```
File created:    4-Aug-88 18:06:52  {ERIS}<TEST>MAIKO>HAND>MAIKO-ARRAY-TESTS.;9

  changes to:    (FUNCTIONS USER::POINTER-ARRAY-TESTS)

previous date:   22-Jun-88 13:52:22  {ERIS}<TEST>MAIKO>HAND>MAIKO-ARRAY-TESTS.;8

  Read Table:    XCL

     Package:    INTERLISP

      Format:    XCCS
```

```
(RPAQQ MAIKO-ARRAY-TESTSCOMS
       (;; Tests for AREF & ASET in Maiko
        ;; TO DO:  Extendable arrays, Adjustable arrays, extend past 2**15 and make sure contents are still there.  Vectors, strings.
        ;; Main test invokation function:
        (FNS MAIKO-ARRAY-TESTS)
        ;; 1-dimensional array tests:
        (FUNCTIONS USER::BIT-ARRAY-TESTS USER::BYTE-ARRAY-TESTS USER::CHAR-ARRAY-TESTS USER::FLOAT-ARRAY-TESTS
               USER::POINTER-ARRAY-TESTS USER::XPOINTER-ARRAY-TESTS)
        ;; Simple AREF & ASET of 1-, 2-, 3-d # arrays:
        (FNS SIMPLE-AREF-ASET-TESTS NEQP)
        ;; Test of past known failures
        (FUNCTIONS USER::PAST-ARRAY-FAILURE-CASES)
        ;; Assure that we compile with CL:COMPILE-FILE:
        (PROPS (MAIKO-ARRAY-TESTS FILETYPE))))

;; Tests for AREF & ASET in Maiko
;; TO DO:  Extendable arrays, Adjustable arrays, extend past 2**15 and make sure contents are still there.  Vectors, strings.
;; Main test invokation function:

(DEFINEQ

(MAIKO-ARRAY-TESTS
  (LAMBDA (LIMIT)                                          ; Edited 22-Jun-88 13:51 by jds
    ;; Main entry point to the Maiko array op-code tests.
    (|for| I |from| 1 |to| LIMIT |do| (PRINTOUT T T "Starting Maiko array op-code tests, iteration #" I T)
                                      (USER::BIT-ARRAY-TESTS 2)
                                      (USER::BYTE-ARRAY-TESTS 2)
                                      (USER::CHAR-ARRAY-TESTS 2)
                                      (USER::FLOAT-ARRAY-TESTS 2)
                                      (USER::POINTER-ARRAY-TESTS 2)
                                      (USER::XPOINTER-ARRAY-TESTS 2)
                                      (PRINTOUT T "  Starting #-array aref/set tests for 1-3 dims.")
                                      (SIMPLE-AREF-ASET-TESTS)
                                      (USER::PAST-ARRAY-FAILURE-CASES 1))))
)

;; 1-dimensional array tests:


(CL:DEFUN USER::BIT-ARRAY-TESTS (USER::LIMIT)
   (FOR USER::LOOP-NO FROM 1 TO USER::LIMIT
      COLLECT (CL:FORMAT T "  Starting bit-array tests, iteration ~D~%" USER::LOOP-NO)
            (FOR USER::MIN-LENGTH IN '(1 9 17 33 32768) AS USER::MAX-LENGTH
               IN '(8 16 32 32767 65535) DO (LET* ((USER::LEN (RAND USER::MIN-LENGTH USER::MAX-LENGTH))
                                                   (USER::ZERO-ARRAY (CL:MAKE-ARRAY USER::LEN :ELEMENT-TYPE
                                                                            'BIT :INITIAL-ELEMENT 0))
                                                   (USER::ONE-ARRAY (CL:MAKE-ARRAY USER::LEN :ELEMENT-TYPE
                                                                           'BIT :INITIAL-ELEMENT 1)))
                                                  (CL:FORMAT T "    Array size = ~D~%" USER::LEN)
                                                  (ERSETQ (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                                                 ((= USER::I USER::LEN))
                                                              (CL:WHEN (CL:/= (CL:AREF USER::ZERO-ARRAY USER::I)
                                                                              0)
                                                                     (CL:ERROR "**Zero-array wasn't zero at
                                                                            element ~d.~%" USER::I))
                                                              (CL:WHEN (CL:/= (CL:AREF USER::ONE-ARRAY USER::I)
                                                                              1)
                                                                     (CL:ERROR "**One-array wasn't one at
                                                                            element ~d.~%" USER::I))))
                                                  (ERSETQ (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                                                 ((= USER::I USER::LEN))
                                                              (CL:SETF (CL:AREF USER::ZERO-ARRAY USER::I)
                                                                     (COND
                                                                        ((EVENP USER::I)
                                                                         1)
                                                                        (T 0)))
                                                                 (CL:WHEN (CL:/= (CL:AREF USER::ZERO-ARRAY USER::I)
                                                                                 (COND
                                                                                    ((EVENP USER::I)
                                                                                     1)
                                                                                    (T 0)))
                                                                        (CL:ERROR "EVENP pattern fails at ~D.~%"
                                                                               USER::I))))))))


(CL:DEFUN USER::BYTE-ARRAY-TESTS (USER::LIMIT)
   ;; Tests of byte arrays, for bytes of length 1, 8, 16, and 32 bits.
   (FOR USER::LOOP-NO FROM 1 TO USER::LIMIT
      COLLECT (CL:FORMAT T "  Starting byte-array tests, iteration ~D~%" USER::LOOP-NO)
            (FOR USER::BYTE-LEN IN '(1 8 16 32) AS USER::MAX-VALUE IN '(2 256 65535 65535)
               DO (CL:FORMAT T "    Byte length = ~D~%" USER::BYTE-LEN)
                  (FOR USER::MIN-LENGTH IN '(1 9 17 33 32768) AS USER::MAX-LENGTH
                     IN '(8 16 32 32767 65535) DO (LET* ((USER::LEN (RAND USER::MIN-LENGTH USER::MAX-LENGTH))
                                                         (USER::ZERO-ARRAY (CL:MAKE-ARRAY USER::LEN
                                                                                  :ELEMENT-TYPE
                                                                                  (LIST 'CL:UNSIGNED-BYTE
                                                                                        USER::BYTE-LEN)
                                                                                  :INITIAL-ELEMENT 0))
                                                         (USER::ONE-ARRAY (CL:MAKE-ARRAY USER::LEN
                                                                                 :ELEMENT-TYPE
                                                                                 (LIST 'CL:UNSIGNED-BYTE
                                                                                       USER::BYTE-LEN)
                                                                                 :INITIAL-ELEMENT 1)))
                                                        (CL:FORMAT T "   Array size = ~D~%" USER::LEN)
                                                        (ERSETQ (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                                                       ((= USER::I USER::LEN))
                                                                    (CL:WHEN (CL:/= (CL:AREF USER::ZERO-ARRAY
                                                                                             USER::I)
                                                                                    0)
                                                                           (CL:ERROR "**Zero-array wasn't zero
                                                                                  at element ~d.~%" USER::I))
                                                                    (CL:WHEN (CL:/= (CL:AREF USER::ONE-ARRAY
                                                                                             USER::I)
                                                                                    1)
                                                                           (CL:ERROR "**One-array wasn't one at
                                                                                  element ~d.~%" USER::I))))
                                                        (ERSETQ (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                                                       ((= USER::I USER::LEN))
                                                                    (CL:SETF (CL:AREF USER::ZERO-ARRAY USER::I)
                                                                           (CL:REM USER::I USER::MAX-VALUE))
                                                                    (CL:WHEN (CL:/= (CL:AREF USER::ZERO-ARRAY
                                                                                             USER::I)
                                                                                    (CL:REM USER::I
                                                                                           USER::MAX-VALUE))
                                                                           (CL:ERROR "EVENP pattern fails at
                                                                                  ~D.~%" USER::I)))))))))


(CL:DEFUN USER::CHAR-ARRAY-TESTS (USER::LIMIT)
   (FOR USER::LOOP-NO FROM 1 TO USER::LIMIT
      COLLECT (CL:FORMAT T "  Starting bit-array tests, iteration ~D~%" USER::LOOP-NO)
            (FOR USER::MIN-LENGTH IN '(1 9 17 33 32768) AS USER::MAX-LENGTH
               IN '(8 16 32 32767 65535) DO (LET* ((USER::LEN (RAND USER::MIN-LENGTH USER::MAX-LENGTH))
                                                   (USER::ZERO-ARRAY (CL:MAKE-ARRAY USER::LEN :ELEMENT-TYPE
                                                                            'CL:CHARACTER :INITIAL-ELEMENT #\D))
                                                   (USER::ONE-ARRAY (CL:MAKE-ARRAY USER::LEN :ELEMENT-TYPE
                                                                           'CL:CHARACTER :INITIAL-ELEMENT
                                                                           (CL:INT-CHAR (CHARCODE "41,133")))))
                                                  (CL:FORMAT T "    Array size = ~D~%" USER::LEN)
                                                  (ERSETQ (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                                                 ((= USER::I USER::LEN))
                                                              (CL:WHEN (NEQ (CL:AREF USER::ZERO-ARRAY USER::I)
                                                                            #\D)
                                                                     (CL:ERROR "**Zero-array wasn't zero at
                                                                            element ~d.~%" USER::I))
                                                              (CL:WHEN (NEQ (CL:AREF USER::ONE-ARRAY USER::I)
                                                                            (CL:INT-CHAR (CHARCODE "41,133")))
                                                                     (CL:ERROR "**One-array wasn't one at
                                                                            element ~d.~%" USER::I))))))))


(CL:DEFUN USER::FLOAT-ARRAY-TESTS (USER::LIMIT)
   (FOR USER::LOOP-NO FROM 1 TO USER::LIMIT
      COLLECT (CL:FORMAT T "  Starting FLOAT-array tests, iteration ~D~%" USER::LOOP-NO)
            (FOR USER::MIN-LENGTH IN '(1 9 17 33 32768) AS USER::MAX-LENGTH
               IN '(8 16 32 32767 65535)
               DO (LET* ((USER::LEN (RAND USER::MIN-LENGTH USER::MAX-LENGTH))
                         (USER::ZERO-ARRAY (CL:MAKE-ARRAY USER::LEN :ELEMENT-TYPE 'FLOAT :INITIAL-ELEMENT 0.0))
                         (USER::ONE-ARRAY (CL:MAKE-ARRAY USER::LEN :ELEMENT-TYPE 'FLOAT :INITIAL-ELEMENT 1.0)))
                        (CL:FORMAT T "    Array size = ~D~%" USER::LEN)
                        (ERSETQ (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                       ((= USER::I USER::LEN))
```

```
                                              (CL:WHEN (CL:/= (CL:AREF USER::ZERO-ARRAY USER::I)
                                                              0.0)
                                                     (CL:ERROR "**Zero-array wasn't zero at element ~d.~%" USER::I))
                                              (CL:WHEN (CL:/= (CL:AREF USER::ONE-ARRAY USER::I)
                                                              1.0)
                                                     (CL:ERROR "**One-array wasn't one at element ~d.~%" USER::I))))
                                  (ERSETQ (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                                 ((= USER::I USER::LEN))
                                              (CL:SETF (CL:AREF USER::ZERO-ARRAY USER::I)
                                                     (CL:SIN (CL:* USER::I (/ 3.1415927 USER::LEN))))
                                              (CL:WHEN (CL:/= (CL:AREF USER::ZERO-ARRAY USER::I)
                                                              (CL:SIN (CL:* USER::I (/ 3.1415927 USER::LEN))))
                                                     (CL:ERROR "SIN pattern fails at ~D.~%" USER::I))))
```

;; Just create 1000 of floats into the array, and read them out, so we can run STORAGE later to see if they leaked.

```
                                  (CL:DO ((USER::I 0 (CL:1+ USER::I))
                                          (CL:ELT (RAND 0 (CL:1- USER::LEN))
                                                 (RAND 0 (CL:1- USER::LEN))))
                                         ((= USER::I 1000))
                                      (CL:SETF (CL:AREF USER::ZERO-ARRAY CL:ELT)
                                             (CL:SIN (CL:* USER::I (/ 3.1415927 USER::LEN))))
                                      (CL:WHEN (CL:/= (CL:AREF USER::ZERO-ARRAY CL:ELT)
                                                      (CL:SIN (CL:* USER::I (/ 3.1415927 USER::LEN))))
                                             (CL:ERROR "SIN pattern fails at ~D.~%" USER::I)))))))
```

```
(CL:DEFUN USER::POINTER-ARRAY-TESTS (USER::LIMIT)
   (FOR USER::LOOP-NO FROM 1 TO USER::LIMIT
      COLLECT (CL:FORMAT T "  Starting pointer-array tests, iteration ~D~%" USER::LOOP-NO)
            (FOR USER::MIN-LENGTH IN '(1 9 17 33 32768) AS USER::MAX-LENGTH
               IN '(8 16 32 32767 65535) DO (LET* ((USER::LEN (RAND USER::MIN-LENGTH USER::MAX-LENGTH))
                                                   (USER::ZERO-ARRAY (CL:MAKE-ARRAY USER::LEN :INITIAL-ELEMENT 0
                                                                            ))
                                                   (USER::ONE-ARRAY (CL:MAKE-ARRAY USER::LEN :INITIAL-ELEMENT 1)
                                                          )
                                                   (USER::GC-ITEM (CREATE FMTSPEC))
                                                   USER::OLD-REFCNT)
                                                  (CL:FORMAT T "    Array size = ~D~%" USER::LEN)
                                                  (ERSETQ (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                                                 ((= USER::I USER::LEN))
                                                              (CL:WHEN (CL:/= (CL:AREF USER::ZERO-ARRAY USER::I)
                                                                              0)
                                                                     (CL:ERROR "**Zero-array wasn't zero at
                                                                            element ~d.~%" USER::I))
                                                              (CL:WHEN (CL:/= (CL:AREF USER::ONE-ARRAY USER::I)
                                                                              1)
                                                                     (CL:ERROR "**One-array wasn't one at
                                                                            element ~d.~%" USER::I))))
                                                  (ERSETQ (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                                                 ((= USER::I USER::LEN))
                                                              (CL:SETF (CL:AREF USER::ZERO-ARRAY USER::I)
                                                                     (COND
                                                                        ((EVENP USER::I)
                                                                         1)
                                                                        (T 0)))
                                                              (CL:WHEN (CL:/= (CL:AREF USER::ZERO-ARRAY USER::I)
                                                                              (COND
                                                                                 ((EVENP USER::I)
                                                                                  1)
                                                                                 (T 0)))
                                                                     (CL:ERROR "EVENP pattern fails at ~D.~%"
                                                                            USER::I))))
```

;; Make sure that putting a pointer to something into an array adds to the refcount.

```
                                                  (ERSETQ (CL:SETQ USER::OLD-REFCNT (\\REFCNT USER::GC-ITEM))
                                                         (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                                                ((= USER::I USER::LEN))
                                                             (CL:SETQ USER::OLD-REFCNT (\\REFCNT USER::GC-ITEM))
                                                             (CL:SETF (CL:AREF USER::ZERO-ARRAY USER::I)
                                                                    USER::GC-ITEM)
                                                             (OR (EQ (CL:AREF USER::ZERO-ARRAY USER::I)
                                                                     USER::GC-ITEM)
                                                                 (CL:ERROR "Filling array with GC sample item
                                                                        failed at ~D.~%" USER::I))
                                                             (CL:WHEN (CL:/= (\\REFCNT USER::GC-ITEM)
                                                                             (CL:1+ USER::OLD-REFCNT))
                                                                    (CL:ERROR "ASET doesn't bump ref-count at
                                                                           ~D.~%" USER::I)))
                                                         (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                                                ((= USER::I USER::LEN))
                                                             (CL:SETQ USER::OLD-REFCNT (\\REFCNT USER::GC-ITEM))
                                                             (CL:SETF (CL:AREF USER::ZERO-ARRAY USER::I)
                                                                    NIL)
                                                             (OR (NOT (CL:AREF USER::ZERO-ARRAY USER::I))
                                                                 (CL:ERROR "Filling array with NIL failed at
                                                                        ~D.~%" USER::I))
                                                             (CL:WHEN (CL:/= (\\REFCNT USER::GC-ITEM)
                                                                          (CL:1- USER::OLD-REFCNT))
                                                                  (CL:ERROR "ASET to NIL doesn't decrement
                                                                         ref-count at ~D.~%" USER::I))))))))

(CL:DEFUN USER::XPOINTER-ARRAY-TESTS (USER::LIMIT)
   ;; Tests of arrays of XPOINTERs.
   (FOR USER::LOOP-NO FROM 1 TO USER::LIMIT
      COLLECT (CL:FORMAT T "  Starting xpointer-array tests, iteration ~D~%" USER::LOOP-NO)
            (FOR USER::MIN-LENGTH IN '(1 9 17 33 32768) AS USER::MAX-LENGTH
               IN '(8 16 32 32767 65535)
               DO (LET* ((USER::LEN (RAND USER::MIN-LENGTH USER::MAX-LENGTH))
                         (USER::ZERO-ARRAY (CL:MAKE-ARRAY USER::LEN :ELEMENT-TYPE 'XPOINTER :INITIAL-ELEMENT 0)
                                )
                         (USER::ONE-ARRAY (CL:MAKE-ARRAY USER::LEN :ELEMENT-TYPE 'XPOINTER :INITIAL-ELEMENT 1))
                         (USER::GC-ITEMS (LIST (CREATE FMTSPEC)
                                               100000 3.55 (CONS 3 4)
                                               (COMPLEX 3.4 5)
                                               4/5
                                               #'(CL:LAMBDA (USER::X)
                                                          (CL:PRINT (USER::DATE USER::X)))
                                               (CL:MAKE-ARRAY 5)))
                         USER::GC-ITEM USER::OLD-REFCNT)
                        (CL:FORMAT T "    Array size = ~D~%" USER::LEN)
                        (ERSETQ (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                       ((= USER::I USER::LEN))
                                    (CL:WHEN (CL:/= (CL:AREF USER::ZERO-ARRAY USER::I)
                                                    0)
                                          (CL:ERROR "**Zero-array wasn't zero at element ~d.~%" USER::I))
                                    (CL:WHEN (CL:/= (CL:AREF USER::ONE-ARRAY USER::I)
                                                    1)
                                          (CL:ERROR "**One-array wasn't one at element ~d.~%" USER::I))))
                        (ERSETQ (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                       ((= USER::I USER::LEN))
                                    (CL:SETF (CL:AREF USER::ZERO-ARRAY USER::I)
                                          (COND
                                             ((EVENP USER::I)
                                              1)
                                             (T 0)))
                                    (CL:WHEN (CL:/= (CL:AREF USER::ZERO-ARRAY USER::I)
                                                    (COND
                                                       ((EVENP USER::I)
                                                        1)
                                                       (T 0)))
                                          (CL:ERROR "EVENP pattern fails at ~D.~%" USER::I))))
                        ;; Make sure that putting a pointer to something into an array adds to the refcount.
                        (FOR USER::GC-ITEM IN USER::GC-ITEMS
                           DO (CL:SETQ USER::OLD-REFCNT (\\REFCNT USER::GC-ITEM))
                              (ERSETQ (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                             ((= USER::I USER::LEN))
                                          (CL:SETF (CL:AREF USER::ZERO-ARRAY USER::I)
                                                 USER::GC-ITEM)
                                          (OR (EQ (CL:AREF USER::ZERO-ARRAY USER::I)
                                                  USER::GC-ITEM)
                                              (CL:ERROR "Filling array with GC sample item failed at ~D.~%"
                                                     USER::I))
                                          (CL:WHEN (CL:/= (\\REFCNT USER::GC-ITEM)
                                                          USER::OLD-REFCNT)
                                                 (CL:ERROR "ASET bumps ref-count at ~D.~%" USER::I)))
                                    (CL:DO ((USER::I 0 (CL:1+ USER::I)))
                                           ((= USER::I USER::LEN))
                                         (CL:SETF (CL:AREF USER::ZERO-ARRAY USER::I)
                                               NIL)
                                         (OR (NOT (CL:AREF USER::ZERO-ARRAY USER::I))
                                             (CL:ERROR "Filling array with NIL failed at ~D.~%" USER::I))
                                         (CL:WHEN (CL:/= (\\REFCNT USER::GC-ITEM)
                                                         USER::OLD-REFCNT)
                                                (CL:ERROR "ASET to NIL decrements ref-count at ~D.~%" USER::I))
)))))))

;; Simple AREF & ASET of 1-, 2-, 3-d # arrays:

(DEFINEQ

(SIMPLE-AREF-ASET-TESTS
  (LAMBDA NIL                                                      ; Edited  9-Jun-88 19:02 by jds
    ;; Just run thru AREF and ASET on simple 1- 2- and 3-d arrays of numbers and make sure they look reasonable.
    (LET ((|array1d| (CL:MAKE-ARRAY '(10)
                            :INITIAL-CONTENTS
                            '(0 1 2 3 4 5 6 7 8 9)))
          (|array2d| (CL:MAKE-ARRAY '(3 10)
                            :INITIAL-CONTENTS
                            '((0 1 2 3 4 5 6 7 8 9)
                              (10 11 12 13 14 15 16 17 18 19)
```

```
                                     (20 21 22 23 24 25 26 27 28 29))))
          (|array3d| (CL:MAKE-ARRAY '(2 3 10)
                              :INITIAL-CONTENTS
                              '(((0 1 2 3 4 5 6 7 8 9)
                                 (10 11 12 13 14 15 16 17 18 19)
                                 (20 21 22 23 24 25 26 27 28 29))
                                ((100 101 102 103 104 105 106 107 108 109)
                                 (110 111 112 113 114 115 116 117 118 119)
                                 (120 121 122 123 124 125 126 127 128 129)))))
          (|array1d-0| (CL:MAKE-ARRAY '(10)
                                :INITIAL-ELEMENT "ASDF"))
          (|array2d-0| (CL:MAKE-ARRAY '(3 10)
                                :INITIAL-ELEMENT 3.5))
          (|array3d-0| (CL:MAKE-ARRAY '(2 3 10)
                                :INITIAL-ELEMENT
                                '|array3d-0|)))

         ;;  1 d array ref

         (|for| \i |from| 0 |to| 9 |do| (NEQP \i (CL:AREF |array1d| \i)
                                        '(CL:AREF |array1d| \i)))

         ;;  2 d array ref

         (|for| \j |from| 0 |to| 2 |do| (|for| \i |from| 0 |to| 9 |do| (NEQP (+ (TIMES \j 10)
                                                                            \i)
                                                                         (CL:AREF |array2d| \j \i)
                                                                         '(CL:AREF |array2d| \j \i))))

         ;; 3 d aref

         (|for| \k |from| 0 |to| 1 |do| (|for| \j |from| 0 |to| 2
                                          |do| (|for| \i |from| 0 |to| 9
                                                 |do| (NEQP (+ (TIMES \k 100)
                                                               (TIMES \j 10)
                                                               \i)
                                                            (CL:AREF |array3d| \k \j \i)
                                                            '(CL:AREF |array3d| \k \j \i)))))

         ;; 1 d array set

         (|for| \i |from| 0 |to| 9 |do| (CL:SETF (CL:AREF |array1d-0| \i)
                                        (DIFFERENCE 10 \i)))

         ;; 1 d array ref

         (|for| \i |from| 0 |to| 9 |do| (NEQP (DIFFERENCE 10 \i)
                                        (CL:AREF |array1d-0| \i)
                                        '(CL:AREF |array1d-0| \i)))

         ;; 2 d array set

         (|for| \j |from| 0 |to| 2 |do| (|for| \i |from| 0 |to| 9 |do| (CL:SETF (CL:AREF |array2d-0| \j \i)
                                                                         (PLUS \j (TIMES \i 10)))))

         ;; 2 d aref

         (|for| \j |from| 0 |to| 2 |do| (|for| \i |from| 0 |to| 9 |do| (NEQP (PLUS \j (TIMES \i 10))
                                                                         (CL:AREF |array2d-0| \j \i)
                                                                         '(CL:AREF |array2d-0| \j \i))))

         ;;  3 d array set

         (|for| \k |from| 0 |to| 1 |do| (|for| \j |from| 0 |to| 2
                                          |do| (|for| \i |from| 0 |to| 9 |do| (CL:SETF (CL:AREF |array3d-0| \k \j \i)
                                                                                   (PLUS \k (TIMES \j 10)
                                                                                         (TIMES \i 100))))))

         ;; 3 d aref

         (|for| \k |from| 0 |to| 1 |do| (|for| \j |from| 0 |to| 2
                                          |do| (|for| \i |from| 0 |to| 9
                                                 |do| (NEQP (PLUS \k (TIMES \j 10)
                                                                  (TIMES \i 100))
                                                            (CL:AREF |array3d-0| \k \j \i)
                                                            '(CL:AREF |array3d-0| \k \j \i))))))))

(NEQP
  (LAMBDA (A B ERROR-MSG)                                        ; Edited 12-Jun-88 18:13 by sybalsky

     ;; if  the two numbers A and B are not equal then halt with error message ERROR-MSG

    (OR (EQP A B)
        (ERROR ERROR-MSG))))
)

;; Test of past known failures

(CL:DEFUN USER::PAST-ARRAY-FAILURE-CASES (USER::LIMIT)

   ;; Repository for past known failure cases, gleened from hand tests, ARs, and failed runs of this test suite.

   (CL:FORMAT T "  Starting test of past failure syndromes.~%")
   (LET ((CL:ARRAY (CL:MAKE-ARRAY 57296 :ELEMENT-TYPE '(CL:UNSIGNED-BYTE 8)
                          :INITIAL-ELEMENT 1)))
        (CL:FORMAT T "   Test of array of 57296 (unsigned-byte 8)s inited to 1s.~%")
        (CL:DO ((USER::I 0 (CL:1+ USER::I)))
               ((= USER::I 57295))
            (CL:WHEN (CL:/= (CL:AREF CL:ARRAY USER::I)
                            1)
                   (CL:ERROR "Array of ones wasn't 1 at element ~D.~%" USER::I)))))
```

;; Assure that we compile with CL:COMPILE-FILE:

(PUTPROPS **MAIKO-ARRAY-TESTS FILETYPE** :COMPILE-FILE)

(PUTPROPS **MAIKO-ARRAY-TESTS COPYRIGHT** (NONE))

FUNCTION INDEX

PROPERTY INDEX