June 20, 2006

# Design++ 6.0 Release Notes for Windows 2000/XP/2003

We are pleased to announce the release of Design++ 6.0 for Windows 2000/XP/2003. Design++ 6.0 includes multiple new modules and enhancements, the most important ones being the following.

## HIGHLIGHTS

### AUTODRAFTER

AutoDrafter provides functionality for automatic generation of CAD drawings from a Design++ model. AutoDrafter creates a drawing model that represents a set of 2D CAD drawings. Each drawing set contains one or more projected views of the then active 3D CAD model. The views can be annotated with text, symbols, dimensions, labels, and lines. All views and annotations are controlled by rules so the drawings can dynamically change in response to model changes.

### REPORTWRITER

ReportWriter is used for creating custom reports from a Design++ model. Report data can be formatted to be viewed by a user or imported into other programs such as estimating or material management systems.  ReportWriter comes with a set of sorting, totaling, sub totaling, and filtering tools. Both ASCII and XML formats are supported. Report templates can be saved along with a project for repeated use.

### QUERYTOOL

QueryTool allows ad hoc queries into Design++ models and libraries.  Queries are specified with regular expression based search filters.  Queries return model/library information matching search filter specifications. Query results can be operated upon, saved, and printed.

### LICENSE MANAGER

The new license usage monitoring capability allows organizations to keep better track of the use of their floating, networked Design++ licenses. For example, the license usage log can be used to analyze the peek usage during a certain time period, or the number of Design++ access denials due to lack of licenses.

Also, network license performance over a slow network connection has been improved significantly.

### DESIGN RULE EDITOR

Design Rule Editor (DRE) is integrated more closely with the rest of the Developer's Interface (UIP). For example, DRE now opens automatically in the vicinity of the location from where the edit request was issued. Also, the 'Context Sensitive Menu' dialog is integrated with the main DRE window. To support rule development, rule compilation warnings and error messages are now shown in an easy-to-notice dialog. Rules are also indented automatically.

## PROJECT PATH HANDLING ENHANCED

Projects can now be loaded from anywhere regardless of the initial project path setting. Also, both the project and license paths can be specified using UNC.

## NEW DESIGN RULE MACROS

New configuration macros, :create and :relate, simplify the syntax for *substructure*, *relations*, and role attribute rules. They also allow partial instantiation of substructure and relation descriptions by isolating any calculation errors or delays from the rest of the rule.
Other new macros include :local-value and :instances.

## CHANGE MANAGEMENT

Change management has been optimized significantly. Also, design rule tracing and error tracking capabilities have been improved.

## COMPONENT CREATION OPTIMIZED

Component creation has been optimized for instantiating large number (>100+) of components of the same class under a single assembly.

## ODBC LINK

ODBC (RDB) link has been optimized significantly for retrieving large amount of data with a single query. The larger the amount retrieved, the bigger the improvement.

## COM/API LINK

Message strings are now allocated dynamically removing an artificial upper bound for message length. Previously messages over 500 KB could cause dppCOMserver to crash with a stack overflow.

## SUPPORTED CAD VERSIONS

AutoCAD: 2000, 2000i, 2002, 2004, 2005, and 2006.
MicroStation: V7, V8 2004 Edition, and V8 XM Edition.
Visio: 2000, 2000 SR1, 2002, and 2003.

## WINDOWS SERVER 2003 SUPPORT

In addition to Windows 2000 and XP, Design++ 6.0 is now also

```
      supported on Windows Server 2003.

--------------------------------------------------
KNOWN LISP ISSUES WHEN PORTING FROM D++ 5.0 TO 6.0
--------------------------------------------------

Here are known Lisp compatibility issues to take into account when
porting a Design++ 5.0 (ACL 6.1 based) application to Design++ 6.0
(ACL 7.0 based). These issues result from Franz's continued effort to
clean up ACL's remaining non-conformances with the ANSI standard. If
you run into other Lisp related issues not covered here, just let us
know.

* LOOP: do nil else
  -----------------
  ERROR:
  ;;; Compiling file h:\d++\code-acl\infix\rule-macro.lisp
  ; While compiling EXPAND-ALL-VALUE-REFERENCES-FOR-INFIX-PREFIX:
  Error: Compound form expected, but found NIL.
  Current LOOP context: DO NIL ELSE.
    [condition type: PROGRAM-ERROR]

  CODE:
  (loop for ref in refs
      for ref-exp = nil
      if (member ref vars)
      do nil         ;;do nothing, it's ok to use assigned
      else if ...
      do ...)

  EXPLANATION:
  The syntax of LOOP requires a _compound_ form -- a symbol (e.g. NIL)
  at the top level of a loop must be parsed as a go tag.  Otherwise
  there would be ambiguity for symbols that might have symbol-macrolet
  definitions with expansions that have side effects.

  SOLUTION:
  (loop for ref in refs
      for ref-exp = nil
      if (member ref vars)
      do (progn)     ;;do nothing, it's ok to use assigned
      else if ...
      do ...)

* FBOUNDP: invalid function spec
  ------------------------------
  ERROR:
  Error: #:THE-COMPONENT33905 is not a valid function spec
    [condition type: TYPE-ERROR]

  CODE:
  (special-operator-p (caar form))

  EXPLANATION:
  From Release Notes for Allegro CL 7.0:
  fboundp now errors when passed an invalid argument, function-name-p
  test whether argument is a valid name. fboundp was out of spec in
```

earlier releases in that it returned nil rather than signaling an error when passed an argument which was not a valid function name or specification. (Thus (fboundp 3) returned nil rather than signaling an error.) fboundp now signals an error when passed an invalid argument. The new function function-name-p returns true if its argument is a valid function spec (and thus a suitable argument to fboundp) and returns nil otherwise. (if (function-name-p spec) (fboundp spec)) thus behaves in release 7.0 as (fboundp spec) did in earlier releases.

SOLUTION:
```
#+(and allegro (not (version>= 7 0)))
(special-operator-p (caar form)))
#+(and allegro (version>= 7 0))
(if (excl:function-name-p (caar form))
    (special-operator-p (caar form)))
```

* ASSOC: non-cons ALIST elements
------------------------------
ERROR:
```
Attempt to take the car of D which is not listp.
[condition type: TYPE-ERROR]
```

CODE:
```
(in-package :d)
(setf xyz '((a b)(c) d))
(defun test ()
  (declare (special xyz))
  (format t "(assoc 'a xyz) ~A~%" (assoc 'a xyz))
  (format t "(assoc 'c xyz) ~A~%" (assoc 'c xyz))
  (format t "(assoc 'd xyz) ~A~%" (assoc 'd xyz)))

D++(8): (compile 'test)
TEST
NIL
NIL
D++(9): (test)
(assoc 'a xyz) (A B)
(assoc 'c xyz) (C)
Error: Attempt to take the car of D which is not listp.
  [condition type: TYPE-ERROR]

Restart actions (select using :continue):
 0: Return to Top Level (an "abort" restart).
 1: Abort entirely from this (lisp) process.
[1] D++(10): :reset
```

EXPLANATION:
According to Common Lisp specification ASSOC function ignores NILs but considers any other non-cons ALIST element to be an error. Since the ALIST in the test case contains a non-cons symbol D, Lisp correctly signals an error.

The reason why the compiled test case doesn't fail in D++ 5.x is that with the most optimized compiler settings Franz skips the non-cons check in ASSOC. But, for the ACL compiler version in D++ 6.0 Franz has further reduced the amount of error checks when the most optimized compiler settings are turned on. This has led into other sequence related problems in D++. To avoid these problems, we are now enabling certain error checks regardless of the compiler settings.

Anyway, one way to avoid having to go through all ASSOC calls in your code is to define your own customized ASSOC version that would ignore all non-cons ALIST elements, instead of signaling an error.

Below is an ASSOC version that should work for you. See the test cases below the function definition.
Note that we are not overriding ACL's ASSOC function, instead we are defining a customized version named MY-ASSOC. You may want to name it to conform to your function naming standards.

Simply paste the code below to one of your project function files and replace all ASSOC calls within your code with a call to your customized ASSOC.

Even though, the customized ASSOC won't be as efficient as the ACL implementation, it probably does not have any overall performance implications. Still, we would recommend going through all your ASSOC calls at some point and fix them so that they could be reverted back to calling ACL's ASSOC.

SOLUTION:

```
(in-package :design++)

;;; Function MY-ASSOC (item alist &key (:test #'eql) :test-not (:key #'identity))
;;; Tka, 10-Mar-05
;;; A special version of the ASSOC function which ignores all non-cons
;;; elements of ALIST. (Common Lisp's ASSOC ignores nils but considers
;;; any other non-cons ALIST element to be an error.)
;;; Note that the argument :TEST-NOT is not implemented as it is
;;; recommended to be deprecated.
(defun my-assoc (item alist &key (test #'eql) test-not (key #'identity))
  (declare (ignore test-not))
  (cond ((null alist) nil)
        ((not (consp (car alist))) ;Not a cons -> skip
         (my-assoc item (cdr alist) :test test :key key))
        ((funcall test (funcall key (caar alist)) item)
         (car alist))
        (t (my-assoc item (cdr alist) :test test :key key))))

#||
;;;Test cases from Common Lisp HyberSpec. For more info, see
;;;http://www.lispworks.com/documentation/HyperSpec/Body/f_assocc.htm

(setq values '((x . 100) (y . 200) (z . 50))) =>  ((X . 100) (Y . 200) (Z . 50))
(my-assoc 'y values) =>  (Y . 200)
(rplacd (my-assoc 'y values) 201) =>  (Y . 201)
(my-assoc 'y values) =>  (Y . 201)
(setq alist '((1 . "one")(2 . "two")(3 . "three"))) =>  ((1 . "one")
              (2 . "two") (3 . "three"))
(my-assoc 2 alist) =>  (2 . "two")
;;(my-assoc-if #'evenp alist) =>  (2 . "two")
;;(my-assoc-if-not #'(lambda(x) (< x 3)) alist) =>  (3 . "three")
(setq alist '(("one" . 1)("two" . 2))) =>  (("one" . 1) ("two" . 2))
```

```
  (my-assoc "one" alist) =>  NIL
  (my-assoc "one" alist :test #'equalp) =>  ("one" . 1)
  (my-assoc "two" alist :key #'(lambda(x) (char x 2))) =>  NIL
  (my-assoc #\o alist :key #'(lambda(x) (char x 2))) =>  ("two" . 2)
  (my-assoc 'r '((a . b) (c . d) (r . x) (s . y) (r . z))) =>   (R . X)
  (my-assoc 'goo '((foo . bar) (zoo . goo))) =>  NIL
  (my-assoc '2 '((1 a b c) (2 b c d) (-7 x y z))) =>  (2 B C D)
  (setq alist '(("one" . 1) ("2" . 2) ("three" . 3))) =>  (("one" . 1)
              ("2" . 2) ("three" . 3))
  ;;(my-assoc-if-not #'alpha-char-p alist :key #'(lambda (x) (char x
  ;;              0))) => ("2" . 2)
  ;;Special tests for non-cons alist elements 10-Mar-05 by tka.
  (setq alist '((a b) d (c))) => ((A B) D (C))
  (my-assoc 'a alist) => (A B)
  (my-assoc 'c alist) => (C)
  (my-assoc 'd alist) => NIL
  (setq alist '((a b) NIL (c))) => ((A B) NIL (C))
  (my-assoc 'a alist) => (A B)
  (my-assoc 'c alist) => (C)
  (my-assoc NIL alist) => NIL
  ||#

* SUBSEQ: end is beyond the end of the sequence
  ---------------------------------------------
  ERROR:
  `end' is beyond the end of the sequence.

  CODE:
  A call to Lisp subseq with an end that exceeds the length of the
  sequence now (with ACL 7.0) generates an error, but it didn't in
  previous version (with ACL 6.1). For example:
  For example:
     (subseq '(a b c d e) 0 10)
  used to return
     (a b c d e)

  Now it returns:
     Error: `end' is beyond the end of the sequence.
       [condition type: SIMPLE-ERROR]
     Restart actions (select using :continue):
      0: Return to Top Level (an "abort" restart).
      1: Abort entirely from this process.
     [1]

  EXPLANATION:
  Until ACL 6.1 SUBSEQ had a bug that allowed the "correct" behaviour.
  The bug was fixed in a patch to ACL 6.2 (on Fri Jan 9 15:12:51 PST 2004).

  SOLUTION:
  Go through your SUBSEQ calls to make sure that END index is within the
  bounds of SEQUENCE.

------------------
DESIGN RULE MACROS
------------------

* Parsing of Design Rules with local location clauses
```

```
  (:local-right-of, :local-above, etc.) fixed to properly return the
  referenced attributes also when the rule fails.

* New Design Rule macro :LOCAL-VALUE introduced.

  ;|| :LOCAL-VALUE ()
  ;||  PURPOSE:
  ;||   Provides access to the current local value of a component's
  ;||   attribute which is being (re)determined by its design rule.
  ;||  ARGUMENTS:
  ;||    NONE
  ;||  RETURNS:
  ;||   The current local value of a component's attribute which is being
  ;||   (re)determined.
  ;||  EXAMPLE:
  ;||   (:! process-plant nr-of-all-walls
  ;||     (format t "~&Current local value: ~A~%" (:local-value))
  ;||     (length (:? self all-walls)))
  ;||
  ;||   ==>
  ;||
  ;||   D++(13):
  ;||   ; --> Redetermining local value of PROCESS-PLANT's NR-OF-ALL-WALLS...
  ;||   Current local value: 4
  ;||   ; PROCESS-PLANT's NR-OF-ALL-WALLS: New value: 8; Old value: 4
  ;||   D++(14):
  ;||

* Design rule macro :LOCAL-ROTATION modified to accept
  ROTATIONS also as <a list of rotations> in addition to as
  <multiple individual rotations>. Thus, the following
  examples will now work.

  (:! transformer geo_rot ;;EXPRESSION (list of multiple rotations)
      (:local-rotation (:?1 wall) ((:x 45) (:z 45))))

  (:! transformer geo_rot ;;LOCAL BINDING (list of rotations)
      (let ((rot '((:y 45) (:z 30))))
        (:local-rotation (:?1 wall) rot)))

* A new optional argument MEMBER-DESCENDANTS-P added for design
  rule macros :N, :LAST, :ANY, and their derivatives like,
  :?1,...,?10 and :FIRST,...,:TENTH. The new argument, which
  defaults to T, determines whether to consider all instances
  that are descendants of the CLASS or just the direct
  instances of the CLASS while searching for the component.

  The default behavior of these macros, when the optional
  argument ASSEMBLY was given, was as if MEMBER-DESCENDANTS-P
  was set to T. If ASSEMBLY was not given, they behaved as if
  MEMBER-DESCENDANTS-P was set to NIL. Thus, the behavior of
  these macros remains the same when the optional argument
  ASSEMBLY is given. When the ASSEMBLY is not given, the
  behavior changes only in the rare case where a class and its
  subclass are both instantiated under the same assembly. For
  example,
  Library: PARTS--A--B
  Model:
           /--A.1
          /---A.2
  ASSEMBLY----A.3
          \---B.1
           \--B.2
```

```
* New Design Rule macro :CREATE introduced for making *substructure*
  and role attribute rules more readable. The :CREATE macro not only
  simplify the syntax for *substructure* and role attribute rules but
  allows partial instantiation of substructure descriptions by
  isolating any calculation errors or delays from the rest of the
  rule.

  ;|| :CREATE (class-or-role &optional (nr 1))
  ;||   TKa, 24-Apr-03
  ;||  PURPOSE:
  ;||   Simplifies the syntax for *substructure* and role attribute
  ;||   rules, and allows partial instantiation of substructure
  ;||   descriptions by isolating any calculation errors or delays from
  ;||   the rest of the rule.
  ;||  ARGUMENTS:
  ;||   class-or-role:
  ;||    A class of component or a role to be created (class or role
  ;||    name)
  ;||   nr (optional): Number of components to be created (non-negative
  ;||    integer, defaults to NIL). Note that if the argument nr is not
  ;||    provided, the attribute NR-<class name>, if it exists, is
  ;||    checked for the value. Otherwise user is prompted for the
  ;||    value.
  ;||  RETURNS:
  ;||   A substructure description.
  ;||  EXAMPLE1:
  ;||   :create macro simplifies the *substructure* rules, just compare
  ;||   these 2 rules producing the same substructure description.
  ;||
  ;||   (:! wall substructure        (:! wall substructure
  ;||       (:create door 1)             (list (list :N 1 'door)
  ;||       (:create window 1))                (list :N 1 'window)))
  ;||
  ;||  EXAMPLE2:
  ;||   Just like with other design rule macros class-or-role can be
  ;||   referenced by name, symbol, or expression.
  ;||
  ;||   (:! wall substructure
  ;||    (let ((nr 2)
  ;||          (window 'window))
  ;||      (:create window nr)))
  ;||   ==>
  ;||   ; --> Redetermining local value of WALL's SUBSTRUCTURE...
  ;||   ; WALL's SUBSTRUCTURE: New value: (:N 2 WINDOW);
  ;||                          Old value: ((:N 1 DOOR)(:N 1 WINDOW))
  ;||
  ;||  EXAMPLE3:
  ;||   If the last expression in a rule is a :create expression (or the
  ;||   rule would return nil), then all :create expressions are
  ;||   combined to form the final substructure description. All other
  ;||   expression are ignored.
  ;||
  ;||   (:! wall substructure
  ;||       (:create door 1)
  ;||       '(:n 2 window) ;IGNORED
  ;||       (:create window 3))
  ;||   ==>
  ;||   ; --> Redetermining local value of WALL's SUBSTRUCTURE...
  ;||   ; WALL's SUBSTRUCTURE: New value: ((:N 1 DOOR) (:N 3 WINDOW));
  ;||                          Old value: (:N 2 WINDOW)
  ;||
  ;||  EXAMPLE4:
  ;||   If the last expression in a rule is not a :create expression,
  ;||   then all :create expressions are ignored and the last expression,
  ;||   unless NIL, is used as the substructure description.
  ;||
  ;||   (:! wall substructure
  ;||       (:create door 3)   ;IGNORED
  ;||       (:create window 3) ;IGNORED
  ;||       '(:n 2 window))
  ;||   ==>
  ;||   ; --> Redetermining local value of WALL's SUBSTRUCTURE...
  ;||   ; D's SUBSTRUCTURE: New value: (:N 2 WINDOW);
  ;||                       Old value: ((:N 1 DOOR) (:N 3 WINDOW))
  ;||
  ;||  EXAMPLE5:
  ;||   Even if one or more :create expressions fail or is delayed, the
  ;||   rest of the rule will proceed allowing partial instantiation of
  ;||   the substructure.
  ;||
  ;||   (:! wall substructure
  ;||       (:create door 1)
  ;||       (:create window (/ 1 0)) ;Attempt to divide by zero
  ;||       (:create window 1))
  ;||   ==>
  ;||   ; --> Redetermining local value of WALL's SUBSTRUCTURE...
  ;||   ; WALL's SUBSTRUCTURE: Rule execution warning: Attempt to divide 1 by zero.
  ;||   ; WALL's SUBSTRUCTURE: New value: ((:N 1 DOOR) (:N 1 WINDOW));
  ;||                          Old value: (:N 2 WINDOW)
  ;||
  ;||  EXAMPLE6:
  ;||   The use of :create macro is not tied to *substructure* rules
  ;||   only; it works equally well with any attributes, like role
  ;||   attributes.
  ;||
  ;||   (:! wall substructure
  ;||       (:create some-doors-and-windows))
  ;||   ==>
  ;||   ; --> Redetermining local value of WALL's SUBSTRUCTURE...
  ;||   ; WALL's SUBSTRUCTURE: New value: (:R SOME-DOORS-AND-WINDOWS);
  ;||                          Old value: ((:N 1 DOOR) (:N 1 WINDOW))
  ;||
  ;||   (:! wall some-doors-and-windows
  ;||       (:create door 1)
  ;||       (:create window 2))
  ;||   ==>
```

```
  ;||   ; WALL's SOME-DOORS-AND-WINDOWS --> Using rule...
  ;||   ; WALL's SOME-DOORS-AND-WINDOWS: ((:N 1 DOOR) (:N 2 WINDOW))
  ;||
  ;||  EXAMPLE7:
  ;||   Even though :create macro is intended to be used in
  ;||   *substructure* or role attribute rules, it does return a proper
  ;||   substructure description even if called in other rules or
  ;||   evaluated as such.
  ;||
  ;||   D++(59): (:create door 2)
  ;||   (:N 2 DOOR)
  ;||   D++(60):
  ;||
```

```
* Design Rule relation access macros fixed to create proper
  dependencies even when the model does not contain any instances of
  the type specified by the access macro's optional CLASS argument.

  This fix affects all relation access macros, like :PARTS,
  :ALL-PARTS, :ASSEMBLY and their derivatives, but only when called
  with the optional CLASS argument and MEMBER-DESCENDANTS-P (defaults
  to T) set to NIL.

* New Design Rule macro :RELATE introduced for making *relations*
  attribute rules more readable. The :RELATE macro not only simplify
  the syntax for *relations* rules but allows partial implementation
  (relating) of relation descriptions by isolating any calculation
  errors or delays from the rest of the rule.
```

```
  ;|| :RELATE (relation from-components to-components)
  ;||   TKa, 02-Jun-03
  ;||  PURPOSE:
  ;||   Simplifies the syntax for *relations* attribute rules, and
  ;||   allows partial instantiation of relation descriptions by
  ;||   isolating any calculation errors or delays from the rest of the
  ;||   rule.
  ;||   [from-component] --[relation]--> [to-component]
  ;||   [to-component] --[inverse-relation]--> [from-component]
  ;||  ARGUMENTS:
  ;||   relation:
  ;||    Relation with which to relate the components (symbol)
  ;||   from-components:
  ;||    <component> | (<component>*) | (<component> <rel-init-data>) |
  ;||    ((<component> <rel-init-data>)*)
  ;||   to-components:
  ;||    <component> | (<component>*) | (<component> <rel-init-data>) |
  ;||    ((<component> <rel-init-data>)*)
  ;||  RETURNS:
  ;||   A relations description.
  ;||  EXAMPLE1:
  ;||   :relate macro simplifies the *relations* rules, just compare
  ;||   these 3 rules all producing the same substructure description.
  ;||
  ;||   (:! wall relations
  ;||       (:relate :mounted-on (list (:? door) (:? window)) self))
  ;||
  ;||   (:! wall relations
  ;||       (:relate :mounted-on (:? door) self)
  ;||       (:relate :mounted-on (:? window) self))
  ;||
  ;||   (:! wall relations
  ;||       (list (list :mounted-on (:? door) self)
  ;||             (list :mounted-on (:? window) self)))
  ;||
  ;||  EXAMPLE2:
  ;||   Relation initialization data can be associated with the
  ;||   components to be related. See GeometricModelingMadeEasy document
  ;||   for more information on using relation initialization data.
  ;||
  ;||   (:! wall relations
  ;||       ;;Mount window to the center of wall
  ;||       (:relate :mounted-on (:? window) self)
  ;||       ;;Align door's bottom face with wall's bottom face and
  ;||       ;;then rotate the door 180 degrees around X axis.
  ;||       (:relate :mounted-on
  ;||           (list (:? door) :face 'bottom)
  ;||           (list self :face 'bottom :X 180)))
  ;||
  ;||  EXAMPLE3:
  ;||   Just like with other design rule macros, relation,
  ;||   from-components, and to-components can be referenced by name,
  ;||   symbol, or expression.
  ;||
  ;||   (:! wall relations
  ;||       (let* ((door (:? door))
  ;||            (window (:? window))
  ;||            (door&window (list door window)))
  ;||         (:relate :mounted-on door&window self)))
  ;||
  ;||  EXAMPLE4:
  ;||   If the last expression in a rule is a :relate expression (or the
  ;||   rule would return nil), then all :relate expressions are
  ;||   combined to form the final relations description. All other
  ;||   expression are ignored.
  ;||
  ;||   (:! wall relations
  ;||       (:relate :mounted-on (:? door) self)
  ;||       (list :mounted-on (:? door) self) ;IGNORED
  ;||       (:relate :mounted-on (:? window) self))
  ;||
  ;||  EXAMPLE5:
  ;||   If the last expression in a rule is not a :relate expression,
  ;||   then all :relate expressions are ignored and the last
  ;||   expression, unless NIL, is used as the relations description.
  ;||
  ;||   (:! wall relations
  ;||       (:relate :mounted-on (:? door) self) ;IGNORED
  ;||       (:relate :mounted-on (:? window) self) ;IGNORED
  ;||       (list :mounted-on (:? door) self))
  ;||
  ;||  EXAMPLE6:
  ;||   Even if one or more :relate expressions fail or is delayed, the
  ;||   rest of the rule will proceed allowing partial instantiation of
  ;||   relation descriptions.
```

```
  ;||
  ;||   (:! wall relations
  ;||       (:relate :mounted-on (:? door) self)
  ;||       (:relate :mounted-on (:? window) ssselfff) ;An unknown component
  ;||       (:relate :mounted-on (:? window) self))
  ;||
  ;||  EXAMPLE7:
  ;||   Even though :relate macro is intended to be used in *relations*
  ;||   attribute rules, it does return a proper relations description
  ;||   even if called in other rules or evaluated as such.
  ;||
  ;||   D++(17): (:relate :mounted-on (:? door) (:?1 wall))
  ;||   (:MOUNTED-ON #<FRAME DOOR.S1652 TKA> #<FRAME WALL.S1496 TKA>)
  ;||   D++(18):
  ;||
```

* Parsing of design rule macros :create and :relate modified to allow
  them to be used in functions that are then called from design
  rules. For example, instead of

```
  ------------------------------
  (:! BUNCH-O-LINES SUBSTRUCTURE
      (:create line-test 3))
  ------------------------------
```
  you can now implement the same with the help of a function
```
  ------------------------------
  (defun bunch-o-lines-substructure ()
    (:create line-test 3))

  (:! BUNCH-O-LINES SUBSTRUCTURE
      (bunch-o-lines-substructure))
  ------------------------------
```

------------------------------
DESIGN++ FUNCTIONS (LISP/API)
------------------------------

* New plane related Design++ Functions introduced, namely
    DPP-PLANE-DISTANCE-TO-ORIGIN DPP-PLANE-NORMAL-VECTOR
    DPP-POINT-FROM-3-PLANES DPP-SAME-PLANES-P DPP-PARALLEL-PLANES-P
    DPP-PLANE-FROM-PLANE-AND-OFFSET

```
  ;|| DPP-PLANE-DISTANCE-TO-ORIGIN (plane)
  ;||  PURPOSE:
  ;||   Returns the distance to origin of a plane.
  ;||  ARGUMENTS:
  ;||   plane
  ;||    A plane (a list of x, y, z and distance to origin)
  ;||  RETURNS:
  ;||   The distance to origin of the plane.
  ;||  EXAMPLE:
  ;||   (dpp-plane-distance-to-origin '(-1 1 0 2.0))
  ;||   ==> 2.0
  ;||

  ;|| DPP-PLANE-NORMAL-VECTOR (plane)
  ;||  PURPOSE:
```

```
;||   Returns the normal vector of a plane.
;||  ARGUMENTS:
;||   plane
;||    A plane (a list of x, y, z and distance to origin)
;||  RETURNS:
;||   The normal vector (a list of x, y and z)
;||  EXAMPLE:
;||   (dpp-plane-normal-vector '(-1 1 0 2.0))
;||   ==> (-1 1 0)
;||

;|| DPP-POINT-FROM-3-PLANES (p1 p2 p3)
;||  PURPOSE:
;||   Returns a point at the intersection of the three planes.
;||  ARGUMENTS:
;||   p1, p2, p3:
;||    Planes (each a list of x, y, z and distance to origin)
;||  RETURNS:
;||   Point of intersection (list of x, y and z), or NIL if
;||   planes do not intersect at a point.
;||  EXAMPLE:
;||   (dpp-point-from-3-planes '(-1 1 0 2) '(1 1 0 2) '(0 0 1 4))
;||   ==> (0.0 2.0 4.0)
;||

;|| DPP-SAME-PLANES-P (p1 p2)
;||  PURPOSE:
;||   Verifies whether two planes are the same
;||  ARGUMENTS:
;||   p1, p2:
;||    Planes (Each a list of x, y, z components of the normal and distance to origin)
;||  RETURNS:
;||   T if planes coincide, otherwise NIL
;||  EXAMPLE:
;||   (dpp-same-planes-p '(1 1 1 0) '(1.0 1.0 1.0 0.0))
;||   ==> T
;||

;|| DPP-PARALLEL-PLANES-P (p1 p2)
;||  PURPOSE:
;||   Determines whether planes are parallel or not.
;||  ARGUMENTS:
;||   p1, p2:
;||    Planes (Each a list of x, y, z components of the normal and distance to origin)
;||  RETURNS:
;||   T (true) if parallel, otherwise NIL (false).
;||  EXAMPLE:
;||   (dpp-parallel-planes-p '(1 1 1 0) '(1.0 1.0 1.0 1.0))
;||   ==> T
;||

;|| DPP-PLANE-FROM-PLANE-AND-OFFSET (p1 offset)
;||  PURPOSE:
;||   Returns a plane that is an offset distance from a plane.
;||  ARGUMENTS:
;||   p:
;||    A planes (a list of x, y, z components of the normal and distance to origin)
;||   offset:
;||    Distance between the original plane and the next.
;||  RETURNS:
;||   Plane (a list of x, y, z components of the normal and distance to origin)
;||   , or NIL if p is NIL.
;||  EXAMPLE:
;||   (dpp-plane-from-plane-and-offset '(0 0 0 0.0) 1.0)
;||   ==> (0.0 0.0 0.0 1.0)
;||
```

* User function DPP-PROMPT-FOR-FILE superseded with DPP-SELECT-PATH,
  Which allows better control of the file or directory selection
  process

```
;|| DPP-SELECT-PATH (&key prompt title directory file-pattern
;||                      file-pattern-prompt exists-p directory-p)
;||  PURPOSE:
;||   Prompts the user to select path
;||  ARGUMENTS:
;||   Key:
;||    prompt (string):
;||      A String to prompt user, defaults to "Choose a file"
;||    title (string):
;||      A dialog title, default is "Choose a file"
;||    exists-p (T/NIL)
;||      Select only existing file. With NIL, user can type the file name.
;||      Default is T.
;||    multiple-p (T/NIL)
;||      Select multiple files (not directories). The files are returned as list of strings.
;||      Default is NIL.
;||    directory-p (T/NIL):
;||      Select directory instead of a file. Default is NIL.
;||    initial-path (string):
;||      A directory from which to display files, defaults to project directory.
;||    initial-file (string):
;||      The default file to select.
;||    file-pattern (string):
;||      Wildcard expression describing wanted file. Default is "*.*"
;||    file-pattern-prompt (string):
;||      Defines the informative text which is associated to the the file-pattern.
;||      Default is "All Files".
;||  RETURNS:
;||    The full pathname string of the selected file or list of strings for multiple files.
;||  NOTE:
;||    The intended :native argument is disabled as in the native dialog in Galaxy 3.0
;||    cancel actions don't call the cancel hook and so UIS doesn't know if native
;||    dialog was canceled.
;||  EXAMPLE:
;||    (dpp-select-path :prompt "Specify a temporary file" :title "Select temporary file"
;||                     :file-pattern "*.tmp" :file-pattern-prompt "Tmp File"
;||                     :exists-p nil :initial-path "C:\\temp\\" :initial-file "temp.tmp")
;||     ==> "C:\\temp\\temporary.tmp"
;||    (dpp-select-path :multiple-p t)
;||     ==> ("D:\\d++60\\projects\\geo_test\\externals\\nr_1.lisp"
;||          "D:\\d++60\\projects\\geo_test\\externals\\nr_7.lisp")
;||    (dpp-select-path  :directory-p t)
;||     ==> "D:\\d++60\\projects\\geo_test\\externals"
```

```
  ;||
```

* New user function DPP-PROJECTS-PATHNAME returns the current Design++ projects directory path specified by DPPPROJECTS environment variable.

```
  ;|| DPP-PROJECTS-PATHNAME ()
  ;||  PURPOSE:
  ;||   To returns DPPPROJECTS pathname.
  ;||  RETURNS:
  ;||   Current DPPPROJECTS pathname as a string
  ;||  EXAMPLE:
  ;||   (dpp-projects-pathname)
  ;||  ==> "D:\\d++-projects\\"
```

* New user function DPP-SET-PROJECTS-PATHNAME changes the Design++ projects directory path specified by DPPPROJECTS environment variable.

```
  ;|| DPP-SET-PROJECTS-PATHNAME (path)
  ;||  PURPOSE:
  ;||   To set DPPPROJECTS pathname.
  ;|| ARGUMENTS:
  ;||  path:
  ;||   New path for Design++ projects (string)
  ;||  RETURNS:
  ;||   New DPPPROJECTS pathname as a string
  ;||  EXAMPLE:
  ;||   (dpp-set-projects-pathname "D:\\d++-projects")
  ;||  ==> "D:\\d++-projects\\"
```

* Unambiguous component reference generation for Design++ Functions DPP-WRITE-EXTERNAL and DPP-WRITE-EXTERNAL-FOR-COMPONENTS fixed to handle correctly the case where a class and its subclass are both instantiated under the same assembly. For example,

```
         Library: PARTS--A--B
         Model:
           /--A.1
          /---A.2
  ASSEMBLY----A.3
          \---B.1
           \--B.2

  ;|| DPP-WRITE-EXTERNAL (&optional (model-name (dpp-get-current-model)) output-file)
  ;||  PURPOSE:
  ;||   Generates an external data file from the model MODEL-NAME. The
  ;||   model is traversed in depth-first order starting from ROOT. For
  ;||   each component the values of the attributes listed in
  ;||   external_attributes attribute (list of symbols) are written to
  ;||   OUTPUT-FILE.
  ;||  ARGUMENTS:
  ;||   model-name (optional):
  ;||    Model name, default = current model (symbol)
  ;||   output-file (optional):
  ;||    Output file name, default = nil (symbol or string)
  ;||  RETURNS:
```

```
;||   NIL
;||  EXAMPLE:
;||   (dpp-write-external)

;|| DPP-WRITE-EXTERNAL-FOR-COMPONENTS (component-list &optional output-file
;||                                    (model-name (dpp-get-current-model)))
;||  PURPOSE:
;||   Generates an external data file for the components in
;||   COMPONENT-LIST. For each component the values of the attributes
;||   listed in EXTERNAL_ATTRIBUTES (EXTERNAL-ATTRIBUTES and
;||   EXTERNAL.ATTRIBUTES are synonyms) attribute (list of symbols)
;||   are written to OUTPUT-FILE.
;||  ARGUMENTS:
;||   component-list (list of frames or component names):
;||    List of components for which the external data file is written
;||    for.
;||   output-file (optional):
;||    Output file name, default = prompt user (symbol or string)
;||   model-name (optional):
;||    Model name, default = current model (symbol)
;||  RETURNS:
;||   Filename if successful, otherwise NIL
;||  EXAMPLE:
;||   (dpp-write-external-for-components '(floor.s559 floor.s3133))
;||   ==> "/home/code/d++/projects/plant/externals/tka-new-ext.lisp"
;||
```

* Design Rule relation access macros fixed to create proper
  dependencies even when the model does not contain any instances of
  the type specified by the access macro's optional CLASS argument.

  This fix affects all relation access macros, like :PARTS,
  :ALL-PARTS, :ASSEMBLY and their derivatives, but only when called
  with the optional CLASS argument and MEMBER-DESCENDANTS-P (defaults
  to T) set to NIL.

* Default values for two keyword arguments for Design++ Function
  DPP-TRACE-RULES changed. Now, :WITH-VALUES defaults to T and
  :WITH-PRETTY-NAMES defaults to NIL. These are the preferred values
  for most Design++ users.

```
;|| DPP-TRACE-RULES (&key (with-values t)
;||                      (with-pretty-names nil)
;||                       (with-cycle-detection nil)
;||                       (with-statistics nil)
;||                      (components-of-class :all))
;||  PURPOSE:
;||  Enables Design Rule tracing with different trace options
;||  ARGUMENTS:
;||   with-pretty-names (keyword):
;||    Whether or not the rules are traced with components' pretty names or real
;||    names (T or NIL (default))
;||   with-values (keyword):
;||    Whether or not the trace should also contain attribute values
;||    (T (default) or NIL )
;||   with-cycle-detection (keyword):
;||    Whether or not to look for infinite cycles that cannot converge
;||    because of inconsistent design rules, typically an application
;||    error, (T or NIL (default))
;||   with-statistics (keyword):
;||    Whether or not to collect change propagation statistics (T or
;||    NIL (default))
;||   components-of-class (keyword):
;||    List of classes whose descendant components' rules are to be
;||    traced. Default value :all means that all rules of all instances
;||    are to be traced. (list of frame-or-ref or :all (default).
;||  RETURNS:
;||   NIL
;||  EXAMPLE:
;||   (dpp-trace-rules :with-values t :components-of-class '(pump steel))
;||   ==> T
;||
```

* A new keyword argument :EXIT-FN added to DEF-EXTERNAL-SERVER macro,
  which is used for defining a C/API client to be an external server
  for Design++.

```
;||   exit-fn (function, default: nil):
;||      A server-specific function to exit the actual server
;||      program. Exit-fn is called by kill-<server-name> without any
;||      arguments. The user-defined exit-fn is expected to exit the
;||      server gracefully.
```

* Index allocation and unique name generation optimized for component
  creation. These optimization becomes noticeable only when
  instantiating large number (>100x) of components of the same class
  under a single assembly.

  Also, Design++ function DPP-ADD-COMPONENT optimized
  significantly for creation on large number of components of
  the same class.

```
;|| DPP-ADD-COMPONENT (class-name assembly
;||                 &optional (nr_comp 1) (model-name (dpp-get-current-model))
;||                           (update-gui-p t))
;||  PURPOSE:
;||   Creates new components to an assembly in a model
;||  ARGUMENTS:
;||   class-name (symbol):
;||    Component's class name
;||   assembly:
;||    Assembly (parent) component in model (frame-or-ref)
;||   nr_comp (integer):
;||    Number of components to be created, default = 1
;||   model-name (symbol):
;||    Model of new component(s), default = current
;||   update-gui-p (T/NIL):
;||    T (default) if user interface is to be updated, otherwise NIL
;||  RETURNS:
;||   Created components (list of frames)
;||  EXAMPLE:
;||   (dpp-add-component 'floor 'parking_structure.s73)
;||   ==> (#<Frame: FLOOR.S82 EXAMPLE>)
;||
;||   (dpp-add-component 'floor 'parking_structure.s73 2)
```

```
  ;||   ==> (#<Frame: FLOOR.S83 EXAMPLE> #<Frame: FLOOR.S84 EXAMPLE>)
  ;||
```

* New Design++ Function DPP-SIMPLIFY-GEO-ROT introduced.

```
  ;||  DPP-SIMPLIFY-GEO-ROT (geo-rot)
  ;||  PURPOSE:
  ;||   Simplifies a list of rotations given in 'geo-rot' format.
  ;||  ARGUMENTS:
  ;||   A list of rotations around major coordinate axes in 'geo-rot'
  ;||   format.
  ;||  RETURNS:
  ;||   A simplified list of rotations around major coordinate axes in
  ;||   'geo-rot' format.
  ;||  EXAMPLE:
  ;||   (dpp-simplify-geo-rot
  ;||    `((:Z -90.0) (:X 90.0) (:Y 90) (:Z 180.0) (:X 90)))
  ;||   ==> ((:X 0.0))
  ;||
```

-------------
DESIGN++ CORE
-------------

* Reading in external data source files fixed to maintain the order of component-attribute-value triplets with indirect component referencing. This optimizes the value retrieval for components with many siblings all with a separate component-attribute-value triplet (indirect component referencing) in the data source.

  Value retrieval further optimized by categorizing component-attribute-value triplets with indirect component referencing also by the triplets' assemblies.

* Parsing of substructure descriptions of form (:N NIL <component> <substructure>*) fixed to consider NIL as <number> and treat it as if it were 0.

* Validity checking of substructure descriptions of form (:N NIL <component> <substructure>*) modified to consider NIL as <number>. Thus, descriptions of this form are now considered valid.

* A number of elusive and unreproducible garbage-collector bugs fixed. Some of the bugs were related to various combinations of sys:resize-areas usage and/or the open-old-area-fence gsgc-parameter.

* A workaround provided for a Windows bug where, under rare circumstances, a socket duplication system call can fail to return a valid socket handle. This problem occurs primarily on Windows XP Home.

* Pretty name generation fixed to handle properly ambiguous frames, that is, multiple frames (in different KBs) with the same name.

* Error in displaying Design++ splash screen during system startup fixed. This was a very rare and hard to reproduce problem which

```
  seemed to occur only on Windows XP.

* Online documentation access for Lisp functions fixed to point to the
  new Common Lisp HyperSpec web site at
  http://www.lispworks.com/reference/HyperSpec/index.html

  Online documentation (PDF) for all Design++ and Lisp functions and
  design rule macros is directly accessible from Command Interpreter
  (Emacs shortcut <ctrl>-c <ctrl>-f) and from Design Rule Editor.

* Evaluating a design rule definition in the Command Interpreter (or
  otherwise) modified NOT to convert the rule string to a lowercase
  string when storing the rule to a component's attribute. This is to
  assure that the cases of potential strings within the rule itself
  are retained.

* Design++ Console window's buffer size modified to be
  user-settable. The buffer size can be set with a new Design++
  command line argument +<number>, where <number> is base 10 and must
  be >= 1,000. The buffer's default maximum size is increased from
  25,000 bytes to 100,000 bytes.

  The new command line argument +<number> will set the maximum size of
  the Console window's buffer to <number> bytes, with the "shrinkage
  factor" set to 15% of <number>. The shrinkage factor is the amount
  that the buffer will shrink by when the maximum size is reached.

  To customize the Console window's buffer size you need to edit your
  Design++ startup batch file <d++>\d++.bat. Look for the following
  lines towards the end of the file.
  ------------
  :WITHOUT-EMACS
  rem No icon on tray, close Console after exiting, new title, show splash for 3 sec
  "%DPPBIN%/bg" \"%DPPBIN%/%DPPIMAGENAME%.exe\" +R +M +t \"Design++ Console\" +Bt -I
\"%DPPIMAGEPATH%\"
  ------------
  For example, to increase the buffer size to 1 MB you would
  have to edit the above line as follows.
  ------------
  :WITHOUT-EMACS
  rem No icon on tray, close Console after exiting, new title, show splash for 3 sec
  rem Set the console window's buffer size to 1,000,000 bytes
  "%DPPBIN%/bg" \"%DPPBIN%/%DPPIMAGENAME%.exe\" +R +M +1000000 +t \"Design++ Console\" +Bt -I
\"%DPPIMAGEPATH%\"
  ------------

* Parsing of Design Rules fixed to handle correctly Lisp special form
  environments within rules.

* Design Rule parsing modified to treat a bound symbol as a variable
  even if the symbol is not formally declared as a special
  variable. This modification allows variables to be introduced (for
  rule macros) by simply setting a value for a symbol. This is
  identical to how symbols are treated in Design++ 5.x and earlier
  versions. For example,
  ----------------
  D++(19): (:? floor)
  #<FRAME FLOOR.S1485 TKA>
  D++(20): (setf my-floor *)
  #<FRAME FLOOR.S1485 TKA>
  ;;Following works as the bound symbol is treated as a variable
  D++(21): (:assembly my-floor)
  #<FRAME BUILDING.S1478 TKA>
  D++(22):
  ----------------
```

* Socket interface fixed to handle the case where a socket server is
  exited explicitly before the client has had a change to
  unregister. The problem could result in error, for example, if the
  client failed to handle the exit message correctly.

  Fixed also the case where the client is exited explicitly after it
  has already unregistered.

* Internal object data structure transformation threshold optimized
  for new ACL compiler settings.

* Franz ACL 7.0 :STREAMA module added. You can verify that the module
  :STREAMA is properly loaded by evaluating (require :streama) which
  should return NIL if the module is loaded.

* Creating a single component fixed to make sure that each of the
  inherited attributes with dependencies are properly inherited.

----------------------------------
DYNAMIC CONFIGURATION ENHANCEMENTS
----------------------------------

* Redetermination optimized to delay the redetermination of attributes
  of those components whose assemblies are marked for dynamic
  instantiation. This reduces unnecessary redeterminations within the
  components that will be deleted along with their assemblies by the
  dynamic instantiation.

* Cycle detection during redetermination modified not to report cycles
  if the dynamic instantiation is expected to change the model
  structure. Since the cycle could be a result of the model not being
  in a consistent state, the attribute's determination is delayed
  until after the next dynamic instantiation.

* Internal data structure for redetermination optimized.

* Change management and dynamic instantiation for the :removal
  strategy fixed to create and maintain a model fully expanded.

* Attribute determination cycles during redetermination and dynamic
  instantiation revisited. As the cycle is likely transitional
  resulting from the model not being in a consistent state, the user
  is not consulted but the attribute is marked for a later
  redetermination.

* Design rule macros :PARTS and :ALL-PARTS (and their derivatives)
  optimized to delay *parts* referring rules if the assembly is
  scheduled for instantiation. This prevents unnecessary rule

(re)determinations based on invalid *parts*.

* Handling of transitional design rule errors during change propagation modified to exit all the rules in the call stack and delay the (re)determination of the initiating rule. This improves the error handling as the NIL returned by the failing rule is no longer propagated up the call stack.

* Handling of transitional errors caused by re-referencing an attribute that is already marked for later (re)determination fixed. Also, design rule failure reporting enhanced to identify the actual erred rule (component & attribute), which caused the failure.

* Handling of transitional errors caused by reference cycles fixed. Also, design rule failure reporting enhanced to print out the reference cycle, which caused the failure.

* Several sequence functions reimplemented to be more tolerant of improper arguments. This will improve the overall fault tolerance against transitional errors during change propagation when the model is in an inconsistent state. Functions affected are elt, length, make-list, map1, butlast, nbutlast, and nth. Note that there will be a minor performance penalty for the added argument checking.

* Change management for the predefined MOUNTED-ON relation fixed and optimized. The MOUNTED-ON relation automates the details of inter-component positioning and orientation.

* Retrieving external data source values explicitly within a design rule, e.g. by calling DPP-FROM-EXTERNAL, fixed to allow the calling rule stack to be delayed if a data source entry refers to the *parts* of an assembly which is scheduled for instantiation.

* Reporting real attribute redetermination failures fixed to report each failed attribute only once.

* Design Rule macro :RULECALL fixed to allow the calling rule to be delayed if the rule it calls errs or is delayed.

* Dependency creation optimized to collect referenced attributes only if a design rule macro is called from within a rule.

* Delaying of a *parts*-referring design rule fixed to delay the calculation only if the assembly exists. This prevents the rule calculation from being delayed unnecessarily in case the (virtual) assembly is never created.

* Printing of the removal propagation warning about a depending component, which is not in the current model, modified not to print the warning if the missing component has already been deleted.

* The removal of *relations* attribute's value fixed to properly disconnect existing relations.

* Dynamic instantiation fixed to handle the case where the local value of an assembly's *substructure* attribute (determined by a rule) is identical with its inherited value.

* Attribute value removal fixed to delay change propagation until the value is properly removed. This is especially important for handling properly the case where the removed local value is identical with the inherited value.

* Dynamic instantiation fixed to avoid instantiating a *relations* spec more than once. In certain situations the relations were instantiated multiple times resulting in duplicate relations.

* Dynamic instantiation fixed to avoid processing instantiation requests more than once.

* Change propagation modified to mark all new depending attributes for redetermination before actually processing any of them. Knowing all the pending redeterminations in advance allows them to be processed in an order that reduces unnecessary redeterminations.

* Redetermination failure reporting modified NOT to report a redetermination that was simply delayed until the components it refers to were created.

* Handling of transitional design rule errors during change propagation modified NOT to delay the (re)determination of the special *component-init* and *substructure-init* rules OR any other rules triggered by their determination. These are initialization rules that by definition should complete immediately after a component is created or a substructure is expanded. This is especially important as these rules are mostly used for side effects and not so much for their actual value.

* :create and :relate design rule macros fixed to correctly delay an attribute's redetermination when the macros refer to parts that have not yet been instantiated.

------------

INSTALLATION

------------

* JAVA is now an installation component including JRE, JAVA/API and RelationBrowser and it is not installed in compact setup. If Java is not installed, then when needed Design++ tries to use Java from Windows registry or from PATH.

* English is the only supported language option. Support for Finnish has been removed.

* Internal pathname representation has been switched to proper Windows pathnames. That is, backslashes "\", not forward slashes "/" are used to separate components in pathnames. Still, any legal combination of the two slashes should work.

* d++.bat file environment variables modified.

  DPPLICENSETYPE
    Entry 'fixed' renamed to 'standalone', which is also used by 'Standalone Server' option.

```
  DPPWITHEMACS
    Was previously DPPNOEMACS. If its value equals 'yes' or not
    defined (default), then Design++ is started with Emacs. If it
    equals 'no', then Design++ is started without Emacs.

  DPPEMACSWINMODE
    Previously if it was defined, then Design++ Emacs Windows mode
    (winmode) was used. Now winmode is used if its value value equals
    'yes'.

  DPPWITHDEVGUI
    Was previously DPPNOGUI. If it equals 'yes' or not defined
    (default), then Design++ development GUI programs (UIP & DRE) are
    started at Design++ startup.  If it equals 'no', then development
    GUIs are not started.

  DPPREMOTECLIENTSOK
    If its value equals 'yes', then Design++ accepts connections from
    remote CAPI clients, like pre 6.0. If not defined (default) or
    equals 'no', then only local CAPI client connections are accepted.

  DPPNOCAD
    Not used anymore, instead use DPPCAD=disabled

---------------
LICENSE MANAGER
---------------

* Floating license performance on Windows XP, NT, and 2000 networks
  improved by using an asynchronous notification mechanism.

* Design++ license manager revisited:

  1. Installation of license service drivers streamlined for Design++
     License manager update utility. For example, the separate DOS
     window prompting whether or not the user wants to proceed is
     eliminated.

  2. License manager dialog enhanced to show also the current license
     manager version.

  3. Parsing of start times for currently active network licenses in
     the license manager dialog fixed to take into account daylight
     saving time.

* Design++ licensing mechanism revisited:

  1. The handling of losing a license unexpectedly, e.g. due to a
     network connection problem, modified to provide a 5-minute grace
     period during which the user can save any unsaved work. The grace
     period starts only after the user has acknowledged a dialog
     informing about the lost license. Design++ exits after the grace
     period unless the license is restored.

  2. License Manager dialog's 'Status' button renamed to 'Refresh'.
```

3. New License Usage Monitoring capability introduced.

   License usage monitoring capability allows organizations to keep better track of the use of their floating, networked Design++ licenses. For example, the license usage log can be used to analyze the peek usage during a certain time period, or the number of Design++ access denials due to lack of licenses.

   License usage monitoring complements the dynamic license usage utility, which shows the current status of active users at any given time. Even though the license usage monitoring is most beneficial for network license server administrators, the license usage log is collected also for standalone licenses.

   All usage information is collected to a usage log file on the license server; one line for every session on every client workstation. For network licensing, the usage log is collected to file %DPPLICENSEPATH%\DP-license-usage-log\DP-license-usage.log

   For a workstation with a standalone license, the usage log is collected to file
   %DPP%\misc\crpk\DP-license-usage-log\DP-license-usage.log

   The DP-license-usage-log subdirectory can also contain several temporary session-specific log files. To assure uninterrupted usage monitoring, it's very important not to delete or edit any of these files.

   For viewing the license usage log there is a new a 'Design++ License Usage Log' dialog which can be accessed from the 'Design++ License Manager' dialog. The usage log dialog has basic selection and sorting capabilities.

   For each Design++ session the following information is stored to the license usage log file. All items are written on a single line and are separated by the "|" character. The usage log is in text format.

```
<Workstation Name> as string
<Worksation IP Address> in dotted decimal format
<User Name> as string
<Product Name> as string
<Local Time Zone> as integer
<Session Start Time (GMT)> as string
<Session Start Time (GMT)> as integer
<Session License Status> one of
  OK            - License requested successfully
  OK_AfterWait  - All licenses in use; waited until one became
                  available
  NotAvailable  - All licenses in use; exited before one
                  became available
  NotAuthorized - Design++ not authorized to run on this
                  workstation/server
<Session End Time (GMT)> as string
<Session End Time (GMT)> as integer
```

   Time zone is represented as a number of hours offset from

Greenwich Mean Time (GMT). Time zone values increase with motion to the west and decrease to the east. For example, Pacific Standard Time (PST) is 8 and Central European Time (CET) is -1.

The session start and end times are recorded in GMT, also known as Universal Time (UTC). Times are recorded both in a human readable format and also as an absolute time represented by a single, non-negative integer (the number of seconds since midnight, January 1, 1900 GMT). The absolute time format is more convenient for any utilities displaying and analyzing the usage information.

* New Windows Server 2003 compatible license manager released. There is one drawback; the Design++ 5.0 and 6.0 license managers are not compatible. That is, once you have upgraded to Design++ 6.0, existing Design++ 5.0 installations will not start without reauthorization and vice versa.

  To allow existing 5.0 installation to share licenses with 6.0, a intermediate Design++ version 5.2 was released. 5.2 images were built off the same source code as 5.0 images except for the changes required by the new license manager. Upgrading to the new license manager by upgrading to 5.2 should be a smooth process as there are no other major changes in the Design++ itself.

  If you are interested in the new license manager for your existing Design++ 5.0 installation, please contact your Design Power representative for the Design++ 5.2 upgrade ZIP file. The installation is easy; simply extract all the files from the Design++52Upgrade.ZIP file starting from your top-level <d++50> directory.

* License usage monitoring revisited:

  1. Updating session and master usage log files modified to detect and warn if the user does not have proper write access to the usage log directory.

  2. Categorizing of the different files (master log, session log, others) in the usage log directory improved.

  3. Parsing usage log entries made more robust.

  4. Checking for orphaned session usage logs modified to be performed only when the user explicitly requests to see the 'Usage Log' from the 'License Manager' dialog.

  5. For network-licensed clients, the updating of session usage log optimized to write the file first on the local hard drive and then copy it to the license server in a non-blocking background process.

  6. For network-licensed clients, the updating of master usage log optimized to update the master log from the local session log so that there is no need to copy the local log file to the license server first.

* New Design++ Function DPP-LICENSE-CHECK-ORPHANED-SESSION-LOGS
  introduced.

```
;|| DPP-LICENSE-CHECK-ORPHANED-SESSION-LOGS ()
;||  PURPOSE:
;||   Checks for orphaned session usage logs, i.e., usage logs of
;||   sessions that have exited abnormally. If found, the master usage
;||   log is updated with the usage information from the orphaned
;||   session log and the orphaned session log is deleted.
;||  ARGUMENTS:
;||  RETURNS:
;||   NIL
;||  EXAMPLE:
;||   (dpp-license-check-orphaned-session-logs)
;||   ==> NIL
;||
```

* New command line arguments introduced for Design++ license manager
  update utility, <d++>\misc\crpk\dppUpdateLicensemanager.exe. This
  update utility can be used to update or reinstall the license
  manager. It will update or install all the required license manager
  files.

  The new command line arguments for dppUpdateLicensemanager are:

```
  -Path <dpplicensepath>
    Specifies the location for the Design++ network license
    directory. Defaults to <d++>\misc\crpk\
  -Uninstall
    Uninstalls the Design++ license manager, but does not remove any
    local licenses.
  -Silent
    Runs the update utility without displaying any dialogs.
```

* License Manager revisited:

  1. Network license performance over a slow network connection
     improved by running license authorization in a separate,
     non-blocking process.

  2. Network license performance further improved by copying the usage
     information updates to the license server in a separate,
     non-blocking process.

  3. The handling of DPPLICENSEPATH enhanced to accept more variety in
     the path setting. For example, assuming that

     - License path on server BASEBALL is c:\dpp_license
     - Partition L: is mapped to \\baseball\c
     - Partition N: is mapped to \\baseball\c\dpp_license

     then all of the following DPPLICENSEPATH settings now work OK.

```
DPPLICENSEPATH=\\baseball\c\dpp_license (UNC, no mapping required)
DPPLICENSEPATH=L:\dpp_license
DPPLICENSEPATH=N:\
DPPLICENSEPATH=N:/
DPPLICENSEPATH=N:
```

4. Initial authorization modified to make sure that the license
   service, either standalone or on the license server, is up and
   running.

5. A license manager hang-up problem caused by starting external
   Design++ processes, like GUI and CAD, while a license manager
   call was in progress is now fixed.

6. New Design++ Function DPP-LICENSE-DEF-AUTHORIZATION-AFTER-METHOD
   introduced.

```
;|| DPP-LICENSE-DEF-AUTHORIZATION-AFTER-METHOD (method)
;||  PURPOSE:
;||  Defines an after method to be called after the initial Design++
;||  license authorization has been verified. This is to allow
;||  applications to complete licensing related initializations. Note
;||  that the method needs to be defined in the application image. It
;||  is too late to define it as part of the application's startup.
;||  ARGUMENTS:
;||   method:
;||    A function or a symbol with function binding to be called AFTER
;||    the initial license authorization has been verified. Method is
;||    called with no arguments.
;||  RETURNS:
;||   Method function
;||  EXAMPLE:
;||   (dpp-license-def-authorization-after-method
;||    #'(lambda ()
;||        (format t "~&This is the license authorization after method.~%")
;||        (format t "~&License related application initializations would come next.~%")))
;||   ==>
;||   ;   Starting Design++...
;||   ;   Requesting a license...
;||   This is the license authorization after method.
;||   License related application initializations would come next.
;||   OK
;||   ;   Registering UI Server
```

* Optimized network licensing modified to start the initial
  authorization early enough so that the license information is
  available by the time an application is loaded in
  <d++>\misc\d++-startup.lisp.

* Network licensing fixed to handle correctly those external clients
  that registered while the initial authorization was still in
  progress.

* Checking for the license server status modified to take into account
  that the user may not have proper access rights to perform the
  check. This is especially true if the license server runs on Windows
  Server 2003, which, since SP1, has significantly restricted access
  rights for remote users.

* Problem in copying the session (local) usage log file to the master
  license usage directory on the network license server fixed.

* Updating an existing session (local) usage log file fixed to be more tolerant against potential file IO errors.

* Thanks to a new Franz ACL 7.0 patch, launching external programs modified not to share any open file handles with the launched program unless explicitly required. This should fix the problem of license manager calls sometimes not returning causing the license manager to hang.

## C/API

* Compiled with VC 6.0sp6, VC 7.0, VC 7.1, and VC 8.0

* Return value formatting fixed for CAPI function dppAttributeDetermineValue

* New user function dppProjectGetProjectsPath returns the current Design++ projects directory path specified by DPPPROJECTSPATH.

* New stringArray functions dppStringArrayCopy and dppStringArraySort

* Uses 'localhost' as default hostname instead of the IP number, so that, for example, if network connection to DNS server is lost, CAPI clients don't exit.

* The problem that prevented C side from detecting that Design++ (Lisp side) had exited is now fixed. Because of the problem CAPI clients were sometimes left running even after Design++ (Lisp side) had exited.

* With dppCommFatalErrorProc, a callback function can now be registered to handle fatal errors. Default is to call system exit() function.

* Fixed dppComponentGetAssembly, dppComponentGetAssemblyAndPartList and dppComponentGetInfo to work with components without a parent, i.e., with model root components.

## COMAPI

* New examples, see:

| | |
|---|---|
| <d++>\comapi\MFC_Client | MFC client example |
| <d++>\comapi\simpleVBclient | Simple VB client |
| <d++>\comapi\Excel | Excel VBA example |

* Fixed to allocate message strings dynamically for messages sent from Design++ and, thus, removing an artificial upper bound for message length.

* The new C/API callback fatalErrorProcHandler routed as a quit message.

* Asynchronous Design++ to COM/API message size limit increased

```
  significantly. Previously messages over 500 KB could cause
  dppCOMserver to crash with a stack overflow.

* Obsolete asynchronous event handler dppAsyncNotify removed, use
  dppEventProc instead. To make the transition easier, dppAsyncNotify
  is temporarily available in a separate COM/API server version. To
  install this alternative version, just execute file
  <d++>/comapi/dppCOMserver2.exe

* Added an alias dppCommShutdownClient for old dppEndCOM function.
  This function name is same as what C/API is using.

* When call to a COM client fails, the dppCOMserver now closes the
  Design++ communication link and exists more gracefully.

--------
JAVA/API
--------

* Delivered and compiled with JAVA 1.5

* JAVA/API uses system properties DPPPORTNUMBER and DPPSERVERNAME if
  defined. This means that there can be multiple concurrent Design++
  sessions with an active JAVA/API link.  When doing
  custom load of dpp.jar, start java like:

    java -DDPPPORTNUMBER=7422 -DDPPSERVERNAME=kanishka -jar dpp.jar

* Created a simple example, which is used in documentation. See
  <d++>\java-api\com\dp\Example\

--------
GEOMETRY
--------

* Added vertex and edge calculations for primitives SYMBOL and
  SYMBOL_WITH_ATTRIBUTES to facilitate :mounted-on relations. Numbered
  vertices are sequentially defined at GEO_LOC and along each
  coordinate axis at GEO_X_SCALE, GEO_Y_SCALE and
  GEO_Z_SCALE. Finally, the center is located at the center of a box
  formed by the numbered vertices. See the updated documentation for
  more details.

-----------------------------
CAD INTEGRATION MANAGER (CIM)
-----------------------------

* Setting DPPNOCAD=t is replace with setting DPPCAD=disabled.

* CAD link initialization for Design++ restart modified to reset
  project-specific CAD settings, like menu specifications, only if the
  CAD link type (AutoCAD vs. Visio) is different from what it was when
  the image was saved.

* Design++ function DPP-CAD-SELECT-SYSTEM renamed to
  DPP-CAD-SET-CAD-SYSTEM.
```

* Introduced a new function dppCadStringTo3DPoint, which is not using strtok and so can be used to parse point value while parsing the main string with strtok. It also handles Lisp floating point print formats.

* A new callback dppCadSetFatalErrorProc introduced to handle fatal C/API errors.

* dpp-cad-send-user-message fixed to return T or NIL.

* New Design++ functions introduced to handle autocalc settings
```
    DPP-CAD-SET-AUTOCALC
    DPP-CAD-GET-AUTOCALC
```

* New Design++ function DPP-CAD-GET-CAD-NAME introduced to query the name of the CAD system currently in use.

* The problem in removing the value of the GEO_TYPE attribute fixed. This problem occurred only if the CAD Integration Manager (CIM) (or a related CAD interface: AutoCAD or Visio) was loaded.

-----------------

## MICROSTATION LINK

-----------------

* Supports MicroStation V8 2004 Edition and V8 XM Edition.

* AutoDrafter is included in the main link.

* AutoDrafter's 'Create View' operation modified to return also the extent location in addition to the actual x and y extents.

* dppextap.ml is used instead of dppextap.mo to link the external application.

* All mdl user functions have been renamed to start with dppExtap.

* The way free and user functions are specified has changed. Still, the the old code is supported for now. For example, previous specification, like
```
     ---------------
     DPP_FREE_FUNC ext_freeFuncs[] = {
       extapCreate_cellWithAttributes, // index = 0, D++ example free
     };
     ---------------
```
  is now specified as
```
     ---------------
     static freeProcFunc_t free_func_table[] = {
       {"example_Symbol", extapCreate_cellWithAttributes},
     };
     ---------------
```
  Now, on Lisp side, instead of using function's index integer to specify the function, the name string is used.

* New user functions
```
    Public void dppExtapPaletteOpen(boolean openPalett);
    Public void dppExtapMenuOpen (boolean menuOpen);
    Public void dppExtapUnitSet(int unittype);
    Public int  dppExtapUnitGet(void);
    Public void dppExtapStringToPoint(Dpoint3d *pt, char *str);
```

* In mdl/include/dppcomm.h, dppmessageP exported so that the common GEO_ data is visible to dppextap. Also, the component name has been added into this structure.

* Error handling and error messages are improved.

* Filenames are now always converted to use backslahes "\" as pathname separators instead of forward slashes "/".

* At the startup, the MS console window has been modified to show information about the different versions being used. The default dppextap code puts up a warning dialog if the msdpp version is different from that of dppextap. It is assumed that if the main or the min version numbers are different, then there might be compatibility problems. This is not the case with sub and patch numbers. The default dppextap also makes sure that the MS version that it was compiled with matches the MS version that it's being used with.

* MS Link has now some tracing functionality, which can be enabled by calling Design++ function DPP-CAD-SET-TRACING.

* Following levels are used with with AutoDrafter:
  "ADDimension" "ADText" "ADCenterline" "ADNote" "ADSymbol"

* With V8, MDL stringLinkage is used instead of the Application Element to mark Design++ components so that the Design++ component name can be seen in standard MS element information dialog box.

* The boolean type in extap functions has been replaced with BoolInt.

## AUTOCAD/ARX LINK

* Supports AutoCAD versions 2004, 2005, and 2006

* AutoCAD version 2007 is NOT supported and does not work.

* New interface functions introduced:

```
    dppDllImport const char *dppExtapGetCadInterfacePath(void);
    dppDllImport const char *dppExtapGetDppProjectsPath(void);
    dppDllImport const char *dppExtapGetProjectCadPath(void);
    dppDllImport const char *dppExtapGetProjectPath(void);
    dppDllImport const char *dppExtapGetDppPath(void);
```

* The default Design++ AutoCAD menu handling modified as follows. If the default menu (dpp-menu.* file) is requested but not found, then a sub menu (dpp-sub-menu.* file), if found, is installed to the AutoCAD menu bar just before the 'Help' menu entry.

* The menu file is now loaded before ant AutoLisp code. This allows users to modify AutoCAD menu in startup.lsp file.

```
* Fixed the coordinate entry for dpp-move command.

* Fixed some synchronization problem for opening, deleting and
  reverting models.

* AutoDrafter is included in the base product and is always loaded.

* Filename handling now supports UNC pathnames.

----------
VISIO LINK
----------

* New utility functions introduced:

  DPP-VISIO-GET-LAST-ERROR-MESSAGE ()
    Returns the last Visio link's error message shown in a dialog box.
    Example:
      (dpp-visio-get-last-error-message) ->
        "Failed to find model's 'v7' document
         Failed to find model's 'v7' page '<NULL>'
         Failed to create geometry for polyline 'POLYLINE.S252'
        "

  DPP-VISIO-GET-SETTINGS ()
    Returns certain Visio link and Visio settings.
    Example:
      (dpp-visio-get-settings) ->
       ((:tracing nil)     ;See dpp-visio-set-tracing
        (:traceFile nil)   ;See dpp-visio-set-tracing
        (:showErrorDbox t) ;Show dppVisio error dialog boxes
        ;;Following settings are Visio Application properties.
        ;;See Visio help for more details.
        ;;Application properties are returned here because
        ;;otherwise they could not be queried at all as the
        ;;function used for setting them, dpp-visio-executeline,
        ;;cannot be used for querying the property values.
        (:showChanges t)   ;See dpp-visio-show-changes
        (:TraceFlags 0)    ;
        (:UndoEnabled t)   ;Whether or not Visio maintains undo information.
       )

  DPP-VISIO-SET-SETTINGS (key-val-list)
    Sets Visio link settings.
    Example:
      (dpp-visio-set-settings
        '((:showChanges NIL)
          (:tracing T)
          (:traceFile "D:/temp/xx.log")
          (:showErrorDbox NIL))

  DPP-VISIO-SET-TRACIG (&optional (status T) (file NIL))
    Toggles Visio link tracing on or off. If the status is T and a
    file name is given, then the trace is also saved into the
    specified file. If the program <d++>\bin\win32-i86\TraceWin.exe is
    running, it gets the trace messages, otherwise the trace is
```

printed to the console window by dppvisio.vls. Note that when tracing is on, command 'Show Settings' scans drawing and prints component mappings into the trace.

DPP-VISIO-VERSION ()
Returns current Visio version information by querying the Windows registry.
Example:

```
  Visio 5        -> "5.0b"
  Visio 2000     -> "6.0"
  Visio 2000 SR1 -> "6.0 SR1" or "6.1"
  Visio 2002     -> "10.0"
  Visio 2003     -> "11.0"
```

DPP-VISIO-CLOSE-STENCILS ()
Closes all stencil windows for current model document window. Returns T if the operation succeeded or NIL if there were errors.

DPP-VISIO-CLEAN-DOCUMENT-STENCIL (&optional (all-p nil))
Removes masters from document stencil. Returns T if the operation succeeded or NIL if there were errors. Note that by default (all-p set to NIL) only unused masters are removed. Trying to remove masters that are in use can cause Visio 2000 SR1 to crash.

DPP-VISIO-CHECKSYMBOL (symbolName)
Checks if the specified symbol exists in document master or in stencil.
Example:

```
   (dpp-visio-checkSymbol "Horizontal") -> T
   (dpp-visio-checkSymbol "Basic Shapes;Double flexi-arrow") -> T
```

DPP-CAD-USING-VISIO-P (&optional cad-system)
Predicate for checking if the current CAD system in use is Visio.
Example:

```
   (dpp-cad-using-visio-P) -> T
```

* Design++ function DPP-VISIO-CLEAN-DOCUMENT-STENCIL has been optimized.

* Error dialogs modified to show Visio error messages, when available, instead of the error codes.

* An optional argument all-p, defaults to nil, is added to Design++ function DPP-VISIO-CLEAN-DOCUMENT-STENCIL. This prevents the problem where deleting the masters that are in use in the current document sometimes causes Visio 2000 SR1 to crash. Now, by default, only unused masters are removed.

* When Visio link tracing is on, command 'Show Settings' scans drawing and prints component mappings into the trace.

* New free primitive BoxWithFill introduced. The new primitive has an optional fill-specification argument (string) for specifying values to shape's 'Fill Format' section. Cell values are:

```
        FillPattern  -     int
```

```
          FillForegnd  -     string/int
          FillBkgnd    -     string/int

  Fill specifications are passed in as GEO_ARGUMENTS value, like
  "2 Red Cyan"

* Symbol scaling modified to do the scaling only if scale != 1.0

---------------------------
USER INTERFACE SERVER (UIS)
---------------------------

* New 'License Usage Log' subdialog introduced for the 'License
  Manager' dialog.

* The maximum dialog sizes are now limited to the size of the current
  display.

------------------------
DESIGN RULE EDITOR (DRE)
------------------------

* Design Rule Editor (DRE) integrated more closely with the rest of
  the Developer's Interface (UIP):
  - DRE now opens automatically in the vicinity of the location from
    where the edit request was issued, e.g., Component Editor or
    Magnifier.
  - Selecting Component Editor's 'Design Rule' pane opens DRE
    automatically assuming DRE is the default rule editing
    tool. Otherwise, rule file is opened in Emacs. Thus, a rule can no
    longer be edited inside the 'Design Rule' pane.
  - To speed up the opening of DRE, it is started automatically
    whenever UIP is started.
  - DRE's 'Context Sensitive Menu' dialog is integrated with the main
    DRE window.
  - Selecting DRE's 'X' (close) button minimizes DRE into the taskbar
    instead of exiting the program.

* The binding of <return> key modified to be <linefeed>+<tab>.

* Design rule compilation modified to show compilation warning and
  error messages also in a dialog in addition to printing them to the
  Command Interpreter. Requiring user acknowledgment assures that
  compilation messages don't go unnoticed.

  Note that the compilation messages are shown in a dialog only when
  the compilation is initiated from the Design Rule Editor (DRE) or
  some other Design++ client/server. If a rule is compiled in Emacs or
  Command Interpreter, then the messages are only printed to the
  Command Interpreter as before.

* The placing of subdialogs fixed to place the dialogs over the parent
  dialog.

* Removed some obsolete reports from 'Rule Check' result dialog.

* DRE title bar now shows 'Design Rule for Attribute <name> in Class
```

```
  <name>'

* Design++ icon is now shown for DRE when using ALT-TAB instead of the
  generic Windows icon.

* The 'Help' menu now opens the DRE section in the Design++ User's
  Manual.

* The size of the rule window has been adjusted to about 72
  characters.

---------------
EMACS INTERFACE
---------------

* The Emacs version that Design++ is delivered with is GNU Emacs 21.4.

* fi:common-lisp-host is now set to "localhost" instead of (system-name)
    (setq fi:common-lisp-host "localhost")
  as this prevents Design++ from exiting if DNS is used and the network
  connection is lost.

* Autosaving is now enabled by default. To disable it, simply
  uncomment the following line in <d++>\misc\dpp.el
    ;;(setq auto-save-default nil)

* When compiling a design rule or a function, the compilation results
  are shown in the console buffer (Design++ Command Interpreter). If
  the buffer is not visible, a new Emacs window is opened for it. The
  default size for the new console window has been reduced to 5
  lines. It used to be half of the current window.

* Setting DPPEMACSWINMODE=yes causes Emacs to be started in Design++
  Emacs Windows mode (winmode).

----------------------------
USER INTERFACE PROGRAM (UIP)
----------------------------

* New Querytool dialog introduced for searching information from
  libraries and models.

* New menus added for AutoDrafter and ReportWriter

* The main dialog has a new special icon for AutoDrafter's drawing
  library.

* Design Rule Editor (DRE) is now opened when clicking a component
  editor's 'Design Rule' pane.

* Design Rule Editor (DRE) is now always opened to the current
  cursor position. The position is no longer fixed.

* Component/classname symbols in frame facet value "#<FRAME x.s123 v>"
  fields are now highlighted. Double-clicking the highlighted symbol
  opens up a component editor for that component or class.
```

* After selecting an attribute, the copy command (Ctrl-C) copies the attribute name into Windows clipboard.

* In Magnifier, the copy command (Ctrl-C) copies the selected items into Windows clipboard.

* 'Mark for Copying' shortcut changed from Crtl-M to Ctrl-C.

* 'Copy Marked' shortcut changed from Shift-Insert to Ctrl-V.

* New 'Properties' dialog introduced for projects, libraries, models, components, and classes.

* New 'Design++ Settings' dialog introduced for viewing current internal settings.

* 'Available Projects' dialog has a new 'Browse...' option which allows projects to be loaded from any directory, not just from the projects directory.

* 'Window>More Windows...' dialog has new 'Close', 'Minimize', and 'Activate' buttons. The 'Close' button now closes the selected dialog and not the 'More Windows...' dialog itself, which can be closed with the X box.

* Many of the dialogs and dialog items have a 'Help' command or menu, which now opens either the help for the selected item or the 'Start Here' document in Acrobat.

## DATABASE LINK

* RDB (ODBC) link optimized significantly for retrieving large amount of data with a single query. The larger the amount retrieved, the bigger the improvement.

* Retrieving 50 KB and larger amounts of data fixed to properly return all of the data and not to cut off anything.

* Design++ function DPP-RDB-SQL function fixed not to filter out #\; (semicolon) and #\return (return) characters from its sql-clause argument. As a result, SQL clauses like
  ```
  (dpp-rdb-sql "insert into test_table values('1;2')" :std)
  ```
  are now passed through as such.

* Both ODBC link and the classic RDB (ORACLE) link are built in. To switch between the two links, use Design++ function DPP-RDB-LINK-TYPE. For example, executing
  ```
  (dpp-rdb-link-type :oracle)
  ```
  resets the current database link and switches to RDB (ORACLE) link.

* Support for national character sets improved for ODBC link.

* Usernames and passwords are now handled as strings instead of symbols. Still, Design++ functions accept symbols. For example, the following username/password combinations are equivalent

```
     :std 'passwd
     "STD" "PASSWD"

----------------
RELATION BROWSER
----------------

* The browser dialog modified to look more like the other UIP grapher
  dialogs.

* Modified to use Java's WindowsClassicLookAndFeel instead of
  WindowsLookAndFeel so that with Windows XP themes, the button
  background colors work as intended.

----
MISC
----

* dppdiag shows following new information with -sys option
  - pending reboot?
  - crypkey service status
  - used inside of a virtual machine?
  - admin privileges?

* dppdiag new argument -log to enables logging into
  %TEMP%\\dppdiag-'pid'.log

=====
```