# Сверхзадачи

Какие цели ставятся перед CAD-менеджерами?

Обычно в первую очередь — это вопросы установки и обеспечения нормальной работы проектировщиков, стандартизации рабочих мест, обеспечения стандартизации оформления документации. При этом крайне желательно одну и ту же работу не делать больше одного раза.

Мы не будем здесь рассматривать вопросы развертывания и установки ПО, поэтому «в сухом остатке» имеем:

- Необходимо максимально упростить собственную работу
- Необходимо обеспечить стандартизацию с минимальными затратами
- И, дополнительно, поставить пользователей в условия, в которых нарушать стандарты предприятия сложно и долго — проще им следовать.

# Цели мастер-класса

Попробуем «разбить» наши сверхзадачи на более мелкие:

1. Попробуем автоматически (т.е. без участия пользователя) настроить рабочее место;
2. Выберем некоторую последовательность действий, которую крайне желательно выполнять каждому пользователю (например, процесс очистки файла от мусора может быть достаточно легко автоматизирован);
3. Расширим процесс очистки, добавив, к примеру, проверку файла на предмет ошибок;
4. Расширим функционал стандартных команд AutoCAD;

Все перечисленное будем выполнять, используя один из доступных в AutoCAD языков программирования — AutoLISP и его расширение — VisualLISP. В конце мы поговорим о возможностях LISP'а и сравним их с технологией .NET. Естественно, попробуем «заглянуть в будущее» с точки зрения CAD-менеджера.

# История языков программирования в AutoCAD

Первый из языков программирования — AutoLISP — появился в AutoCAD в далеком 1986 году. В 1999 году Autodesk сделала огромный шаг вперед, подарив разработчикам расширение VisualLISP. Именно это расширение и подарило LISP'у возможность работы с другими приложениями, подключения к базам данных, работы с реестром Windows и массу других функций.

В 2004 году, вместе с AutoCAD 2005, Autodesk анонсировала использование технологии .NET в качестве программирования. Мы коснемся .NET в конце мастер-класса.

# Загрузка DWG в AutoCAD

Прежде чем приступать к настройке рабочего места, следует немного заняться теорией.

Пользователь работает не в AutoCAD, а в файле dwg (документе). Настройку надо проводить **до** того, как пользователь получит возможность внести в файл какие бы то ни было изменения. При загрузке документа AutoCAD выполняет некоторую последовательность шагов:

- Сначала загружаются файлы меню (*.cuix)
- Следом загружаются файлы acad.lsp, acaddoc.lsp или им подобные
- После этого выполняется загрузка файлов mnl (lisp-файлы, имеющие такое же имя, как и имя файла меню. mnl-файлы должны либо располагаться в определенных местах)
- В конце AutoCAD начинает загружать приложения из автозагрузки (StartupSuite)

Более подробно весь этот процесс рассмотрен в статьях, приведенных на сайте www.adn-cis.org.

Самое главное: вмешавшись в нужном месте, можно практически полностью перенастроить AutoCAD! Одно из самых простых решений — это редактирование файла acad*doc.lsp.

# Редактирование файла acaddoc.lsp

Прежде чем приступать к редактированию кода, давайте подумаем — что мы собираемся сделать.

1. Установим значения некоторых системных переменных (например, **blipmode**, **gridmode**, и, например, **secureload**)
2. Учитывая, что не все настройки регулируются системными переменными, выполним команду AutoCAD (к примеру, **_.viewres**)
3. Запустим AutoCAD и проверим результаты.

Откроем AutoCAD, потом вызовем редактор VisualLISP IDE (команда **vlide**) и в ней откроем файл **acad2005doc.lsp** (при установках по умолчанию он располагается в **c:\program files\autodesk\autocad 2015\support\en-us** для английской версии и **c:\program files\autodesk\autocad 2015\support\ru-ru** для русской версии). В самом конце добавим несколько строк:

```
;;; \Samples\Lsp\acad2015doc(01).lsp
;; Сначала сообщим о начале работы наших дополнений
(princ "\nНачало работы дополнительных операций")
;; Теперь установим системные переменные
(setvar "blipmode" 1)
(setvar "gridmode" 0)
(setvar "secureload" 1)
;; Сообщаем об окончании работы дополнений
(princ "\nДополнительные операции завершены")
;; Тихий выход
(princ)
```

Наш файл создан на основе стандартного шаблона acadiso.dwt, внутри которого установлены **blipmode** равная 0, а **gridmode** — 1:

```
Команда: (getvar "blipmode")
0
Команда: (getvar "gridmode")
1
```

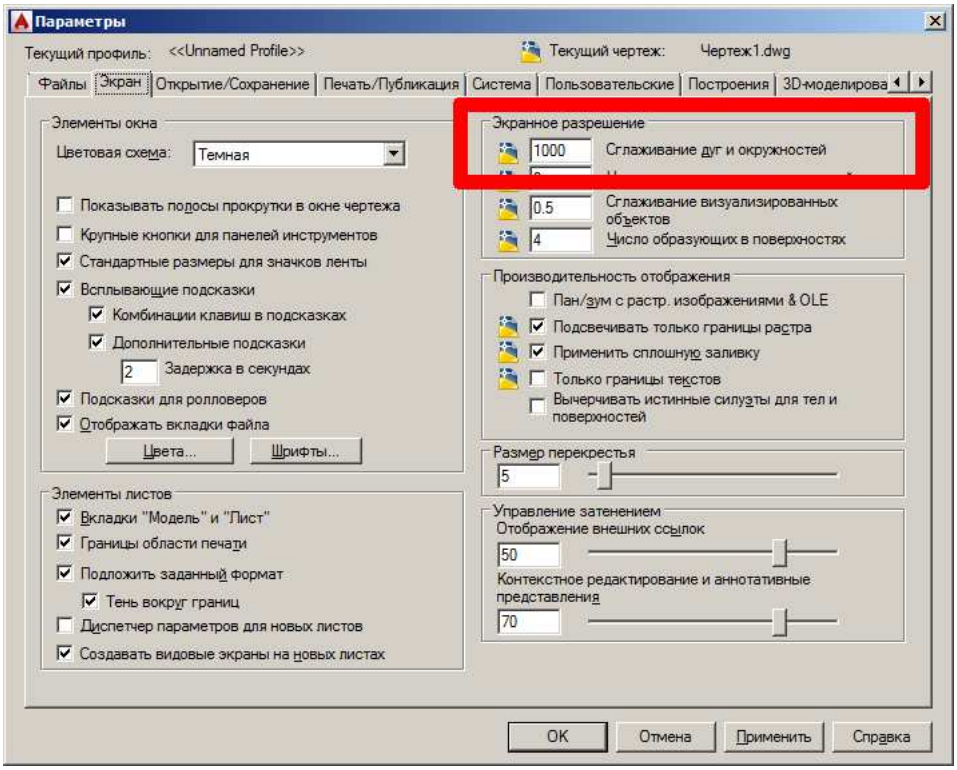
Создадим новый файл на основе того же шаблона и проверим значения этих системных переменных:

```
Команда: (getvar "blipmode")
1
Команда: (getvar "gridmode")
0
```

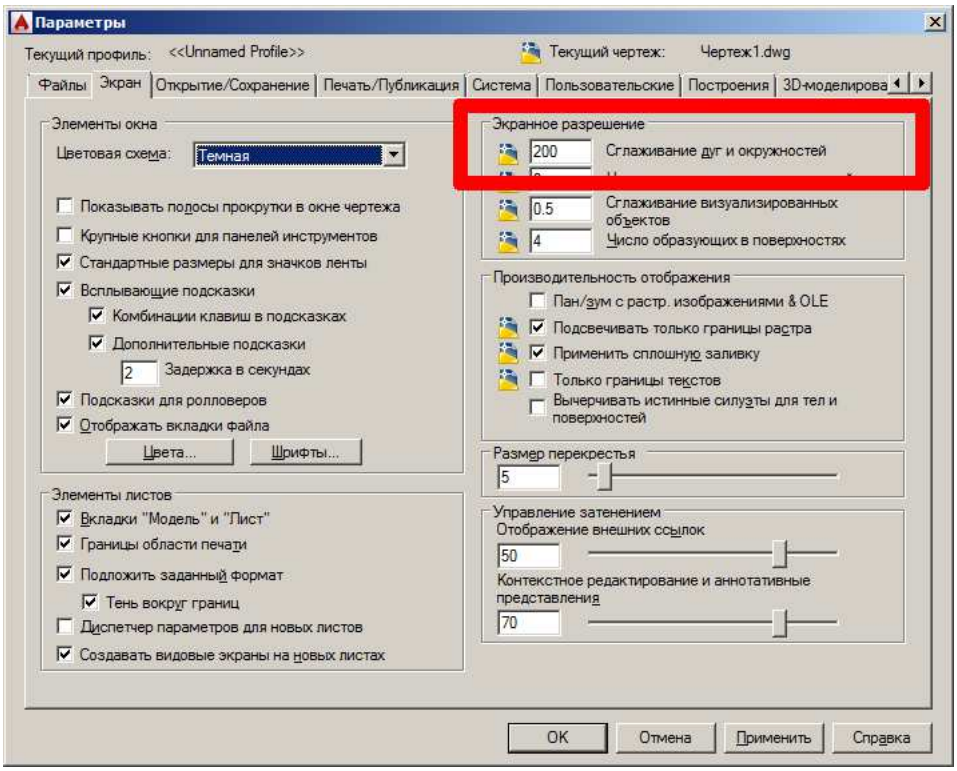
Пока оставим эту часть «как есть» и добавим изменение настройки «плавности кривых»:

```
;;; \Samples\Lsp\acad2015doc(02).lsp
;; Сначала сообщим о начале работы наших дополнений
(princ "\nНачало работы дополнительных операций")
;; Теперь установим системные переменные
(setvar "blipmode" 1)
(setvar "gridmode" 0)
(setvar "secureload" 1)
;; Добавим выполнение команд
(vl-cmdf "_.viewres" "_y" 200)
;; Сообщаем об окончании работы дополнений
(princ "\nДополнительные операции завершены")
;; Тихий выход
(princ)
```

Перед выполнением этого кода мы видим, что значение плавности равно 1000 (значение по умолчанию):




Сохраним наш файл acad2015doc.lsp. Создадим новый файл dwg и снова вызовем окно свойств:




Как видим, настройка изменилась безо всякого участия пользователя.

Уже сейчас, имея тестовую задачу, мы множество раз вносили изменения в стандартный файл AutoCAD. Это нельзя назвать идеальным решением. Причина этого достаточно проста.

Представим себе, что подобные изменения внесены в файл **`acad2015doc.lsp`** и этот файл помещен в развертывание, которое используется на предприятии. С этого развертывания AutoCAD был установлен, к примеру, на 100 компьютеров — и тут вдруг (как это обычно и бывает) потребовалось внести какие-то изменения: изменить перечень устанавливаемых переменных, или добавить какие-либо новые команды, или удалить выводимые в командной строке сообщения…

На самом деле можно сделать следующее: создать свой собственный lsp, который и вызывать из **acad2015doc.lsp**

# Создание собственного lsp

Находясь в VLIDE, создадим новый файл и внесем в него код:

```
;;; \Samples\Lsp\aur_2014-starter(01).lsp
(defun aur_2014-starter ()
  ;; Сначала сообщим о начале работы наших дополнений
  (princ "\nНачало работы дополнительных операций : отдельный
файл\n")
  ;; Теперь установим системные переменные
  (setvar "blipmode" 1)
  (setvar "gridmode" 0)
  (setvar "secureload" 1)
  ;; Добавим выполнение команд
  (vl-cmdf "_.viewres" "_y" 200)
  ;; Сообщаем об окончании работы дополнений
  (princ "\nДополнительные операции завершены : отдельный файл")
  ;; Тихий выход
  (princ)
  ) ;_ end of defun

;; Вызовем только что определенную функцию
(aur_2014-starter)
```

Включим фантазию и поставим себя в условия, приближенные к «боевым»: есть сеть, в сети есть сервер с выделенным как раз для наших целей каталогом (например, \\server\acad_users). Этот каталог подключен как сетевой диск (например, S:). Созданный только что файл сохраним, в каталог S: под именем “aur_2014-starter.lsp”. А в файле acad2015doc.lsp вместо внесенных ранее строк введем:

```
;;; \Samples\Lsp\acad2015doc(03).lsp
(load “s:\\aur_2014-starter.lsp”)
```

Если мы оставим код так, как сейчас предложено, то при настройках «по умолчанию» AutoCAD выдаст запрос: надо ли загружать такой-то файл оттуда-то? Желательно бы сделать так, чтобы этот вопрос не задавался:

```
;;; \Samples\Lsp\ acad2015doc.lsp
(defun aur_2014-load-and-start (/ secureload)
  ; Запомним старое значение SECURELOAD :
  (setq secureload (getvar "secureload"))
  ; установим SECURELOAD в 0 - в таком случае AutoCAD
  (setvar "secureload" 0)  ; никаких вопросов не задает
  (load "s:\\aur_2014-starter.lsp") ; Загрузим наш lsp
  (setvar "secureload" secureload) ; Восстановим исходное значение
SECURELOAD
  (princ) ; Тихий выход
  ) ;_ end of defun

(aur_2014-load-and-start) ; самовызов функции
```

Снова создадим новый файл и посмотрим лог выполнения в текстовом окне AutoCAD:

```
Начало работы дополнительных операций : отдельный файл_.viewres
Нужно быстрое зумирование? [Да/Нет] <Д>: _y
Точность аппроксимации (1-20000) <200>: 200
Команда:
Дополнительные операции завершены : отдельный файл
```

Внесем дополнительные изменения в свой (уже отдельный) lsp-файл. Во-первых, поместим установку системных переменных в цикл, а, во-вторых, установим еще несколько переменных:

```
;;; \Samples\Lsp\aur_2014-starter(02).lsp
(defun aur_2014-start ()
  ;; Сначала сообщим о начале работы наших дополнений
  (princ "\nНачало работы дополнительных операций : отдельный
файл")
  ;; Теперь установим системные переменные
  (foreach sysvar '(("blipmode" . 0)
                    ("gridmode" . 0)
                    ("secureload" . 1)
                    ("cmdecho" . 1)
                    ("nomutt" . 0)
                    ("isavepercent" . 0)
                    ("tooltips" . 0)
                    )
    (setvar (car sysvar) (cdr sysvar))
    ) ;_ end of foreach
  ;; Добавим выполнение команд
  (vl-cmdf "_.viewres" "_y" 200)
  ;; Сообщаем об окончании работы дополнений
  (princ "\nДополнительные операции завершены : отдельный файл")
  ;; Тихий выход
  (princ)
  ) ;_ end of defun

;; Вызовем только что определенную функцию
(aur_2014-start)
```

И теперь проверим результаты (попробуйте проверить изменения самостоятельно).

Таким образом, на данный момент мы можем абсолютно безболезненно, легко, просто и незаметно для пользователя менять настройки AutoCAD'a.

Пора переходить к следующему шагу…

# Создание собственных команд

Предлагаю создать команду, которая будет выполнять следующее:

1. Очистка файла от графического и неграфического мусора
2. Очистка файла от зарегистрированных приложений.

Эта команда должна вызываться 2-3 символами.

Откроем наш файл aur_2014-starter.lsp и добавим в него определение собственной команды:

```
(defun c:pua()
  )
```

Как известно, очистку файла от мусора выполняет команда _.purge. Но у этой команды есть еще и режим работы в командной строке: _.-purge (обратите внимание на знак «-»). Вызовем эту команду внури lsp:

```
;;; \Samples\Lsp\aur_2014-starter(03).lsp
;;; Показаны только добавленные строки
(defun c:pua ()
  (command "_.-purge" "_a" "*" "_n")
  ) ;_ end of defun
```

Сейчас выполняется только очистка от графического и неграфического мусора (полный аналог обработки стандартного окна очистки файла). Добавим сюда дополнительную очистку геометрии нулевой длины, пустых текстовых объектов и непривязанных данных. Для этого проанализируем те опции, которые нам предоставляет _.-purge:

Команда: _-PURGE

```
Укажите тип неиспользуемых объектов для удаления
[Блоки/стилиВЫносныхэлементов/Рзмстили/Группы/Слои/типыЛин/мАтериа
лы/мвЫноскастиль/Пстили/Формы/Тстили/Млстили/стиливидовРАзрезов/тб
```

```
лстИли/визУальные стили/Зарегприл/геометрия Нулевой длины/пустыЕ
текстовые объекты/неПРивязанные данные/Все]:
Enter type of unused objects to purge
[Blocks/DEtailviewstyles/Dimstyles/Groups/LAyers/LTypes/MAterials/
MUltileaderstyles/Plotstyles/SHapes/textSTyles/Mlinestyles/SEction
viewstyles/Tablestyles/Visualstyles/Regapps/Zero-length
geometry/Empty text objects/Orphaned data/All]:
```

На данный момент нас интересуют пустые текстовые объекты (опция _e); геометрия нулевой длины (опция _z) и непривязанные данные (опция _o). Следовательно, код нашей команды становится таким:

```
(defun c:pua ()
  (command "_.-purge" "_a" "*" "_n")
  (command "_.-purge" "_e")
  (command "_.-purge" "_z")
  (command "_.-purge" "_o")
  ) ;_ end of defun
```

Мы собирались еще и зарегистрированные приложения (опция _r) почистить. Добавляем:

```
(defun c:pua ()
  (command "_.-purge" "_a" "*" "_n")
  (command "_.-purge" "_e")
  (command "_.-purge" "_z")
  (command "_.-purge" "_o")
  (command "_.-purge" "_r" "*" "_n")
  ) ;_ end of defun
```

Но дело в том, что очистку обычно приходится выполнять несколько раз. Практика показала, что за 3 прохода файл гарантированно очищается. Добавим указание, что все команды надо выполнить 3 раза:

```
 (defun c:pua ()
  (repeat 3
    (command "_.-purge" "_a" "*" "_n")
    (command "_.-purge" "_e")
    (command "_.-purge" "_z")
    (command "_.-purge" "_o")
    (command "_.-purge" "_r" "*" "_n")
    ) ;_ end of repeat
  ) ;_ end of defun
```

Можете взять любой свой dwg-файл, который хотелось бы почистить, и выполнить на нем нашу очистку.

Расширим нашу команду дальше: добавим проверку файла на ошибки с одновременным исправлением:

```
;;; \Samples\Lsp\aur_2014-starter(04).lsp
;;; Показаны только добавленные строки
 (defun c:pua ()
  (repeat 3
    (command "_.-purge" "_a" "*" "_n")
    (command "_.-purge" "_e")
    (command "_.-purge" "_z")
    (command "_.-purge" "_o")
    (command "_.-purge" "_r" "*" "_n")
    ) ;_ end of repeat
  (command "_.audit" "_y")
  ) ;_ end of defun
```

Сохраним наш файл как s:\lsp\cmd-pua.lsp.

На данный момент мы уже успешно настроили AutoCAD и создали свою собственную команду AutoCAD. И все это было сделано быстро, просто и без привлечения профессиональных разработчиков.

На самом деле можно сделать намного более серьезные и интересные вещи. Например, можно заставить AutoCAD выполнять очистку файла при попытке его сохранения; или активировать определенный текстовый стиль при начале работы команды _.dtext или _.mtext; или при переходе в пространство листа автоматически устанавливать значения некоторых переменных.

# Расширение стандартных команд

В AutoCAD есть понятие «реактора» - т.е. можно абсолютно прозрачно и незаметно для пользователя выполнить некоторые действия при старте команды, при ее окончании (как успешном, так и нет), при смене каких-либо системных переменных и еще при массе действий. Есть пара нюансов — во-первых, применение реакторов требует от разработчика достаточно высокой квалификации, и, во-вторых, применение command, как правило, запрещено.

Мы рассмотрим вариант принудительной установки текстового стиля AUR_2014 активным, как только пользователь вызывает команду создания однострочного или многострочного текста. Мы не будем создавать или модифицировать текстовый стиль — для упрощения кода предположим, что соответствующий текстовый стиль уже есть в чертеже (см. файл **AUR_2014-TextStyle.dwg**).

Находясь в VLIDE, создадим новый файл и сразу его сохраним как **aur_2014-cmd-react.lsp**. Для упрощения кода мы предусмотрим собственную реакцию только на начало и на корректное окончание команды:

```
;;; \Samples\Lsp\aur_2014-cmd-react.lsp
(vlr-command-reactor ; создаем командный реактор
  "AUR_2014-CommandReactor" ; Строка-идентификатор
  '(
    (:vlr-commandwillstart . aur_2014-vlr-command-start) ; Имя
lisp-функции, выполняемой
     ; при CommandWillStart – т.е. в момент, когда команда уже
вызвана, но еще
     ; не запущена
    (:vlr-commandended . aur_2014-vlr-command-end)
     ; Имя lisp-функции, выполняемой после окончания работы
команды
     ; но при этом управление еще обратно в AutoCAD не передано
    )
  ) ;_ end of vlr-command-reactor
```

Известно, что текущий текстовый стиль можно изменить, установив системную переменную TEXTSTYLE в необходимое значение. Воспользуемся этим:

```
;;; \Samples\Lsp\aur_2014-cmd-react.lsp
(defun aur_2014-vlr-command-start (react cmd)
  (setq cmd ; установим переменной cmd новое значение
        (strcase ; 3. Значение из п.2. - гарантированно строка.
Переведем ее в верхний регистр
          (car ; 2. Из п.1 получим первую часть списка
            cmd ; 1. Возьмем переданный параметра
            ) ;_ end of car
          ) ;_ end of strcase
        ) ;_ end of setq
  (if (wcmatch cmd "*TEXT") ; Если команда оканчивается на символы
TEXT, то
    (progn
      (setq *aur_2014-cur-textstyle* (getvar "textstyle"))
        ; Запомним старое значение TEXTSTYLE
      (setvar "textstyle" "AUR_2014")
        ; А потом установим новое значение
      ) ;_ end of progn
    ) ;_ end of if
  ) ;_ end of defun
```

```
(defun aur_2014-vlr-command-end (react cmd)
  (setq cmd ; установим переменной cmd новое значение
        (strcase ; 3. Значение из п.2. - гарантированно строка.
Переведем ее в верхний регистр
          (car ; 2. Из п.1 получим первую часть списка
            cmd ; 1. Возьмем переданный параметра
            ) ;_ end of car
          ) ;_ end of strcase
        ) ;_ end of setq
  (if (wcmatch cmd "*TEXT"); Если команда оканчивается на символы
TEXT, то
    (progn
      (if *aur_2014-cur-textstyle* ; Если раньше была сохранено
значение TEXTSTYLE, то
        (setvar "textstyle" *aur_2014-cur-textstyle*)
        ; Восстанавливаем старое значение TEXTSTYLE
        ) ;_ end of if
      (setq *aur_2014-cur-textstyle* nil)
        ; И уничтожаем "память" о старом значении TEXTSTYLE
      ) ;_ end of progn
    ) ;_ end of if
  ) ;_ end of defun
```

В результате при вводе как однострочного, так и многострочного текста, можно уже не беспокоиться насчет активации нужного текстового стиля. Правда, неплохо?

Мы рассмотрели вопрос т.н.командного реактора. Но есть аналогичный механизм, позволяющий выполнить дополнительные действия при смене системных переменных! Т.е, например, при смене пространства модели на пространство листа можно установить определенные значения некоторых системных переменных. Допустим, при активации пространства листа необходимо установить **PSLTSCALE** равной 1 и **DIMLFAC** равным -1

Вспомним, что при смене пространства меняются системные переменные **TILEMODE** и **CVPORT**: если хотя бы одна из них равна 1, то пользователь находится в пространстве листа. Кроме того, пользователь может «ходить» между листами — в таком случае меняется системная переменная **CTAB**. Вот изменение **CTAB** и будем отслеживать.

Точно так же, как и для командных реакторов, создаем новый файл **aur_2014-sysvar-react.lsp**:

```
;;; \Samples\Lsp\aur_2014-sysvar-react.lsp
(vlr-sysvar-reactor
  "AUR_2014-sysvar-reactor"
  '((:vlr-sysvarchanged . aur_2014-vlr-sysvar-changed))
  ) ;_ end of vlr-sysvar-reactor

(defun aur_2014-vlr-sysvar-changed (react value)
  (setq
    value
     (strcase
       (car
         value
         ) ;_ end of car
       ) ;_ end of strcase
    ) ;_ end of setq
  (if (or (= value "CTAB") ; Если произошла смена листа
          (= value "TILEMODE") ; или переход модель - лист
          ) ;_ end of or
    (progn
      (foreach sysvar '(("dimlfac" . -1)
                        ("psltscale" . 1)
```

```
                    )
        (setvar (car sysvar) (cdr sysvar))
        ) ;_ end of foreach
      ) ;_ end of progn
    ) ;_ end of if
  ) ;_ end of defun
```

Теперь, как бы пользователь ни старался, при активации листа у него гарантированно будет **PSLTSCALE** равной 1 и **DIMLFAC** равной -1.

На данный момент мы умеем при старте каждого файла dwg программно устанавливать определенные значения системных переменных; можем заставить AutoCAD выполнить некоторые команды; мы умеем создавать свои собственные команды и расширять работу имеющихся. Наконец, мы можем заставить AutoCAD реагировать особым образом, если меняется какая-либо системная переменная. Самое время перейти к следующему шагу.

# Настройка AutoCAD

Сначала добавим некоторые пути к путям поддержки (Support Path). Представим себе, что в сети для пользователей AutoCAD организован некий каталог (к примеру, \\server\acadusers) и этот каталог подключен как сетевой диск S: В этом каталоге организованы подкаталоги:

| Путь | Хранимые данные |
|---|---|
| s:\lsp | Lsp-файлы, которые должны быть загружены. |
| s:\arx | Arx-модули. Для упрощения предполагаем, что в каталоге хранятся только arx соответствующей версии и разрядности AutoCAD |
| s:\dotnet | .NET-сборки |
| s:\menu | Подгружаемые файлы меню |
| s:\support | Прочие файлы поддержки (например, шрифты и типы линий) |

Нам необходимо добавить эти каталоги как пути поддержки и (некоторые из них) как доверенные пути. Начнем с того, что попроще: с путей поддержки.

Для начала получим имеющиеся пути поддержки. Для этого можно воспользоваться очень старой функцикй LISP: getenv

```
(getenv "ACAD")
```

Результатом будет строка, в которой через точку с запятой перечисляются все пути поддержки (привожу только часть):

```
"C:\\Users\\Idevent-2\\appdata\\roaming\\autodesk\\autocad
2015\\r20.0\\enu\\support;C:\\program files\\autodesk\\<…>"
```

Сравним эту строку с каждым из добавляемых путей. Для сравнения используем функцию **wcmatch**. Есть одна тонкость: при сравнении строк имеет значение регистр символов. Поэтому переведем и полученную строку, и добавляемые пути в верхний регистр. Пройдемся по каждому пути и проверим — перечислен он или нет в имеющихся. Если нет — добавим в строку. А потом установим новую строку как новые пути поддержки:

```
;;; \Samples\Lsp\aur_2014-supportpaths.lsp
(defun aur_2014-supportpaths (/ str)
  (setq str (strcase ; переводим в верхний регистр
              (getvar "ACAD") ; все имеющиеся пути поддержки
              ) ;_ end of strcase
        ) ;_ end of setq
  (foreach ; 1. по каждому элементу
           path
         ; 2. Из списка ниже
              '("s:\\lsp"
                "s:\\arx"
                "s:\\dotnet"
```

```
                "s:\\support"
                )
    (if   ; если
      (not ; нет
        (wcmatch ; совпадения
          str ; имеющихся путей
          (strcat "*"
                  (strcase parh) ; и проверяемого пути
                  "*"
                  ) ;_ end of strcat
          ) ;_ end of wcmatch
        ) ;_ end of not
       (setq str (strcat path ";" str)) ; то добавить этот путь к
старым путям
       ) ;_ end of if
    ) ;_ end of foreach
  (setenv "ACAD" str) ; и установить новые пути поддержки
  ) ;_ end of defun
```

Пути поддержки изменены, теперь надо загрузить соответствующие приложения.

К сожалению, перед собственно загрузкой приложений потребуется выполнить некоторые действия. Дело в том, что с сервера безболезненно можно загружать только fas / vlx / lsp файлы. Возможность загружать arx-файлы с сетевых каталогов обычно зависит от квалификации разработчика (встречались ситуации, когда загрузка arx-модулей с сервера терпела фиаско). А вот .NET-сборки просто так с сервера точно не загрузить: как правило, требуется вмешательство в реестр, в раздел **HKEY_LOCAL_MACHINE** — а туда обычным пользователям позволен доступ только на чтение.

Намного проще скопировать эти файлы в какой-либо каталог на локальной машине и загружать их уже с локального компьютера. В качестве «получателя» можно создать свой каталог, но мы сейчас воспользуемся каталогом, который а) гарантированно есть на каждом компьютере; и б) в этот каталог каждый пользователь имеет право записи. Мы не будем оперировать с доверенными каталогами — намного проще временно отключить проверку безопасности и выполнить загрузку модулей. Можно, конечно, создавать свои собственные каталоги, но мы поступим проще: воспользуемся каталогом временных файлов (получить его в лиспе можно через функцию уже знакомую функцию **getenv**) Итак, приступим:

```
;;; \Samples\Lsp\aur_2014-modules.lsp
(defun aur_2014-modules (/ path file secureload)
;;; Временно отключим проверку безопасности:
  (setq secureload (getvar "secureload"))
  (setvar "secureload" 0)
          ; I. Пройдемся по каталогу arx:
  (foreach ; по каждому файлу
            file
                (vl-directory-files ; найти все файлы
                  (setq path "s:\\arx") ; в каталоге s:\arx
                  "*.arx" ; отвечающие маске
                  1 ; дополнительно указываем, что получать надо
только файлы
                  ) ;_ end of vl-directory-files
    (vl-file-copy ; копируем файл
      (strcat ; складываем
        path ; путь s:\arx
        "\\" ; добавляем слеш в конце
        file ; и имя найденного файла
        ) ;_ end of strcat
      (setq file ; 5. результат шагов 1-4 присвоим переменной file
            (strcat ; 1. складываем
              (getenv "TEMP") ; 2. имя каталога временных файлоа
              "\\" ; 3. добавим слеш в конце
```

```
                file ; 4. и не забудем имя копируемого файла
                ) ;_ end of strcat
              ) ;_ end of setq
        ) ;_ end of vl-file-copy
      (if (findfile file) ; если файл в каталоге %TEMP% найден
        (arxload file) ; то выполнить его загрузку
        ) ;_ end of if
      ) ;_ end of foreach

          ; II загружаем все .NET-модули
    (foreach ; по каждому файлу
             file
                 (vl-directory-files ; найти все файлы
                   (setq path "s:\\dotnet") ; в каталоге s:\dotnet
                   "*.dll" ; отвечающие маске
                   1 ; дополнительно указываем, что получать надо
только файлы
                   ) ;_ end of vl-directory-files
      (vl-file-copy ; копируем файл
        (strcat ; складываем
          path ; путь s:\arx
          "\\" ; добавляем слеш в конце
          file ; и имя найденного файла
          ) ;_ end of strcat
        (setq file ; 5. результат шагов 1-4 присвоим переменной file
              (strcat ; 1. складываем
                (getenv "TEMP") ; 2. имя каталога временных файлоа
                "\\" ; 3. добавим слеш в конце
                file ; 4. и не забудем имя копируемого файла
                ) ;_ end of strcat
              ) ;_ end of setq
        ) ;_ end of vl-file-copy
      (if (findfile file) ; если файл в каталоге %TEMP% найден
        (command "_.netload" file) ; то выполнить его загрузку
        ) ;_ end of if
      ) ;_ end of foreach
  ;;; Восстановим старое значение SECURELOAD
    (setvar "secureload" secureload)
    ) ;_ end of defun
```

Как уже было сказано, lsp-файлы могут вполне легко загружаться и с сервера. Поэтому сделаем очень интересный шаг: вспомним, что мы создавали файл s:\aur_2014-starter.lsp, и в acad2015doc.lsp обеспечили его вызов. Откроем наш «стартер» и добавим в него автоматическую загрузку всех *.lsp-файлов с сервера (временное отключение безопасности мы уже выполнили):

```
(foreach file (vl-directory-files "s:\\lsp" "*.lsp" 1)
  (load (strcat "s:\\lsp\\" file)
        ) ;_ end of load
  ) ;_ end of foreach
```

Этот код добавим перед словами `;; Тихий выход`

В результате мы получим такое содержание функции-«стартера» (привожу код файла полностью):

```
;;; \Samples\Lsp\aur_2014-starter(05).lsp
(defun aur_2014-start (/ secureload)
  ;; Сначала сообщим о начале работы наших дополнений
  (princ "\nНачало работы дополнительных операций : отдельный
файл")
  ;; Теперь установим системные переменные
```

```
  (foreach sysvar '(("blipmode" . 0)
                    ("gridmode" . 0)
                    ("secureload" . 1)
                    ("cmdecho" . 1)
                    ("nomutt" . 0)
                    ("isavepercent" . 0)
                    ("tooltips" . 0)
                    )
    (setvar (car sysvar) (cdr sysvar))
    ) ;_ end of foreach
  ;; Добавим выполнение команд
  (vl-cmdf "_.viewres" "_y" 200)
 (setq secureload (getvar "secureload"))
  (setvar "secureload" 0)
  (foreach file (vl-directory-files "s:\\lsp" "*.lsp" 1)
    (load (strcat "s:\\lsp\\" file)
          ) ;_ end of load
    ) ;_ end of foreach
  (setvar "secureload" secureload)

  ;; Сообщаем об окончании работы дополнений
  (princ "\nДополнительные операции завершены : отдельный файл")
  ;; Тихий выход
  (princ)
  ) ;_ end of defun

;; Вызовем только что определенную функцию
(aur_2014-start)

(defun c:pua ()
  (repeat 3
    (command "_.-purge" "_a" "*" "_n")
    (command "_.-purge" "_e")
    (command "_.-purge" "_z")
    (command "_.-purge" "_o")
    (command "_.-purge" "_r" "*" "_n")
    ) ;_ end of repeat
  (command "_.audit" "_y")
  ) ;_ end of defun
```

Мало программ ухитряются обходиться без взаимодействия с пользователем. Бывают пользователи, которые просто отключают командную строку и используют только меню (например, выпадающее). Для этого мастер-класса я создал пользовательское меню и сохранил его как s:\menu\aur_2014.cuix. Создадим дополнительную функцию, которая скопирует файл меню «с сервера» и загрузит копию:

```
;;; \Samples\Lsp\aur_2014-menuload.lsp
(defun aur_2014-menuload (/ file)
  (if (not (menugroup "aur_2014")) ; 1. Если не найдена группа
меню "AUR_2014"
    (progn ; 2. то
      (vl-file-copy ; 3. Скопируем
        "s:\\menu\\aur_2014.cuix" ; 4. файл меню с сервера
        (setq file ; 7. Результат шагов 5-6 присвоить новой
переменной
              (strcat
                (getenv "TEMP") ; 5. Копирование выполнять в
каталог %temp%
                "\\aur_2014.cuix" ; 6. Имя файла-копии
                ) ;_ end of strcat
```

```
            ) ;_ end of setq
          ) ;_ end of vl-file-copy
        (command "_.menuload" vile)
        ) ;_ end of progn
      ) ;_ end of if
    ) ;_ end of defun
```

Загрузка пользовательского меню выполняется не сложнее всего остального: мы точно так же можем скопировать *.cuix-файл на локальный компьютер и выполнить команду _.cuiload (допускается использование более старой команды_.-menuload; результат будет таким же. В коде я использовал именно старый подход)

После загрузки всех созданных lsp их необходимо запустить на выполнение. Вот тут нам и сослужит еще одну службу наш «стартер». Дополним его соответствующими вызовами. Представим себе, что у нас вообще первый запуск «стартера». В таком случае надо будет сделать следующее:

1. Прежде всего необходимо дополнить пути поддержки
2. Загрузить модули arx / .NET
3. Загрузить меню

Получается, что перед словами `;; Тихий выход` добавляются:

```
(aur_2014-supportpaths)
(aur_2014-modules)
(aur_2014-menuload)
```

Полный код «стартера» становится таким:

```
;;; \Samples\Lsp\aur_2014-starter.lsp
(defun aur_2014-start (/ secureload)
  ;; Сначала сообщим о начале работы наших дополнений
  (princ "\nНачало работы дополнительных операций : отдельный
файл")
  ;; Теперь установим системные переменные
  (foreach sysvar '(("blipmode" . 0)
                    ("gridmode" . 0)
                    ("secureload" . 1)
                    ("cmdecho" . 1)
                    ("nomutt" . 0)
                    ("isavepercent" . 0)
                    ("tooltips" . 0)
                    )
    (setvar (car sysvar) (cdr sysvar))
    ) ;_ end of foreach
  ;; Добавим выполнение команд
  (vl-cmdf "_.viewres" "_y" 200)
 (setq secureload (getvar "secureload"))
  (setvar "secureload" 0)
  (foreach file (vl-directory-files "s:\\lsp" "*.lsp" 1)
    (load (strcat "s:\\lsp\\" file)
          ) ;_ end of load
    ) ;_ end of foreach
  (setvar "secureload" secureload)

 (aur_2014-supportpaths)
 (aur_2014-modules)
 (aur_2014-menuload)

  ;; Сообщаем об окончании работы дополнений
  (princ "\nДополнительные операции завершены : отдельный файл")
  ;; Тихий выход
  (princ)
  ) ;_ end of defun
```

```
;; Вызовем только что определенную функцию
(aur_2014-start)

(defun c:pua ()
  (repeat 3
    (command "_.-purge" "_a" "*" "_n")
    (command "_.-purge" "_e")
    (command "_.-purge" "_z")
    (command "_.-purge" "_o")
    (command "_.-purge" "_r" "*" "_n")
    ) ;_ end of repeat
  (command "_.audit" "_y")
  ) ;_ end of defun
```

# Подведем промежуточные итоги

Подведем некоторые итоги:

- Мы можем привести AutoCAD к стандартизированному на предприятии виду
- Мы можем создать собственные команды, упрощая и ускоряя работу конечного пользователя
- Мы можем расширить поведение штатных команд, создавая и расширяя автоматическую поддержку стандартов предприятия
- Мы можем настроить AutoCAD, добавив ему нужные нам пути поддержки
- Мы можем загрузить в AutoCAD дополнительные модули, написанные на C++ или с применением .NET-технологий
- Мы можем обеспечить конечного пользователя дополнительным меню

И все это за достаточно короткое время и с использованием инструментов, которые доступны в AutoCAD изначально!

Все, что необходимо для «быстрого старта», уже известно. Хотелось бы поговорить еще о некоторых возможностях LISP'a.

# Работа с реестром Windows

LISP (а, точнее, VisualLISP) позволяет легко и просто читать данные из реестра и выполнять запись в реестр. Единственное требование — пользователь должен иметь право на запись в соответствующий раздел реестра.

Если просмотреть все коды, которые мы разработали, то можно увидеть, что мы нередко указывали напрямую в коде, с какими каталогами необходимо работать. Прежде всего это касается «серверных» каталогов.

В то же время можно потребовать у службы IT доменными политиками напрямую в определенной ветке реестра ввести необходимые нам данные и работать уже с ними.

Допустим, у нас есть договоренность об использовании ветки HKEY_CURRENT_USER\Software\AUR2014 для хранения и дополнения необходимых данных. Фактически на данный момент нам необходим только один ключ — тот, в котором будет храниться путь к серверному каталогу. Пускай имя ключа будет ServerPath. В таком случае для чтения этих данных в любом месте вместо «**s:\\<…>**» можно будет ввести

```
(strcat (vl-registry-read "HKEY_CURRENT_USER\\Software\\AUR2014"
"ServerPath") "\\" "<...>")
```

И дальше работать уже с нормальными данными. Чем хорош такой подход?

Например, сменилось имя сетевого диска — вместо S: по каким-то соображениям было решено использовать диск B:; или вообще обойтись без сетевых дисков, и использовать абсолютные пути (типа \\SERVER\ACADUSERS); или Вы разрабатываете технологию, которую будете применять в нескольких разных сетях... Неужели придется переписывать все коды? Нет: если эти данные будут вноситься централизованно (оставим эту задачу службе IT), то независимо ни от чего Ваш код будет использовать корректные данные.

Естественно, что можно и записывать данные в реестр. Это позволяет:

- Хранить некоторые данные между сессиями AutoCAD'a

- Просмотреть эти данные в удобной структурированной форме
- При необходимости изменить эти данные можно очень быстро, централизованно и абсолютно прозрачно для пользователя.

На самом деле возможности LISP'а намного шире, но у нас уже не хватает времени даже на беглый обзор.

# Вспомним наши цели

Мы добились того, что AutoCAD настраивается в полном соответствии с требованиями предприятия.

Мы упростили работу CAD-менеджера, потратив на это минимум времени и сил.

Мы подарили пользователям новые команды, ускоряющие их работу

Мы создали условия, в которых нарушить стандарт предприятия сложнее, чем ему следовать.

Мы создали систему, которую можем расширять и модернизировать, обеспечив автоматическую загрузку модулей, разработанных на любых языках (LSP, .NET, C++)

# Сравним LISP и .NET

| LISP | .NET |
|---|---|
| Не зависит от версии и разрядности AutoCAD | Зависит от версии AutoCAD (по крайней мере в трехлетнем цикле) |
| Интерфейс – только ком.строка и диалоги. И разработать их не совсем просто | Мощнейшая библиотека для разработки практически любого интерфейса |
| Отладка и изменение кода может идти прямо во время выполнения. Но отладка не является простой задачей | Отладка и изменение кода во время выполнения стали возможны только в AutoCAD 2015 |
| Многие вещи (например, работа с XML или строками) требуют отдельной разработки | Те же самые обработки XML и строк встроены в .NET |
| Безтиповой язык, есть определенные трудности при разработке | Контроль типов данных обязателен |
| Достаточно узкая область применения: AutoCAD и все | AutoCAD, Revit, Inventor, сторонние приложения… |

Обе технологии имеют свои плюсы и свои минусы. Для быстрой разработки несложных приложений можно использовать LISP. Но как только вопрос выходит за рамки AutoCAD и начинает затрагивать плотную работу со сторонними приложениями, разработку сложного интерфейса — то я бы рекомендовал обратить внимание на .NET как на более перспективное и развивающееся направление.