»* обчислюється після того, як користувач залишить текстове поле. У цей момент змінна *$value* містить рядок, уведений користувачем. У змінній для подальшого використання збережеться рядок «Користувач: <дані, що уведено користувачем зі змінної *$value* >».

Б) `(action_tile "dat" "(setq typ (read $data)}")`

Для поля *«dat»* вираз дії у функції *«action_tile»* обчислюється після того, як користувач залишить поле. У цей момент у змінну *typ* буде занесено дані, які зчитує функція *«read»* зі змінної *$data* поля *«dat».*

В) `(action_tile "editl" "(setq ns Svalue)")`

Після того як користувач залишить текстове поле *«editl»* змінна *$value* буде містити рядок, уведений користувачем, і він збережеться в змінної *ns*.

Г)`(action_tile "editl" "(setq newtile $key)")`

У наведеному рядку зберігається ім'я обраного поля й, у випадку необхідності, програма може посилатися на нього. Змінна *newtile* встановлюється в значення атрибута *key* обраного поля, тобто в "*editl*".

Змінна *$key* часто використовується всередині функцій, що служать для виконання дій, привласнених декільком різним полям.

5. Ініціалізація режимів і значень

При ініціалізації діалогового вікна визначене поле може бути спочатку включене або відключено, підсвічено або не підсвічено. Ці операції виконуються за допомогою виклику функції «*mode_tile»*. Перед тим, відключити підсвічене поле, необхідно за допомогою виклику функції «*mode_tile»* підсвітити інше поле. Адже відключене підсвічене поле безглузде, що в деяких випадках може призвести до помилки.

Приклад "самовідключення" поля - послідовність "сторінок" діалогових вікон, що перегортаються вказівкою клавіші "Наступний" або "Попередній". Клавіша "Наступний" на останній "сторінці" відключається. Те ж саме відбувається із клавішею "Попередній" при переході на першу "сторінку". В обох випадках використана клавіша відключається й підсвічується інше поле.

5.1. Функція «mode_tile»

Функція встановлює режим для даного поля, заданого ключем.

Формат:

```
(mode_tile <ключ> <режим>),де
```

<*режим*> - ціле число в діапазоні 0..4 (див. табл. 4);

<*ключ*> - рядок коду поля.

При вимиканні поля його колір приглушається, і воно стає недоступним для користувача. Для того щоб це поле стало доступним, його потрібно включити.

Наприклад, `(mode_tile "k" 1)` виключає поле з кодом "k" (воно стає недоступним), `(mode_tile"k"0)` включає поле з кодом "k" (воно знову стає доступним)

Таблиця 4. Значення аргументу <режим> функції mode_tile

Значення	Дія
1	Виключити (зробити недоступним) поле
0	Уключити поле
2	Вибрати поле
3	Підсвітити вміст поля
4	Уключити/виключити підсвічування зображення

Приклад 5.1.

А)

```
setq name_str "ЛІНЗА") (set_tile "name" name_str) (mode_tile "name" 2)
```

Показаний набір функцій виконує установку значення **"ЛІНЗА"** для тексту в полі «*name»* і встановлює курсор на поле.

Додатковий виклик функції «*mode_tile»* може підсвітити вміст текстового поля:

```
(mode_tile "name" 3)
```

Б)

Припустимо, що перемикач із ключем «*group_on»* керує групою «*group»*. Коли перемикач відключений, поля групи недоступні й не можуть бути змінені. У цьому випадку дія перемикача може бути задана як:

```
(action_tile "group_on" "(mode_tile \"group\" (- 1 (atoi $value)))")
```

Вирахування й виклик функції *«atoi»* у виразі дії встановлюють значення аргументу *«mode»* функції *«mode_tile».* Значення перемикача дорівнює нулеві, коли він відключений, і одиниці, коли він включений.

Вирахування змінює значення аргументу *«mode»* і керує приступністю групи.

6. Обробка полів і атрибутів

Усередині дії часто буває необхідно одержати доступ до атрибутів DCL-файлу. Це забезпечують функції «*get_attr»* і «*get_tile»*. Функція «*get_attr»* видає значення, збережені в DCL-файлі, а «*get_tile»* видає поточні значення атрибута "*value*". Установити значення атрибута "*value*" поля можна за допомогою функції «*set_tile»*.

6.1. Функція «get_attr»

Функція зчитує початковий зміст зазначеного атрибута поля з файлу DCL.

Формат:

(get_attr <ключ> <атрибут),де

<*ключ*> - рядок коду поля ;

<*атрибут*> - ім'я атрибута поля.

Функція «*get_attr»* повертає значення атрибута як рядок. Повертається значення, що спочатку встановлене в описі поля. Зміни стану поля, що можуть відбутися після введення користувача або виклику функції «*set_tile»* НЕ ВІДБИВАЮТЬСЯ.

6.2. Функція «set_tile»

Функція змінює вміст поля, заданого ключем. Установлює діюче значення атрибута *value* для зазначеного поля. Усі аргументи є рядками.

Формат:

(set_tile <ключ> <значення>) ,де

<*код*> - код поля;

<*значення*> - рядок із новим значенням, що привласнюється атрибутові *«value»* зазначеного поля.

Значення, що повертається — значення другого аргументу, переданого функції.

Для діалогів і текстових полів значення атрибута *«value»* заміняють заголовки (значення атрибута *«label»).*

6.3. Функція «get_tile»

Функція зчитує вміст поля (поточне значення атрибута «*value»*).

Формат:

(get_tile <ключ>) ,де

<*ключ*> - рядок з кодом поля.

Значення, що повертається - рядок із поточним значенням атрибута *«value».*

6.4. Функція «client_data_tile»

Функція зв'язує з полем діалогового вікна рядок із користувальницькими даними.

Формат:

(Client data tile <ключ> <дані>), де

<*ключ*> - рядок із кодом поля;

<*дані*> - рядок із користувальницькими даними.

Значення, що повертається — nil.

Звернутися до даних, приєднаним до поля діалогового вікна, можна,

якщо прочитати значення змінної *$data* у функції *«action_tile».* У такий спосіб можна змінювати дію, що приписана полю, у залежності від користувальницьких даних.

Приклад 6.1.

Запис `(client_data_tile "usfam" "2110")` - зв'язує рядок "2110" із полем *«usfam».*

7. Обробка груп вибору

Кнопки вибору використовуються в групах вибору. Значення кожної окремої кнопки може дорівнювати одиниці, якщо вона включена, або нулеві, якщо вона відключена. Значенням групи кнопок вибору є ключ, обраної в даний момент кнопки. Проводиться перевірка включення тільки однієї кнопки. Додаток може привласнити дія кожній окремій кнопці вибору, але більш прийнятним є присвоєння дії групі кнопок вибору як єдиному цілому й використання ключа, що надсилається, для визначення, яка з кнопок обрана в даний момент.

Приклад 7.1.

Припустимо, що група кнопок вибору визначає вид тривимірного об'єкта, що треба відобразити після закриття діалогового вікна. При цьому група складається з чотирьох кнопок із ключами *"front", "top", "left", "right".* Тоді фрагмент програми для обробки дій користувача може мати наступний вигляд:

```
   . . .
(action_tile "view_set" "(pick $value)")
```

```
   . . .
(defun pick_view (which)
     (cond
       ((= which "front") (setq show_which 0))
       ((= which "top") (setq show_which 1))
       ((= which "left") (setq show_which 2))
       ((= which "right") (setq show_which 3))
     ))
```

Наведений приклад показує, що всі кнопки вибору зв'язані з однієї змінною *«pick»,* котра може мати кілька значень.

8. Функції, що працюють зі списками

Для формування списків, елементи яких відображаються в полях «*list_box»* та «*popup_box»*, застосовуються функції «*start_list», «add_list», «end_list»*.

8.1. Функція «start_list»

Функція починає операцію редагування списку в поле «*list_box»* або «*popup_list».*

(start_list <ключ> [<операція> [<номер>]]),де

<*ключ*> - код поля, список якого редагується;

<*операція*> - ціле число, що позначає тип операції:

1- змінити обрані елементи;

2- додати нові елементи;

3- видалити старий список і створити новий;

<*номер*> - номер елемента, починаючи з якого вносяться зміни. Елементи в списку нумеруються, починаючи з нуля.

Значення, що повертається — список, що буде редагуватися.

Виклик функції «*start_list*» є оцінкою початку операцій формування або редагування списку, виклик функції «*end_list*» - оцінкою закінчення.

Зміни в сам список вносяться за допомогою функції «*add_list*» і інших функцій, що розташовуються між «*start_list*» *і «end_list»*.

Не допускається між *«start_list»* і *«end_list»* використовувати функцію *«set_tile».*

Приклад 8.1.

`(start_list "pr")` - указує початок операції формування заново списку для поля з кодом "pr".

`(start_list "pr" 2)` - указує початок операції додавання нових елементів списку для поля з кодом "pr".

`(start_list "pr" 1 4)` - указує початок операції заміни елементів списку, починаючи з 5-го для поля з кодом "pr".

8.2. Функція «add_list»

Функція керує рядками списку «*list_box*» або «*popup_list*».

Формат:

`(add_list <рядок>)`,де

<рядок> -рядок, що додається до списку або заміняє елемент списку

Значення, що повертається - значення аргументу <рядок>, якщо операція у виконана успішно. При збої повертається nil.

Тип операції (додавання або заміна) визначається попереднім викликом функції start_list.

Приклад 8.2.

```
(add_list   "Деталь   3217-06")
```

8.3. Функція «end_list»

Функція завершує операцію редагування списку в поле «*list_box»* або «*popup_list».*

```
(end_list)
```

Значення, що повертається - nil.

Примітка:

Список можна змінити після його створення. Функція «mapcar» корисна для переносу списку з Автоліспа у відображуване поле списку:

```
(startlist "ключ" <операція>)
 (mapcar 'addlist newnames)
 (endlist)
```

При обробці списків значення поля «*list_box»* може містити початкові пробіли, особливо якщо обрано кілька пунктів, що утруднює перевірку за допомогою функцій порівняння рядків. Спочатку варто перетворити значення до цілого, використовуючи функції *«atoi»* або *«read».*

Приклад 8.3.

Передбачається, що список "mylist" дозволяє вибрати тільки один пункт одночасно. Наведений нижче фрагмент коду перевіряє, чи обраний третій пункт списку.

```
(setq index (get_tile "mylist"))
(if ( and   (/= index "") (= 2 (atoi index)))  true))
```

Оскільки функція «*atoi»* повертає 0 при порожньому рядку і коли в рядку знаходиться значення "О", спочатку варто перевірити, чи є рядок порожнім.

Значення списку, що розкривається, ніколи не починається із символу пробілу, тому додаткову перевірку в цьому випадку виконувати не обов'язково.

Списки, що розкриваються, не дозволяють виконувати множинний вибір. Якщо в поле списку можна вибрати одночасно кілька пунктів, програма повинна не тільки виконати перетворення, але й перебрати всю безліч значень у рядку.

9. Функції, що працюють із графічними полями.

Поля «*image»* і «*image_button»* є графічними полями із прямокутною зоною, що може бути заповнена завданим кольором, слайдом, малюнком. Для заповнення полів служать функції «*start_image», «end_image», «fill_image», «slide_image», «vector_image».*

9.1. Функція «start_image»

Починає процес заповнення графічного поля.

Формат:

`(start__image  <ключ>)`, де

<код> — код графічного поля (значення атрибута key).

Значення, що повертається — рядок зі значенням аргументу. При помилці повертається nil.

Не допускається між функціями «*start_image*» і «*end_image*» використовувати функцію «*set tile*».

9.2. Функція «end_image»

Функція завершує процес формування зображення графічного поля.

Формат:

```
(end_image )
```

Значення, що повертається — nil.

Виклику функції «*end_image*» повинен обов'язково передувати виклик функції *start_image*.

9.3. Функція «fill_image»

Функція заповнює прямокутник усередині графічного поля заданим кольором.

Формат:

`(fill_image <xl> <yl> <ширина> <висота> <колір>)`, де

<x1>, <y1> - координати (у пікселах) лівого верхнього кута

заповнюваного прямокутника;

<ширина>, <висота> - ширина й висота (у пікселах) прямокутника;

<колір> - ціле число з номером кольору, обраного для заповнення. Як номер кольору можна використовувати стандартні номери системи AutoCAD від 0 до 256, а також деякі від'ємні значення.

Значення, що повертається — ціле число (значення аргументу *<колір>).*

Координати графічного поля визначаються в такий спосіб. Лівий верхній кут має координати (0,0), а координати правого нижнього кута можуть бути отримані за допомогою функцій dimx_tile і dimy_tile.

Приклад 9.1.

Функція `(fill_image 5 6 20 31 1)` заповнює червоним кольором (номер 1) прямокутник із лівим верхнім кутом у точці (5,6) і нижнім правим кутом (25,37).

9.4. Функція «slide_image»

Функція заповнює графічне поле слайдом з окремого файлу слайда або бібліотеки слайдів.

Формат:

`(slide_image <x1> <y1> <ширина> <висота> <слайд>)`

, де

<x1>, <y1>, <ширина> і <висота> мають той же зміст, що і функції «*fill_image»;*

<*слайд*> - ім'я файлу слайда (розширення не вказується) або ім'я бібліотеки слайдів (ім'я слайда з бібліотеки вказується в круглих дужках).

Значення, що повертається - рядок (значення аргументу <слайд>).

Приклад 9.2.

Функція `(slide_image "prizm")` заповнює графічне поле слайдом із файлу «prizm.sld».

Функція `(slide_image "optic(lense)")` заповнює поле слайдом «lense» із бібліотеки слайдів «optic».

9.4. Функція «vector_image»

Функція малює відрізок заданого кольору всередині графічного поля діалогового вікна.

Формат:

`(vector_image <x1> <y1> <x2> <y2> <колір>)`, де

<*x1*>, <*y1*> - координати (у пікселах) початку відрізка, що малюється;

<*x2*>, <*y2*> - координати (у пікселах) кінця відрізка;

<*колір*> - ціле число з номером кольору.

Значення, що повертається - ціле число (значення аргументу *<колір>).*

Координати, використовувані функцією «*vector_image*», обчислюються так само, як у функції «*fill_image*».

Приклад 9.3.

Функція `(vector_image "igp2" 5 11 33 11 2)` - малює відрізок жовтого кольору (2) із крапки (5,11) у крапку (33,11)

9.5. Функція «dimy_tile» («dimx_tile»)

Функція обчислює вертикальний (горизонтальний) розмір графічного поля.

Формат:

`(dimy_tile <ключ>) (dimx_tile <ключ>)`, де

<*ключ*> — рядок із кодом графічного поля.

Значення, що повертається - ціле число – номер останнього піксела поля в одиницях діалогового вікна (верхній для вертикалі і лівий для горизонталі крайній піксел має номер 0).

Приклад 9.4.

Функція `(dimy_tile "igp2")` може повернути 32 (усього в такому полі по вертикалі буде 33 піксела)

9.6. Створення зображень

Послідовність викликів для створення поля зображення й клавіші зображення аналогічна послідовності обробки списку. Функція «*start_image»* починає створення зображення, функція «*end_image»* закінчує його. Визначення результату малювання здійснюється шляхом використання функцій «*vector_image», «fill_image», «slide_image».*

Приклад 9.5.

А) Зафарбувати червоним кольором поле зображення "my_image":

РОЗВ'ЯЗОК.

```
 (setq width (dimx_tile "my_image")
         height dimy_tile "my_image"))
 (start_image "my_image")
(fill_image 0 0 width height 1); 1 -  червоний
колір
(end_image)
```

Б) Намалювати навколо поля "my_image" без фарбування границю синього кольору (вектори малюються проти вартовий стрілки):

РОЗВ'ЯЗОК.

```
(setq width (dimx_tile "my_image")
      height (dimy_tile "my_image"))
(start_image "my_image")
(vector_image 0 0 0 height 2)
(vector_image 0 height width height 2)
(vector_image width height width 2)
(vector_image width O O O 2)
(end_image)
```

В) Намалювати посередині зеленому полі "my_image" вертикальну фіолетову лінію.

РОЗВ'ЯЗОК.

```
(setq width (dimx_tile "my_image")
       height (dimy_tile "my_image"))
(start_image "my_image")
(fill_image 0 0 width height 3);
(setq x (/ width 2.0)
```

```
(vector_image x 0 height 4);4- фіолетовий колір
(end_image)
```

Д) Відобразити графічне поле "my_image" зі слайдом із файлу "oep.sld".

РОЗВ'ЯЗОК.

```
(setq x (dimx_tile "my_image")
    y (dimy_tile "my_image"))
(start_image "my_image")
(slide_image 0 0  x y "oep")
(end_image)
```

Вектори слайда часто малюються білим кольором, що є кольором тла зображення за замовчуванням. Якщо слайд не видно на екрані, треба змінити атрибут «colo»r на значення «graphic_background».

9.7. Клавіші зображення

Клавішу зображення можна обробляти як і звичайну клавішу, тобто при її «натисканні» викликається відповідна дія. Однак мається можливість визначення клавіші так, що дія буде залежати від місця вказівки в зоні клавіші. Для цього програма обробки дії клавіші повинна одержати координати крапки вказівки. Координати задаються в діапазоні розміру зображення, що повертаються функціями визначення розмірів.

Приклад 9.6.

Клавіша зображення "image_set" розділена на дві прямокутні області різного кольору. Потрібно визначити, яку з областей указав користувач та повернути в змінній result рядок "Ліва половина" або "Права половина".

Для цього варто перевірити тільки один розмір:

```
(action_tile "image_set" "(pick $key $value $x)")

(defun pick (key val x)
    (setq threshold (/ (dimx_tile key) 2))
    (if (> x threshold) (setq result "Права половина") (setq result "Ліва половина"))
    )
```

10. Обробка ковзних шкал

Дія ковзної шкали перевіряє код причини. Як приклад розглянемо базову схему функції управління ковзною шкалою "my_slider". Вона викликається зі зв'язаного з нею виразу дії. Поле "slider_info" використовується цією функцією для виводу на екран у десятковій формі поточного значення стану шкали і є текстовим полем:

```
(action_tile "my_slider" "(slider_acfion $value $reason)")

(defun slider_action (val why)
    (if (or (= why 2) (= why 1)) (set_tile
    "slider_info" val))
)
```

11. Поля текстового введення

Дії, що керують полями текстового введення, аналогічні функціям обробки ковзних шкал, але при цьому значення поля введення завжди видно на екрані і немає необхідності виконувати дії, зв'язані із проміжними результатами.

```
(action_tile "myedit"(edit_action
$value $reason)

(defun edit_action (val why)
 (if (or (= why 2) (= why 1)) (set_tile "myeditbox"
val)
)
```

12. Додаткові прийоми програмування діалогових вікон.

12.1. Тимчасовий вихід із діалогового вікна

Іноді за умовами роботи додатка потрібно тимчасово залишити діалогове вікно, щоб указати на екрані точку або об'єкти засобами Автокада, виконати малювання за допомогою функції *command* , а потім знову повернутися у відкрите вікно. Така процедура виконується за допомогою функції (done_dialog) з аргументом status.

Приклад 12.1.

У наступному прикладі наведено діалог та програму, де від користувача потрібне завдання крапки введенням трьох координат на

клавіатурі у відповідних полях edit_box, або вказівка мишею. Результатом є список pt, що вміщує координати точки.

Опис вікна діалогу:

```
tempexit:   dialog   {1abe1="Завдання  крапки";
: row {
: column {
   : edit_box{ label="X: "; value="0"; key="xp"; }
   : edit_box{label="Y: "; value="0"; key="yp"; }
   : edit_box{label="Z: "; value="0"; key="zp"; } }
//кінець column
: button {1abe1="Указати <"; key="mousep";
height=6;}} //кінець row spacer_l;
ok_button;
}
```

Опис програми обробки даних:

```
(defun tempe (/  dcl_id step pt)
  (setq dcl_id (load_dialog "tempexit.dcl"))
  (setq step 2)
  (if (null pt)(setq pt (list 0.0 0.0 0.0)))
  (while   (>= step 2)
    (if (null (new_dialog "tempexit" dcl_id))(exit))
    (set_tile "xp" (rtos (car pt)   2 5))
    (set_tile "yp" (rtos   (cadr pt)  2 5))
    (set_tile "zp" (rtos   (caddr pt) 2 5))
    (action_tile  "accept" (strcat
     "(setq x (atof (get_tile \"xp\")))"
     "(setq y (atof (get_tile \"yp\")))"
```

```
    "(setq z (atof (get_tile \"zp\")))"
     "(setq pt (list x y z) ) "
     "(done_dialog 1)");strcat
    );action_tile
   (action_tile "mousep" "(done_dialog 3)")
   (setq step (start_dialog))
    (if(= step 3)(setq pt (getpoint "Задайте точку:
   "))
     ));if   while
  (unload_dialog dcl_id)
  (princ)
  )
```

Функція *tempe* використовує змінну *step*, що приймає значення 3, Якщо діалогове вікно закрите тимчасово (для вказівки точки), і 1, якщо користувач натиснув кнопку ОК. Значення перемінної step формується функцією start_diaiog, що повертається значення якої залежить від значення аргументу попереднього виклику функції *done_diaiog* (1 або 3).Що зчитуються при тимчасовому виході координати вказується користувачем крапки містяться за допомогою триразового звертання до функції *set tile* у поля *edit box* при повторному вході в діалогове вікно.

12.2. Вкладені діалогові вікна

Можлива ситуація, коли по натисканню якоїсь кнопки поточного діалогового вікна потрібно відкрити наступне вікно, з якого потім необхідно повернутися в попереднє. Тому виникає необхідність працювати із вкладеними друг у друга діалоговими вікнами.

Система AutoCAD розрахована на підтримку восьми рівнів

вкладеності діалогових вікон, але реально рекомендується використовувати не більш чотирьох рівнів (крім того, багато одночасно відкритих вікон незручно для користувача).

Програмування вкладених вікон досить просто — можна в LISP-вираз, використовуване як другий аргумент функції action_tile, увести звертання до функціям new_dialog і start_dialog. Можна порадити в таких випадках в одному DCL-файлі розмістити опису всіх діалогів (основного й вкладених), щоб звертання до функції load_dialog повнити тільки один раз і не перевантажувати LISP-вираз у функціях action_tile.

Приклад 12.2.

Нижче наведено текст файлу "twodia.dcl" та програми керування. Файл діалогу містить описи двох простих діалогів (dial і dia2). Діалог dia2 викликається з діалогу dial натисканням кнопки "**Викликати наступне вікно".**

Функція twodia відкриває перше діалогове і при натисканні кнопки "**Викликати наступне вікно"** звертається до функції opendia2, що відкриває вкладене діалогове вікно.

```
dial: dialog {1abel="Діалогове вікно першого рівня";
:text{label="приклад: ";key="tl"; }
:button {1abe1="Викликати наступне вікно";
key="b1"; }
ok_button;
}
```

```
dia2 : dialog{label="Діалогове вікно
другого рівня";
: text {1abe1="Це вкладене вікно";key="t2";
ok_only;
}

(defun twodia   (   /  dcl__id)
    (if (< (setq dcl_id (load_dialog "twodia.dcl"))
0) (exit) )
    (if(not (new_dialog "dial" dcl__id) ) (exit) )
    (action_tile "tl" "(setq mytxt $value) ")
    (action_tile "bl" "(opendia2 dcl_id) ")
    (start_dialog)
    (unload_dialog dcl_id)
     (princ)
) ; defun dial

(defun opendia2 (dc /)
    (if (not (new_dialog "dia2" dc) ) (exit) )
    (start_dialog)
) ; defun dia2
```

12.3. Дані, що обумовлені додатком

Функція *client_data-tile* привласнює полю обумовлені додатком значення змінних. Ці дані доступні через змінну дії $data. Дані додатка мають сенс тільки під час виконання додатка. Використання даних додатки схоже з використанням атрибутів. Основне розходження полягає в тому, що атрибути призначені тільки для читання, у той час як дані додатки можуть змінюватися під час виконання програми й користувач може перевірити визначений їм атрибут у DCL-файлі додатка.

Дані додатка визначені текстовими рядками.

Приклад 13.1.

Описати програму, що обробляє дані з вікна діалогу, яке вміщує поле списку "colorlist" із можливістю багаторазового вибору.

Застосування даних додатка усуває необхідність у глобальній змінній рядків списку.

Розв’язок може мати наступний вигляд:

```
    . . .
(client_data_tile
     "colorlist" "Red-Green-Blue Cyan-Magenta-Yellow
                  Hue-Saturation-Value")
    . . .
(action_tile "colorlist" "(setq usrchoice (mkjist
$value $data))")
;
(defun mklist (readnumb datalist)
  (setq readlist (strcat "(" readnumb ")"))
```

```
  (setq readlist (read readlist))
  (while (setq item (car readlist))
    (setq retlist (cons (nth item datalist) retlist))
    (setq readlist (cdr readlist))
  )
  (reverse retlist)
); end mklist
```

Функція *client_data_tile* заповнює рядки поля списку *colorlist* даними

При обранні поля *colorlist* викликається функція користувача *mklist* та привласнюється змінній *usrchoice* список, що є результатом роботи функції *mklist*. У функцію передаються список обраних рядків через змінну $value та вихідний список рядків через змінну $data.

Рекомендована література

1. Полещук Н.Н. Visual LISP и секреты адаптации Autocad.-СПб.:БХВ-Петербург, 2001.-576с
2. Полещук Н.Н. "AutoCAD: разработка приложений, настройка и адаптация" (серия "В подлиннике").- СПб.: БХВ-Петербург, 2006. -992 с.
3. Полещук Н.Н. САПР на базе Автокад - как это делается / С.А.Зуев, Н.Н.Полещук.-Спб.:БХВ-Петербург, 2004. – 1168с
4. Полещук Н.Н. "AutoCAD 2004: разработка приложений и адаптация" (серия "В подлиннике .-СПб.: БХВ-Петербург, 2004. - 624 с.
5. Полещук Н.Н. "AutoLISP и Visual LISP в среде AutoCAD" (серия "В подлиннике") /Н.Н.Полещук, П.В.Лоскутов.- СПб.: БХВ-Петербург, 2006, 960 с