ALEX LUNDIN

Design Engineer
Wattstopper, Legrand
2240 Campbell Creek Blvd
Plano TX, 75074

TELEPHONE (469) 394-5175
EMAIL alexander.m.lundin@gmail.com

SAMPLES OF WORK

Source control links

https://bitbucket.org/AlexLundin10202014

https://github.com/AlexLundinEducational

Auto LISP Source code image DEFAULT-ERROR-HANLDER (full source code posted at the end)

```
(defun DEFAULT-ERROR-HANDLER
       (
       ;arguments
       THECALLINGFUNCTIONSNAME
       ERRORMESSAGEFROMAUTOCAD
       /
       ;local variables
       *ERROR*
       )
;;; <Arguments and Return>
;;; Arguments:
;;; theCallingFunctionsName     - name of calling function, which is defined in the error handler inside the caller
;;;     errorMessageFromAutoCAD     - error message generated from AutoCAD software environment
;;; Return:
;;; None                - the calling LISP rountine will terminate upon calling this function
;;; </Arguments and Return>
;;; Standard *error* function for Legrand
;;; View the file below for a working example and more in depth notes on how to implement and use it
;;; C:\WS_Blocks\Default\LISP\Lisp Functions\Subfunction Library\DEFAULT-ERROR-HANDLER.LSP
    (defun *error* ( msg / THECALLINGFUNCTIONSNAME)
       (setq theCallingFunctionsName "DEFAULT-ERROR-HANDLER")
       (DEFAULT-ERROR-HANDLER theCallingFunctionsName msg)
    )
;;; function body
;;; Default error handler for all functions at Legrand
;;; This error function is useful for debugging
;;; AutoCAD has built in error messages
;;; They are sent into this function from the calling function
;;; This conditional block prints the calling function and the error message for the client
    (princ "\nDEFAULT-ERROR-HANDLER --> activated.")
    (princ "\n--------------------------report--------------------------")
    (cond

       ;default messsage, exit occurs naturally
       ((= errorMessageFromAutoCAD "Function cancelled")
       (princ (strcat "\n" theCallingFunctionsName " --> was canceled."))
       (princ "\nPossible causes:")
       (princ "\n-Terminating a custom command early.")
       )
       ;default messsage, exit occurs naturally
       ((= errorMessageFromAutoCAD "quit / exit abort")
       (princ (strcat "\n" theCallingFunctionsName " --> was quit."))
       (princ "\nPossible causes:")
       (princ "\n-Terminating a custom command early.")
       (princ "\n-Choosing a incorrect object type.")
       )

       ; known condition that results from localizing a keywork called vb-String
       ; localizing blue keywords is always a bad idea
       ((= errorMessageFromAutoCAD "bad argument type: fixnump: nil")
       (princ (strcat "\n" theCallingFunctionsName " --> encountered a common error."))
       (princ "\nThe following is the error message from the AutoCAD Software.")
       (princ "\n")
       (princ errorMessageFromAutoCAD)
       (princ "\nPossible causes:")
       (princ "\n-Localizing a LISP keyword during programming.")
       (princ "\n  ^ to solve this, look at your local variables, any blue words must not be localized.")
       (princ "\n-A function requiring an integer argument has been passed an argument of incorrect data type with the value noted in the error message.")
       )
       ;t conditional
       ;catch all others
       ;any other error message, other than the two default ones above
       (t
       (princ (strcat "\n" theCallingFunctionsName " --> encountered an unexpected error."))
       (princ "\nThe following is the error message from the AutoCAD Software.")
       (princ "\n")
       (princ errorMessageFromAutoCAD)
       ;control now passes back to AutoCAD and the end of the t conditional
       ;the client will see what the error message is
       )
    )
    ;the first two conditionals make it here and the end
    ;no return value
    (princ "\n-------------------------end report-------------------------")
    (princ "\nExiting now.")
    (vl-exit-with-value 1)

)
```

AutoCAD Video Demonstrations

Control Bar
https://autode.sk/2SWDVCJ

- This is a GUI component I created for a single point of contact, that our user's can have for all 100+ pieces of LISP code.

VLEX Test Runner
https://autode.sk/2sByZYq

- This is a regression test suite I built to allow new developers to start contributing to our source code, with a degree of certainty, on how their changes will effect the current build. There is pass or fail status for each test, making it clear what worked and what didn't during the current changes the new developer made.

CoreBot
https://knowledge.autodesk.com/community/screencast/2d79ca13-b46f-405e-a6e8-cf135bfb04c8

- This is an example of a method I developed to print AutoCAD drawings without opening the program. This is just one of the many Automation Techniques I use for streamlining workflows.

Excel File Block Replace
https://knowledge.autodesk.com/community/screencast/373f02cf-73ff-4ac9-90e6-7cb24624520b

- This LISP function reads an excel file and replaces outdated part numbers, with the block name that matches the new part number

Model Space Viewports
https://knowledge.autodesk.com/community/screencast/24d5d6fd-8f9f-498d-b4fe-fa212c729cbe

- This is an example of a LISP function I created to generate viewports from rectangles in model space.

Samples of University Projects

Webpage Maintenance, using Test Driven Development for enhancements
https://github.com/AlexLundinEducational/SE-4367-Testing

- This is the repository we created for our Software Testing Class

Maintainability assessment of code and simple Desktop GUI
https://autode.sk/2HoonRK

- Our first assignment was to launch the webpage, and write our assessment of how easy or difficult this webpage will be to maintain or edit

Allow user to add to facts XML file
https://autode.sk/2l4AXg9

- This video shows the evolution of the Webpage, as the semester progressed

Manual Junit Tests - for validating webpage XML data from user
https://autode.sk/2taZEfw

- This video shows the first round of manual testing, before we learned about Automated regression testing

Automated Selenium Tests - from Input Domain Model for Facts object
https://autode.sk/2zbXhOT

- This video shows me using selenium tests that my team created to validate the input domain for the facts object

Full Source Code From DEFAULT-ERROR-HANLDER Above

```
;;; -----------------------------------------------------------------------------------------------------------------------------------------------
;;; <ZZ-example>

;;; Write an example call to the Subfunction here

;;; This will allow other developers to verify how you intend your code to work

;;; Example function c:ZZ with DEFAULT-ERROR-HANDLER Subfunction Call

;;;(defun c:ZZ ( / )

;;;	;;; *error* function

;;;	;;; AutoCAD looks for this specific function definition whenever the variable *error* contains a error message

;;;	;;; This is the standard *error* function for Legrand

;;;	;;; This error function is useful for debugging

;;;	;;; You can define this error handling function inside any of the other functions for easy error trapping

;;;	;;; The syntax for the function name must stay exactly as follows

;;;	;;; The error function has asterisks on either side and accepts exactly 1 argument, msg

;;;	;;; The AutoCAD LISP programming environment will look for this function and call it for any unexpected error during runtime

;;;	;;; Inside the function definition *error* you can get creative with the implementation that makes the most sense

;;;	;;; This one makes use of another sub function to print out the calling functions name and the error message

;;;	(defun *error* ( msg / THECALLINGFUNCTIONSNAME)

;;;		(setq theCallingFunctionsName "c:ZZ")

;;;		(DEFAULT-ERROR-HANDLER theCallingFunctionsName msg)

;;;	)

;;;	;;; c:ZZ function body

;;;

;;;	;legal example

;;;	(setq listOfIntegers (list 55 47 2))

;;;	(setq indexValue 0)

;;;	; this line attempts access on a list vaiable through the 0 index

;;;	; this is legal because the supplied variable is a list

;;;	(setq valueAtIndex (nth indexValue listOfIntegers))

;;;

;;;	;illegal example, activates *error* function, which is user defined within each LISP program

;;;	;The default *error* function Wrapper, which exists withing each .lsp template, calls DEFAULT-ERROR-HANDLER to give consistent results to users
;;;	(setq integerValue 55)

;;;	(setq indexValue 0)

;;;	; this line attempts access on a list vaiable through the 0 index

;;;	; this is illegal because the variable being accessed, is actually a integer, which is a invalid data type for list access by index

;;;	(setq valueAtIndex (nth indexValue integerValue))
```

```
;;;

;;;)

;;; </ZZ-example>

;;;-----------------------------------------------------------------------------------------------------------------------------------------------------
;;; <Subfunction Name>

;;; DEFAULT-ERROR-HANDLER

;;; </Subfunction Name>

;;; <Developer Notes>

;;; This function is the operating procedure for error handling.

;;; The standard *error* function is defined in larger functions for modular code reuse and consistent error trapping.

;;; AutoCAD looks for the *error* function automatically, so each code that uses the standard *error* implementation, will call DEFAULT-ERROR-HANDLER
;;; this DEFAULT-ERROR-HANDLER by using the wrapper.

;;; The standard *error* function for Legrand is defined in LISP Functions / Templates folder

;;; C:\WS_Blocks\Default\LISP\Lisp Functions\Templates

;;; Alex Lundin

;;; 11-18-2017

;;;

;;; Alex Lundin

;;; 01-08-2018

;;; Revision 1.1

;;; removed second end report bar on the t conditional catch all

;;;

;;; Alex Lundin

;;; 01-12-2018

;;; Revision 1.2

;;; added exit and end of function call to ensure any DEFAULT-ERROR-HANDLER explicity made by the programmer, will terminate before return to caller
;;;

;;; Alex Lundin

;;; 08-09-2018

;;; Revision 2.0

;;; Started adding more error handling for common programming errors, such as localizing specific keyworks for the LISP programming language
;;; </Developer Notes>

(defun DEFAULT-ERROR-HANDLER
                        (
                        ;arguments
                        THECALLINGFUNCTIONSNAME
                        ERRORMESSAGEFROMAUTOCAD
                        /
                        ;local variables
                        *ERROR*
```

```
                )

;;; <Arguments and Return>

;;; Arguments:

;;;         theCallingFunctionsName  - name of calling function, which is defined in the error handler inside the caller

;;;         errorMessageFromAutoCAD                 - error message generated from AutoCAD software environment

;;; Return:

;;;         None                                    - the calling LISP rountine will terminate upon calling this function

;;; </Arguments and Return>


;;; Standard *error* function for Legrand

;;; View the file below for a working example and more in depth notes on how to implement and use it

;;; C:\WS_Blocks\Default\LISP\Lisp Functions\Subfunction Library\DEFAULT-ERROR-HANDLER.LSP

        (defun *error* ( msg / THECALLINGFUNCTIONSNAME)

                (setq theCallingFunctionsName "DEFAULT-ERROR-HANDLER")
                (DEFAULT-ERROR-HANDLER theCallingFunctionsName msg)
        )

;;; function body


;;; Default error handler for all functions at Legrand

;;; This error function is useful for debugging

;;; AutoCAD has built in error messages

;;; They are sent into this function from the calling function

;;; This conditional block prints the calling function and the error message for the client


        (princ "\nDEFAULT-ERROR-HANDLER --> activated.")
        (princ "\n---------------------------report---------------------------")
        (cond

                ;default messsage, exit occurs naturally
                ((= errorMessageFromAutoCAD "Function cancelled")
                (princ (strcat "\n" theCallingFunctionsName " --> was canceled."))
                (princ "\nPossible causes:")
                (princ "\n-Terminating a custom command early.")
                )
                ;default messsage, exit occurs naturally
                ((= errorMessageFromAutoCAD "quit / exit abort")
                (princ (strcat "\n" theCallingFunctionsName " --> was quit."))
                (princ "\nPossible causes:")
                (princ "\n-Terminating a custom command early.")
                (princ "\n-Choosing a incorrect object type.")
                )

                ; known condition that results from localizing a keywork called vb-String
                ; localizing blue keywords is always a bad idea
                ((= errorMessageFromAutoCAD "bad argument type: fixnump: nil")
                (princ (strcat "\n" theCallingFunctionsName " --> encountered a common error."))
                (princ "\nThe following is the error message from the AutoCAD Software.")
                (princ "\n")
                (princ errorMessageFromAutoCAD)
                (princ "\nPossible causes:")
                (princ "\n-Localizing a LISP keyword during programming.")
                (princ "\n^ to solve this, look at your local variables, any blue words must not be localized.")
                (princ "\n-A function requiring an integer argument has been passed an argument of incorrect data type with the value
```

```
noted in the error message.")
                        )
                        ;t conditional
                        ;catch all others
                        ;any other error message, other than the two default ones above
                        (t
                        (princ (strcat "\n" theCallingFunctionsName " --> encountered an unexpected error."))
                        (princ "\nThe following is the error message from the AutoCAD Software.")
                        (princ "\n")
                        (princ errorMessageFromAutoCAD)
                        ;control now passes back to AutoCAD and the end of the t conditional
                        ;the client will see what the error message is
                        )
        )

        ;the first two conditionals make it here and the end
        ;no return value
        (princ "\n-------------------------end report-------------------------")
        (princ "\nExiting now.")
        (vl-exit-with-value 1)

)
```