

File created: 6-Nov-91 16:12:40 {DSK}<python>RELEASE>loops>2.0>patches>LMPATCH.;2

changes to: (FUNCTIONS |SubclassResponsibility|)

previous date: 6-Nov-91 15:53:14 {DSK}<python>RELEASE>loops>2.0>patches>LMPATCH.;1

Read Table: XCL

Package: INTERLISP

Format: XCCS

; Copyright (c) 1991 by Venue. All rights reserved.

(RPAQQ LMPATCHCOMS
(

;;; Patches courtesy of Tom Lipkis @ Savoir

;; This one makes STREQUAL run MUCH faster in LOOPS; it checks for equality from the END of strings first.

```
(COMS (FNS |FastSTREQUAL|)  
      (DECLARE\ : DONTVAL@LOAD DOCOPY (P (MOVD? 'STREQUAL 'OLDSTREQUAL)  
                                         (MOVD ' |FastSTREQUAL| 'STREQUAL))))
```

;; These use COMPILER-LET instead of MACROLET to handle the LOOPS method inheritance macros; it conses a LOT less as a result

```
(FUNCTIONS LOOPS-FUNCALL |Method| |_Super| |_Super?| |_SuperFringe| |SubclassResponsibility|)
```

;; These improve copying performance; CopyDeepDescr is iterative instead of straight recursive, CopyInstance used to force UID creation
;; which was making LoopsWindow active values stay around even after their Window was collected

```
(FNS |CopyDeepDescr| |CopyInstance|)
```

;; Improves performance dumping instance files

```
(METHODS |Object.SaveInstance?|))
```

;;; Patches courtesy of Tom Lipkis @ Savoir

;; This one makes STREQUAL run MUCH faster in LOOPS; it checks for equality from the END of strings first.

(DEFINEQ

(|FastSTREQUAL|

```
(LAMBDA (X Y) ; Edited 10-Jun-88 16:16 by TAL  
  (DECLARE (LOCALVARS . T))  
  (AND (STRINGP X)  
       (STRINGP Y)  
       (PROG ((LEN (|ffetch| (STRINGP LENGTH) |of| X)))  
             (COND  
              ((NEQ LEN (|ffetch| (STRINGP LENGTH) |of| Y))  
               (RETURN)))  
              (RETURN (PROG ((BASEX (|ffetch| (STRINGP BASE) |of| X))  
                             (BNX (|ffetch| (STRINGP OFFST) |of| X))  
                             (FATPX (|ffetch| (STRINGP FATSTRINGP) |of| X))  
                             (BASEY (|ffetch| (STRINGP BASE) |of| Y))  
                             (BNY (|ffetch| (STRINGP OFFST) |of| Y))  
                             (FATPY (|ffetch| (STRINGP FATSTRINGP) |of| Y)))  
              (COND  
               ((OR (NEQ 0 BNX)  
                    (NEQ 0 BNY)  
                    FATPX FATPY)  
                (GO SLOWLP)))  
              LP (COND  
                 ((EQ 0 LEN)  
                  (RETURN T)))  
                 (|add| LEN -1)  
                 (COND  
                  ((NEQ (\\GETBASEBYTE BASEX LEN)  
                       (\\GETBASEBYTE BASEY LEN))  
                   (RETURN)))  
                 (GO LP)  
              SLOWLP  
              (COND  
               ((EQ 0 LEN)  
                (RETURN T))  
               ((NEQ (\\GETBASECHAR FATPX BASEX BNX)  
                    (\\GETBASECHAR FATPY BASEY BNY))  
                (RETURN)))  
              (T (|add| BNX 1)  
                 (|add| BNY 1)  
                 (|add| LEN -1)  
                 (GO SLOWLP))))))))
```

)

(DECLARE\ : DONTVAL@LOAD DOCOPY

(MOVD? 'STREQUAL 'OLDSTREQUAL)

```
(MOVD ' |FastSTREQUAL| ' STREQUAL)
)
```

:: These use COMPILER-LET instead of MACROLET to handle the LOOPS method inheritance macros; it conses a LOT less as a result

```
(DEFMACRO LOOPS-FUNCALL (FN &REST ARGS)
  ;; The optimizer for this doesn't make sure FN is evaluated first, but that's ok for loops and save a GENSYM and binding
  `(APPLY* ,FN ,@ARGS))

(DEFDEFINER (|Method| (:NAME (CL:LAMBDA (|method-body|)
                                (CL:MULTIPLE-VALUE-BIND (|class-name| |selector|)
                                                          (PARSE-METHOD-BODY |method-body|)
                                                          (|MethName| |class-name| |selector|))))))
  METHOD-FNS (&WHOLE |method-body|)
  (CL:MULTIPLE-VALUE-BIND (|class-name| |selector| |args| |declarations| |forms| |doc| |qualifiers|
                          |method-type|)
    (PARSE-METHOD-BODY |method-body|)
    (CL:ASSERT (|Class?| ($! |class-name|)))
    (LET (|function-name| |function-type| |body|)
      ;; Compute the name of the function
      (SETQ |function-name| (|MethName| |class-name| |selector|))
      ;; Compute the type of the function
      (SETQ |function-type| (OR (LISTGET |qualifiers| :FUNCTION-TYPE)
                               :IL))
      (CL:CHECK-TYPE |function-type| (CL:MEMBER :CL :IL))
      ;; Get the body of the function, with the top level comments removed
      (SETQ |body| `(CL:COMPILER-LET ((|*ArgsOfMethodBeingCompiled*| ',|args|)
                                     (|*ClassNameOfMethodOwner*| ',|class-name|)
                                     (|*SelectorOfMethodBeingCompiled*| ',|selector|)
                                     (|*SelfOfMethodBeingCompiled*| ',(CAR |args|)))
                          ,. |forms|))
      ;; Build the function definition form
      `(PROGN ,(CL:ECASE |function-type|
                  (:CL `(CL:DEFUN ,|function-name| ,|args|
                          ,@|declarations| ,|body|))
                  (:IL `(DEFINEQ (,|function-name| (LAMBDA ,|args|
                                                      ,@|declarations|
                                                      ,|body|))))
                (INSTALL-METHOD-FN ',|class-name| ',|selector| ',|function-name| ',(CDR |args|)
                                     ',|doc|
                                     ',|function-name|))))))

(DEFMACRO |Super| (&REST |Send-Super-Args|)
  (DECLARE (CL:SPECIAL |*ArgsOfMethodBeingCompiled*| |*ClassNameOfMethodOwner*|
            |*SelectorOfMethodBeingCompiled*| |*SelfOfMethodBeingCompiled*|))
  (COND
    ((NULL |Send-Super-Args|)
     ; Args default to args of the method
     `(LOOPS-FUNCALL (|FindSuperMethod| ,|*SelfOfMethodBeingCompiled*| ',|*SelectorOfMethodBeingCompiled*|
                     (LOADTIMECONSTANT (|OldClass| ,|*ClassNameOfMethodOwner*|)))
                     ,. |*ArgsOfMethodBeingCompiled*|))
    ((NEQ |*SelectorOfMethodBeingCompiled*| (CADR |Send-Super-Args|))
     ; Selectors must match
     (ERROR "Selector to _Super does not match method selector" (CADR |Send-Super-Args|)))
    ((NEQ |*SelfOfMethodBeingCompiled*| (CAR |Send-Super-Args|))
     ; Self must match
     (ERROR "Can't _Super to other than first arg of method" (CDR |Send-Super-Args|)))
    (T
     ; Args differ
     (APPEND `(LOOPS-FUNCALL (|FindSuperMethod| ,|*SelfOfMethodBeingCompiled*| '
                             |*SelectorOfMethodBeingCompiled*|
                             (LOADTIMECONSTANT (|OldClass| ,|*ClassNameOfMethodOwner*|)))
              ,|*SelfOfMethodBeingCompiled*|
              (CDDR |Send-Super-Args|))))))

(DEFMACRO |Super?| (&REST |Send-Super-Args|)
  (DECLARE (CL:SPECIAL |*ArgsOfMethodBeingCompiled*| |*ClassNameOfMethodOwner*|
            |*SelectorOfMethodBeingCompiled*| |*SelfOfMethodBeingCompiled*|))
  (COND
    ((NULL |Send-Super-Args|)
     ; Args default to args of the method
     `(LOOPS-FUNCALL (|FindSuperMethod| ,|*SelfOfMethodBeingCompiled*| ',|*SelectorOfMethodBeingCompiled*|
                     (LOADTIMECONSTANT (|OldClass| ,|*ClassNameOfMethodOwner*|))
                     (FUNCTION NIL))
                     ,. |*ArgsOfMethodBeingCompiled*|))
    ((NEQ |*SelectorOfMethodBeingCompiled*| (CADR |Send-Super-Args|))
     ; Selectors must match
     (ERROR "Selector to _Super does not match method selector" (CADR |Send-Super-Args|)))
    ((NEQ |*SelfOfMethodBeingCompiled*| (CAR |Send-Super-Args|))
     ; Self must match
     (ERROR "Can't _Super to other than first arg of method" (CDR |Send-Super-Args|)))
    (T
     ; Args differ
     (APPEND `(LOOPS-FUNCALL (|FindSuperMethod| ,|*SelfOfMethodBeingCompiled*| '
                             |*SelectorOfMethodBeingCompiled*|
                             (LOADTIMECONSTANT (|OldClass| ,|*ClassNameOfMethodOwner*|))
                             (FUNCTION NIL))
              ,|*SelfOfMethodBeingCompiled*|
              (CDDR |Send-Super-Args|))))))
```

```
(ERROR "Can't _Super? to other than first arg of method" (CDR |Send-Super-Args|))
(T
  (APPEND `(LOOPS-FUNCALL (|FindSuperMethod| ,|*SelfOfMethodBeingCompiled*| '
                        |*SelectorOfMethodBeingCompiled*|
                        (LOADTIMECONSTANT (|OldClass| ,|*ClassNameOfMethodOwner*|)
                        (FUNCTION NIL))
                        ,|*SelfOfMethodBeingCompiled*|)
    (CDDR |Send-Super-Args|))))
```

```
(DEFMACRO |SuperFringe| (&REST |Send-Super-Args|)
  (DECLARE (CL:SPECIAL |*ArgsOfMethodBeingCompiled*| |*ClassNameOfMethodOwner*|
            |*SelectorOfMethodBeingCompiled*| |*SelfOfMethodBeingCompiled*|))
  (COND
    ((NULL |Send-Super-Args|)
     ; Args default to args of the method
     `(|for| |cls| |in| (|fetch| |localSupers| |of| (LOADTIMECONSTANT (|OldClass| ,|*ClassNameOfMethodOwner*|)))
       |do| (LOOPS-FUNCALL (OR (|FetchMethod| |cls| ' ,|*SelectorOfMethodBeingCompiled*|)
                             (FUNCTION NIL))
                           ,.|*ArgsOfMethodBeingCompiled*|)))
    ((NEQ |*SelectorOfMethodBeingCompiled*| (CADR |Send-Super-Args|))
     ; Selectors must match
     (ERROR "Selector to _Super does not match method selector" (CADR |Send-Super-Args|))
     ((NEQ |*SelfOfMethodBeingCompiled*| (CAR |Send-Super-Args|))
      ; Self must match
      (ERROR "Can't _SuperFringe to other than first arg of method" (CDR |Send-Super-Args|)))
    (T `(|for| |cls| |in| (|fetch| |localSupers| |of| (LOADTIMECONSTANT (|OldClass| ,|*ClassNameOfMethodOwner*|)))
        |bind| (|argList| _ (MAPCAR '(, (CAR |Send-Super-Args|)
                                   ,.(CDDR |Send-Super-Args|))
                                  (FUNCTION EVAL)))
        |do| (APPLY (OR (|FetchMethod| |cls| ' ,|*SelectorOfMethodBeingCompiled*|)
                      (FUNCTION NIL))
                  |argList|))))
```

```
(DEFMACRO |SubclassResponsibility| ()
  (DECLARE (CL:SPECIAL |*ArgsOfMethodBeingCompiled*| |*ClassNameOfMethodOwner*|
            |*SelectorOfMethodBeingCompiled*| |*SelfOfMethodBeingCompiled*|))
  `(HELPCHECK (CONCAT "Method " ,|*SelectorOfMethodBeingCompiled*| " of class " ,|*ClassNameOfMethodOwner*|
                    " needs to be defined in class ")
    (_ ,|*SelfOfMethodBeingCompiled*| |ClassName|)))
```

:: These improve copying performance; CopyDeepDescr is iterative instead of straight recursive, CopyInstance used to force UID creation which was making LoopsWindow active values stay around even after their Window was collected

(DEFINEQ

```
(|CopyDeepDescr|
  (LAMBDA (|descr| |newObjAList|) ; Edited 14-Jun-88 12:52 by TAL
    (DECLARE (LOCALVARS . T))
    ;; Copies instances active values and lists, but bottoms out on anything else
    (SELECTQ (TYPENAME |descr|)
      (|instance| (OR (CDR (FASSOC |descr| |newObjAList|))
                    (_ |descr| |CopyDeep| |newObjAList|)))
      (|annotatedValue|
       (|create| |annotatedValue|
        |annotatedValue| _ (|CopyDeepDescr| (|fetch| |annotatedValue| |of| |descr|)
                                             |newObjAList|)))
      (LISTP (|bind| \t2 |val| |for| |valTail| |on| |descr|
                 |do| (COND
                      (\t2 (FRPLACD \t2 (SETQ \t2 (LIST (|CopyDeepDescr| (CAR |valTail|)
                                                                           |newObjAList|))))
                      (T (SETQ |val| (SETQ \t2 (LIST (|CopyDeepDescr| (CAR |valTail|)
                                                                           |newObjAList|))))))
                (COND
                 ((AND (CDR |valTail|)
                      (NLISTP (CDR |valTail|)))
                  (FRPLACD \t2 (|CopyDeepDescr| (CDR |valTail|)
                                                |newObjAList|))))
                |yield| |val|))
      |descr|)))
```

```
(|CopyInstance|
  (LAMBDA (|oldInstance|) ; Edited 16-Sep-88 17:26 by TAL
```

::; make a new instance with the same contents as self, or copy into an instance if given

```
(LET ((|newInstance| (_ (|Class| |oldInstance|)
                       |CreateInstance|)))
```

::; Creating UID for copy loses big. E.g., AVs as default IV value in class generally have UID. When IV is first accessed, AV is copied and stored in instance. If copy has UID it will never go away, and in the case of LispWindowAV this causes the window, bitmap, stream, etc. to stay around also.

```
##(COND ((AND (fetch OBJUID of oldInstance) (NULL (fetch OBJUID of newInstance))) (* ; "Old one not temporary, but new one has non OBJUID yet")
        (UID newInstance)))#
```

;; Copy IVSource down one layer of list structure.

```
(|FillIVs| |newInstance| (|Class| |oldInstance|)
  (MAPCAR (|IVSource| |oldInstance|)
    (FUNCTION APPEND)))
|newInstance|))
```

)

;; Improves performance dumping instance files

```
(|\\BatchMethodDefs|)
```

```
(METH |Object| |SaveInstance?| (|file| |outInstances|)
  "Save this instance if referred to by another unless it is already on this list to be saved"
  (|category| (|Object|)))
```

```
(|Method| ((|Object| |SaveInstance?|) |self| |file| |outInstances|)
```

; edited: 26-Oct-84 15:36

"Save this instance if referred to by another unless it is already on this list to be saved"

```
(COND
  ((|type?| |instance| |self|)
  (NOT (FMEMB |self| |outInstances|)))
  (T (NOT (MEMBER |self| |outInstances|))))))
```

```
(|\\UnbatchMethodDefs|)
```

```
(PUTPROPS LMPATCH COPYRIGHT ("Venue" 1991))
```

FUNCTION INDEX

|CopyDeepDescr|3 |CopyInstance|3 |FastSTREQUAL|1

MACRO INDEX

LOOPS-FUNCALL2 |_Super|2 |_SuperFringe|3
|SubclassResponsibility|3 |_Super?|2

DEFINER INDEX

|Method|2
