March 1, 2002

Design++ 5.0 Release Notes for Windows NT4/2000/XP

We are pleased to announce the release of Design++ 5.0 for Windows NT4/2000/XP. This is the third major Design++ release for the PC platform.

HIGHLIGHTS

Design++ 5.0 includes several new modules and enhancements, the most important ones being the following:

VISIO LINK

A new standard CAD link with Microsoft Visio is introduced. Since Visio is a 2D-only system, the Visio Link maps the 3D Design++ primitives into their 2D representation (when feasible) based on the selected view in Visio. The Visio Link supports Visio versions 2000, 2000 SR1, and 2002. Note that the Visio Link is sold as a separate Design++ module.

JAVA/API

A preliminary version of Java/API is introduced. The Java/API comes with a set of Java classes and methods, which allow Java to communicate with Design++.

RELATION BROWSER

Relation Browser, a new visual tool for exploring relations between model components, is introduced. Unlike the Model Editor, which displays only the product structure (partof/subparts) relation, Relation Browser lets the user browse all model relations.

Relation Browser is implemented in Java using the new Java/API. To assure the availability of Java this Design++ release includes Java Runtime Environment (JRE) 1.3.1

GEOMETRIC MODELING MADE EASIER

A new relation, mounted-on (and its opposite relation, mounts) is introduced. It is an outcome of our efforts to make the geometric modeling significantly easier by automating the details of inter-component positioning and orientation through the use of predefined relations like mounted-on.

The mounted-on relation comes with a symbolic topology language that allows components to be positioned relative to one another as they are related.

NETWORK LICENSING ENHANCED - USAGE MONITORING AND LICENSE SHARING

The License Manager dialog is enhanced to display the network license usage information. This allows easy monitoring of the current floating license users.

Network licensing is further enhanced to allow several server installations to share a single license pool, even when running different Design++ versions. The licensing information is hosted in a separate, user-specifiable, directory, which does not have to reside within the Design++ directory. For better performance the client workstations can opt to host Design++ directory on their local disk drives and only mount the licensing information.

COM/API ENHANCED

The COM/API is enhanced to allow clients to register with unique names, thus allowing Design++ to communicate with multiple Visual Basic applications simultaneously. This is identical to what the C/API provides.

NEW UIP VERSION - NO MORE RESOURCE FILES

The Developer's User Interface (UIP) is modified to save the user preferences to Windows registry instead of resource files. This should eliminate the infamous resource file corruption problem.

NEW USER FUNCTIONS

Several new user functions are introduced, the most notable being dpp-eval-at-interval and dpp-attribute-evaluation-order.

dpp-eval-at-interval allows scheduling of expressions to be evaluated repeatedly at a later time. For example, a global garbage collection could be scheduled to run every night while the system is idle.

dpp-attribute-evaluation-order allows customizing of the attribute value determination order.

ONLINE DOCUMENTATION ACCESS

Online documentation (PDF) for all functions and design rule macros is now directly accessible from Emacs (shortcut <ctrl>-c <ctrl>-f) and Design Rule Editor. Give it a try!

Adobe Acrobat Reader is included on the delivery CD's contrib directory.

AUTOCAD LINK

ObjectARX-based AutoCAD Link supports AutoCAD releases 2000, 2000i and 2002. The old ADS-based AutoCAD Link and AutoCAD releases 13 and 14 are no longer supported.

MICROSTATION LINK

MicroStation Link has a preliminary support for MicroStation V8. On Windows 2000 and XP, the MicroStation Link is only supported with MicroStation/J, not with MicroStation 95 or SE.

WINDOWS XP SUPPORT

In addition to Windows NT4 and 2000, Design++ 5.0 is also supported on both Windows XP Professional and Home versions.

DESIGN++ FUNCTIONS (LISP/API)

---

* Design++ Function DPP-WRITE-EXTERNAL fixed to properly format values of type (:list-of :string). For example, it used to format value ("ABC" "DEF" "GHI") incorrectly as (ABC DEF GHI).
  DPP-WRITE-EXTERNAL modified also to make sure that the generated component references identify unambiguous components. The referencing used to start from the direct assembly, which can be ambiguous if the assembly has siblings of the same class.

```
;|| 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 (symbol)
;||   output-file (optional):
;||    Output file name, default = nil (symbol or string)
;||  RETURNS:
;||   NIL
;||  EXAMPLE:
;||   (dpp-write-external)
;||
```

* Design++ Function DPP-WRITE-EXTERNAL modified to start the referencing all the way from the model root. This assures that the generated indirect component reference clause, when accessed in an external data file, will generate dependencies to the whole substructure branch starting from the model root. Thus, any change in the substructure hierarchy will trigger value redetermination.

* A new Design++ Function DPP-WRITE-EXTERNAL-FOR-COMPONENTS
  introduced. Unlike DPP-WRITE-EXTERNAL, which writes the external
  data file for all the components in a model,
  DPP-WRITE-EXTERNAL-FOR-COMPONENTS allows user to specify the
  components for which the external data file is written for.

```
;|| 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++ Function POINT-INSIDE-POLYGON-P reimplemented. The old
  implementation failed to detect certain special cases. For example,

```
       (dpp-point-inside-polygon-p
          (list 7000.0 2000.0 0.0)
          (list '(0.0 0.0 0.0) '(14000.0 0.0 0.0)
                '(14000.0 4000.0 0.0) '(0.0 4000.0 0.0)))
       ==> NIL
```

* A new Design++ Function DPP-SEGMENT-SEGMENT-INTERSECTION introduced.

```
;|| DPP-SEGMENT-SEGMENT-INTERSECTION (p0 p1 p2 p3)
;||  PURPOSE:
;||   Returns the intersection of the lines represented by the pairs of
;||   points if that intersection is within both line segments, i.e.,
;||   true segment intersection.
;||  ARGUMENTS:
;||   p0, p1, p2, p3:
;||    Point (list of x, y and z)
;||  RETURNS:
;||   Intersection point (list of x, y and z) if segments intersect, NIL
;||   otherwise.
;||  EXAMPLE:
;||   (dpp-segment-segment-intersection '(0 0 0) '(1 1 0) '(0 1 0) '(1 0 0))
;||   ==> (0.5 0.5 0.0)
```

```
;||
```

* A new optional argument, ADD-P, added for Design++ Function
  DPP-ATTACH-EXTERNALS. The new argument allows incremental addition
  of new external data files. Values defined in the new external data
  files override existing value definitions and trigger proper
  consistency maintaining value propagation.
  A new Design++ Function DPP-DETACH-EXTERNALS introduced for
  detaching select (or all) external data files attached to a model.

```
;|| DPP-ATTACH-EXTERNALS (&optional (model-name (dpp-get-current-model))
;||                                  extarnal-files add-p)
;|| PURPOSE:
;||  Attaches external files to model
;|| ARGUMENTS:
;||  Optional:
;||   model-name (symbol):
;||    Model-name, default = current model
;||   extarnal-files (list of strings):
;||    External file names in current project's externals directory,
;||    default = NIL
;||   add-p (T/NIL):
;||    Whether to add the new external files with the existing ones,
;||    default = NIL
;|| RETURNS:
;||  T if all external files processed successfully, otherwise NIL
;||  indicating that some of the external files were not found in the
;||  project's externals directory.
;|| EXAMPLE:
;||  (dpp-attach-externals 'a-bike '("ext.lisp" "pipe.lisp"))
;||  ==> T
;||  (dpp-attach-externals 'a-bike)
;||  ==> T
;||  (dpp-attach-externals 'a-bike '("/tmp/temp.lisp" "pipe.lisp"))
;||  ==> NIL ;File /tmp/temp.lisp not in externals directory.
;||
;|| Modified to properly return T or NIL.
;|| Modified NOT to filter out non-Lisp (*.lisp) files.
;|| Added a new optional argument, ADD-P, that allows incremental
;|| addition of new external data files. Values defined in the new
;|| external data files override existing value definitions and
;|| trigger proper consistency maintaining value propagation.
;||

;|| DPP-DETACH-EXTERNALS (&optional (external-files :all)
;||                                  (model-name (dpp-get-current-model)))
;|| PURPOSE:
;||  Detaches external files from model
;|| ARGUMENTS:
;||  Optional:
;||   extarnal-files (list of strings):
;||    External file names in current project's externals directory,
;||    default = :ALL
;||   model-name (symbol):
;||    Model-name, default = current model
```

```
;|| RETURNS:
;||  List of model's remaining external files after the requested
;||  external files have been detached. NIL indicates that all
;||  external files were detached.
;|| EXAMPLE:
;||  (dpp-attach-externals 'tka '("ext1.lisp" "ext2.lisp"))
;||  ==> T
;||  (dpp-detach-externals '("ext1.lisp"))
;||  ==> ("ext2.lisp")
;||  (dpp-detach-externals :all)
;||  ==> NIL
;||
```

* A new Design++ Function DPP-WRITE-REPORT-FOR-COMPONENTS
  introduced. Unlike DPP-WRITE-REPORT, which writes the report for all
  the components in a model branch, DPP-WRITE-REPORT-FOR-COMPONENTS
  allows user to specify the components for which the report file is
  written for.

```
;|| DPP-WRITE-REPORT-FOR-COMPONENTS (component-list &optional output-file
;||                                   (model-name (dpp-get-current-model)))
;|| PURPOSE:
;||  Generates a report for the components in COMPONENT-LIST.
;||  For each component the values of the attributes listed in
;||  report_attributes attribute (list of symbols) are written
;||  to OUTPUT-FILE.
;|| ARGUMENTS:
;||   component-list (list of frames or component names):
;||    List of components for which the report is to be written.
;||   output-file (optional):
;||    Output filename, default = prompt user (symbol or string)
;||   model-name (optional):
;||    Model name, default = current model (symbol)
;|| RETURNS:
;||  Filename if successful, otherwise NIL
;|| EXAMPLE:
;||  (dpp-write-report-for-components
;||     '(floor.S27464 ceiling.s27465) "/tmp/report.txt")
;||  ==> "/tmp/report.txt"
;||
```

* Design++ Function DPP-DELETE-CLASS fixed to handle multi-class
  ambiguity, i.e., identically named classes in different libraries.

```
;|| DPP-DELETE-CLASS (class-name class-kb-name
;||                   &optional delete-subclasses-p)
;|| PURPOSE:
;||  Deletes existing class from library (optional class+subclasses)
;|| ARGUMENTS:
;||  class-name:
;||   Name of class to be deleted (symbol)
;||  class-kb-name:
;||   Kb-name of class to be deleted (symbol)
;||  Optional:
;||   delete-subclasses-p:
```

```
;||    T for yes or NIL (default) for no, deletes all subclasses under
;||    class to be deleted
;|| RETURNS:
;||  T if class deleted else NIL
;|| EXAMPLE:
;||  (dpp-delete-class 'testclass 'reports)
;||  ==> T
;||
```

* Design++ Function DPP-REMOVE-VALUE fixed to make sure that the attribute, whose value is being removed, is not scheduled for determination (by a design rule) as that would override the value removal.
  Similarly, attribute value determination by a design rule optimized to make sure that the attribute is not scheduled unnecessarily for later determination, which would be redundant.

```
;|| DPP-REMOVE-VALUE (component attribute)
;||  PURPOSE:
;||   Removes the local value of component's attribute
;||  ARGUMENTS:
;||   component:
;||    Component (frame-or-ref)
;||   attribute:
;||    Attribute (symbol)
;||  RETURNS:
;||   NIL
;||  EXAMPLE:
;||   (dpp-remove-value 'with_centerline.s99 'geo_length)
;||   ==> NIL
;||
```

* Design++ Function DPP-PROJECT-CAD-PATHNAME fixed to return a proper pathname for AutoCAD 2000 (arx2000).

```
;|| DPP-PROJECT-CAD-PATHNAME
;|| PURPOSE:
;||  Returns project cad pathname
;|| ARGUMENTS:
;|| RETURNS:
;||  pathname string or NIL
;|| EXAMPLE:
;||  (dpp-project-cad-pathname)
;||  ==> "/home/code/d++/projects/bike/acad/"
;||
```

* Design++ Function DPP-COPY-FILE modified to preserve the time by setting the modified-time on TO-PATHNAME be the same as FROM-PATHNAME.

```
;|| DPP-COPY-FILE (from-pathname to-pathname)
;|| PURPOSE:
;||  Copies the file named FROM-PATHNAME to file named
;||  TO-PATHNAME. Does not allow an existing file TO-PATHNAME to be
;||  overwritten.
```

```
;|| ARGUMENTS:
;||  from-pathname (string or pathname):
;||    File to be copied.
;||  to-pathname (string or pathname):
;||    Target file.
;|| RETURNS:
;||  Returns the target file pathname TO-PATHNAME if the operation is
;||  successful, otherwise NIL.
;|| EXAMPLE:
;||  (dpp-copy-file "/home/code/d++/projects/bike/functions/func1.wfasl"
;||                 "/home/code/d++/projects/bike/functions/newfunc1.wfasl")
;||  ==> #p"/home/code/d++/projects/bike/functions/newfunc1.wfasl"
;||
;|| Modified to preserve the time by setting the modified-time on
;|| TO-PATHNAME be the same as FROM-PATHNAME.
;||
```

* Design++ Function DPP-POINT-INSIDE-POLYGON-P modified to handle any polygon parallel to XY-plane and not just the ones on the XY-plane (Z-coordinate is 0.0).
  Note that DPP-POINT-INSIDE-POLYGON-P is strictly planar and will test for polygon inclusion ONLY for polygons and points that are coplanar and parallel to the XY-plane.

```
;|| (DPP-POINT-INSIDE-POLYGON-P point polyline
;||                           &optional (boundary-included-p t))
;||
;||  EFFECT: Determines whether the point is inside the polygon.
;||
;||  RETURNS: T (true) is inside, otherwise NIL (false)
;||
;||  EXAMPLE: (dpp-point-inside-polygon-p
;||             '(0 0 0) '((1 1 0) ( -1 1 0) (-1 -1 0) (1 -1 0.0)))
;||           ==> T
;||
```

* A new Design++ Function DPP-EVAL-AT-INTERVAL introduced for scheduling expressions to be evaluated repeatedly at a later time. For example, a global garbage collection could be scheduled to run every night while the system is idle.

```
;||DPP-EVAL-AT-INTERVAL (expression-or-function
;||                                &key (start-time :now)
;||                                (number-of-evals 1))
;||                                (interval '(1 :minute)))
;|| PURPOSE:
;||  Schedules EXPRESSION-OR-FUNCTION to be evaluated repeatedly at a
;||  later time. Starting at START-TIME, the EXPRESSION-OR-FUNCTION is
;||  evaluated repeatedly for a NUMBER-OF-EVALS times with INTERVAL
;||  time in between each evaluation.
;||
;||  Scheduling of the evaluation is assigned to a separate process
;||  named uniquely using the argument values. See RETURNS:
;||  below. Each call to dpp-eval-at-interval activates a new
;||  scheduler process. Processes can be killed either via
```

```
;||  'File>Processes...' dialog or with the top-level :kp command in
;||  the Command Interpreter.
;||
;|| ARGUMENTS:
;||  expression-or-function (expression, function name, or function):
;||    Expression or function to be scheduled for evaluation at a
;||    later time
;||  start-time (<hour>.<minute>, default :now):
;||    Time when repeated evaluation of expression-or-function
;||    begins. Start-time is based on a 24-hour clock, thus, the
;||    evaluation can be scheduled 24 hours ahead of time. Default
;||    value :now triggers the evaluation immediately.
;||  number-of-evals (integer, default 1):
;||    Number of times expression-or-function is to be evaluated.
;||  interval (list of integer and <time unit>, default '(1 :minute)):
;||    The time between each evaluation. Supported <time units> are
;||    :second, :minute, :hour, and :day. Note that the time between
;||    evaluations is measured from the start of an evaluation.
;||
;|| RETURNS:
;||  The name of the internal process scheduling the evaluation. The
;||  name can be used to identify the scheduling process in case it
;||  needs to be killed.
;||
;|| EXAMPLES:
;||  The first example schedules a global garbage collection (GC) to be
;||  run every night at 3:00 AM for the next 30 days. Global GC's are
;||  time consuming and, thus, it's nice to have the memory cleaned
;||  while the system is idle.
;||
;||  You may want to consider adding this example to one of your
;||  project's function files or even to your personal Design++
;||  initialization file, %HOME%\d++-init.lisp.
;||
;||  D++(8): (dpp-eval-at-interval
;||            '(excl:gc t)
;||            :start-time 03.00
;||            :interval '(24 :hour)
;||            :number-of-evals 30)
;||  "GC: at 03:00 for 30 evals at 24 hour interval"
;||  D++(9):
;||
;||  The second example is a general test case, where the expression
;||  prints out the current time every time it is evaluated.
;||
;||  D++(10): (dpp-eval-at-interval
;||             '(multiple-value-bind (second minute hour date month year
;||                                    day-of-week daylight-saving-p
time-zone)
;||                (decode-universal-time (get-universal-time))
;||                (declare (ignore day-of-week daylight-saving-p time-zone))
;||                (format t "~&DPP-EVAL-AT-INTERVAL: ~2,'0D/~2,'0D/~D
~2,'0D:~2,'0D:~2,'0D~%"
;||                 month date year hour minute second))
;||             :start-time 17.42
```

```
;||            :interval '(3 :second)
;||            :number-of-evals 5)
;||  "MULTIPLE-VALUE-BIND: at 17:42 for 5 evals at 3 second interval"
;||  D++(11):
;||  DPP-EVAL-AT-INTERVAL: 07/13/2000 17:42:00
;||  DPP-EVAL-AT-INTERVAL: 07/13/2000 17:42:03
;||  DPP-EVAL-AT-INTERVAL: 07/13/2000 17:42:06
;||  DPP-EVAL-AT-INTERVAL: 07/13/2000 17:42:09
;||  DPP-EVAL-AT-INTERVAL: 07/13/2000 17:42:12
;||
;||  D++(11):
;||
```

* New CAD Integration related Design++ Functions introduced:
  - DPP-CAD-READY-P, checks whether CAD program has started, registered, and is ready to accept requests.
  - DPP-CAD-REGISTER-CAD-HAS-REGISTERED-FUNCTION, registers a function to be called after the CAD program has started and registered with Design++.
  - DPP-CAD-REGISTER-CAD-HAS-UNREGISTERED-FUNCTION, registers a function to be called after the CAD program has unregistered with Design++ and is about to exit.

```
;|| DPP-CAD-READY-P ()
;|| PURPOSE:
;||  Predicate for checking whether CAD program has started,
;||  registered, and is ready to accept requests.
;|| RETURNS:
;||  T or NIL
;|| EXAMPLE:
;||

;|| DPP-CAD-REGISTER-CAD-HAS-REGISTERED-FUNCTION (fn-or-fn-name)
;|| PURPOSE:
;||  Registers a function to be called after the CAD program has
;||  started and registered with Design++
;|| ARGUMENTS:
;||  fn-or-fn-name (function object or function name):
;||   Function to be called after CAD has registered
;|| RETURNS:
;||  Function or function name
;|| EXAMPLE:
;||  (dpp-cad-register-cad-has-registered-function
;||   #'(lambda ()(format t "~&CAD program has just registered.~%")))
;||  ==> #<Interpreted Function (unnamed) @ #x235efcb2>
;||

;|| DPP-CAD-REGISTER-CAD-HAS-UNREGISTERED-FUNCTION (fn-or-fn-name)
;|| PURPOSE:
;||  Registers a function to be called after the CAD program has
;||  unregistered with Design++ and is about to exit.
;|| ARGUMENTS:
;||  fn-or-fn-name (function object or function name):
;||   Function to be called after CAD has unregistered
;|| RETURNS:
```

```
;||  Function or function name
;|| EXAMPLE:
;||  (dpp-cad-register-cad-has-unregistered-function
;||   #'(lambda ()(format t "~&CAD program has just unregistered.~%")))
;||  ==> #<Interpreted Function (unnamed) @ #x235f01ca>
;||
```

* A new optional argument, STORE-VALUE-TO-ATTRIBUTE-P, added for Design++ Function DPP-FROM-EXTERNAL. The new argument controls whether or not the retrieved value is stored to the attribute. The new argument defaults to T. Thus, by default, the value is stored to the attribute.

  Note that the dependencies specified in the indirect component reference of the selected (component attribute value) -triplet are still created even if STORE-VALUE-TO-ATTRIBUTE-P is NIL. The dependencies are created in anticipation that the external value will be stored to the attribute at some later time. See Example2 below.

```
;|| DPP-FROM-EXTERNAL
;||      (component attribute
;||       &optional (create-dependencies-when-indirect-reference-p t)
;||                 (store-value-to-attribute-p t))
;||  PURPOSE:
;||   Retrieves the attribute value of a component from external
;||   data sources. The retrieved value is stored to the attribute,
;||   and then returned. The new value is also locked.
;||   If the value is not defined in external data sources, NIL is
;||   returned.
;||   If the model of the component does not have any external data
;||   sources attached to it, user is asked to select the data sources
;||   from a menu.
;||  ARGUMENTS:
;||   component:
;||    Component (frame-or-ref), whose attribute value is to
;||    be retrieved
;||   attribute:
;||    Attribute (symbol), whose value is to be retrieved
;||   Optional:
;||    create-dependencies-when-indirect-reference-p
;||    (T (default) or NIL):
;||     Controls whether dependencies specified in the indirect
;||     component reference of the selected (component attribute
;||     value) -triplet are to be created or not.
;||    store-value-to-attribute-p (T (default) or NIL):
;||     Controls whether or not the retrieved value is stored to the
;||     attribute. By default, the value is stored to the attribute.
;||  RETURNS:
;||   Attribute value of component retrieved from external
;||   data sources or NIL if not defined
;||  EXAMPLE1:
;||   A Design Rule that changes the determination order of
;||   attribute value.
;||
```

```
;||   (:! blade geo_length
;||      (or ;Value defined in external data sources?
;||          (dpp-from-external self 'geo_length)
;||          ;;If not, let's use standard rule body.
;||          5000))
;||
;||   Determination order is thus now:
;||    1. Local value
;||    2. External value
;||    3. Value by Rule
;||    4. Inherited value
;||    5. Value from user
;||
;||  EXAMPLE2:
;||   A Design Rule that modifies the value retrieved from external
;||   data sorces. Note that STORE-VALUE-TO-ATTRIBUTE-P is NIL so that
;||   the value is not stored to the attribute prematurely. Note also
;||   that the dependencies specified in the indirect component
;||   reference of the selected (component attribute value) -triplet
;||   are still created even if STORE-VALUE-TO-ATTRIBUTE-P is NIL. The
;||   dependencies are created in anticipation that the external value
;||   will be stored to the attribute at some later time.
;||
;||   (:! component attribute
;||      (adjust-external-value
;||         (dpp-from-external component attribute t nil)))
;||
;||  Optional predicate argument
;||  create-dependencies-when-indirect-reference-p added.
;||
;||  Optional predicate argument store-value-to-attribute-p added
;||
```

* A new Design++ Function DPP-RENAME-CURRENT-MODEL introduced for renaming the currently active model without saving it to a file. DPP-RENAME-CURRENT-MODEL is a fast alternative for DPP-SAVE-AS-MODEL which always saves the copied model to a file. With DPP-RENAME-CURRENT-MODEL the model saving responsibility is left to the user.

```
;|| DPP-RENAME-CURRENT-MODEL (new-current-model-name)
;|| PURPOSE:
;||  Renames the current model without saving it to a file. An
;||  associated CAD model is also renamed. Note that if the renamed
;||  model is not saved later by the user, the model cannot be opened
;||  once the current Design++ session has exited.
;|| ARGUMENTS:
;||  new-current-model-name:
;||   New current model name (symbol)
;|| RETURNS:
;||  New current model name if the renaming succeeded otherwise nil
;|| EXAMPLE:
;||  (dpp-rename-current-model 'tka2)
;||  ==> tka2
;||
```

* A new Design++ Function DPP-ATTRIBUTE-EVALUATION-ORDER introduced for setting the attribute value determination order. Note that since DPP-ATTRIBUTE-EVALUATION-ORDER is really implemented as a macro, its argument should not be quoted.

```
;|| DPP-ATTRIBUTE-EVALUATION-ORDER (&optional evaluation-order)
;||  PURPOSE:
;||  Sets the attribute value determination order. The default
;||  evaluation order is
;||     1. LOCAL     (use local value)
;||     2. RULE      (use design rule to determine the value)
;||     3. INHERITED (use inherited value)
;||     4. EXTERNAL  (retrieve the value form external data sources)
;||     5. USER      (prompt the value from user).
;||  ARGUMENTS:
;||   evaluation-order (optional):
;||    New attribute value evaluation order (list of (:local :external
;||    :rule :inherited :user) or NIL (default)). Note that since
;||    dpp-attribute-evaluation-order is implemented as a macro, the
;||    list of evaluation order should NOT be quoted.
;||  RETURNS:
;||   The new attribute value evaluation order or the current
;||   evaluation order if evaluation-order is NIL (default).
;||  EXAMPLE1:
;||   (dpp-attribute-evaluation-order)
;||   ==> (:LOCAL :EXTERNAL :RULE :INHERITED :USER)
;||  EXAMPLE2:
;||   To give a higher precedence for the values defined in external
;||   data sources, add the following expression to one of your
;||   project's function files.
;||   (eval-when (load eval)
;||     (dpp-attribute-evaluation-order
;||       (:LOCAL :EXTERNAL :RULE :INHERITED :USER)))
;||
```

* Design++ Function POINT-INSIDE-POLYGON-P reimplemented. The old implementation failed to detect certain special cases. For example,

```
      (dpp-point-inside-polygon-p
        '(3700 4000 0)
        '((3000 0 0)(5700 0 0)(5700 4000 0)(3000 4000 0)(3000 0 0))
        nil)
      ==> T
```

  Also added min/max box checking for X, Y and Z which eliminates points outside more efficiently.

* Relation mechanism enhanced to allow relations to have WHEN-RELATED and WHEN-UNRELATED methods. The WHEN-RELATED method is called right after components are related (with both the relation itself and its opposite relation), and the WHEN-UNRELATED method is called right before components are unrelated.

  The new methods are defined with new DPP-DEF-RELATION keyword

arguments WHEN-RELATED-METHOD and WHEN-UNRELATED-METHOD:

```
DPP-DEF-RELATION (relation-name
                  &key (opposite-relation-name relation-name)
                       relation-rule-access-command
                       all-relation-rule-access-command
                       monitored-attributes
                       WHEN-RELATED-METHOD
                       WHEN-UNRELATED-METHOD
                       when-modified-method)

For example,
(dpp-def-relation mounted-on
                  :opposite-relation-name mounts
                  :relation-rule-access-command :mounted-on
                  :all-relation-rule-access-command :all-mounted-on
                  :when-related-method
                   #'mounted-on-when-related-method
                  :when-unrelated-method
                   #'mounted-on-when-unrelated-method)
```

The WHEN-RELATED and WHEN-UNRELATED methods are functions (funcallable objects) which are called with a fixed set of arguments:

```
WHEN-RELATED-METHOD (component1 component2 relation-name
                     component1-relation-init-data
                     component2-relation-init-data)
WHEN-UNRELATED-METHOD (component1 component2 relation-name)
```

To support the new relation methods both Design++ Function DPP-RELATE and the special RELATIONS attribute, which specifies a component's required relations, are modified to allow relation-specific initialization data to be passed to a relation's WHEN-RELATED method when the relation is established. For example,

```
(DPP-RELATE (list (:? leg) :face 'top)
            (list (:? round-top) :face 'bottom)
            :mounted-on)

(:! table RELATIONS
    (list :mounted-on
          (list (:? leg) :face 'top)
          (list (:? round-top) :face 'bottom)))
```

[NOTE that a single relation-spec is no longer required to be in a list. Thus, the above example now works OK.]

In both cases the specification for the component argument has been extended to allow it to be either a single component or a list of a component followed by the component's relation initialization data:

```
<component> = component |
              '(' component <relation-initialization-data> ')'
```

```
;|| DPP-RELATE (component1 component2 &optional
;||                      (relation :connected)
;||                      (model-name (dpp-get-current-model)))
;||  PURPOSE:
;||   Relates component1 and component2 bidirectionally with
;||   given relation and its inverse relation. Relates component1
;||   to component2 with given relation and component2 to component1
;||   with its inverse relation.
;||   Design++ provides a set predefined relations :CONNECTED,
;||   :IN-OUT, and :INCLUDES.
;||  ARGUMENTS:
;||   component1 (frame-or-ref or (list frame-or-ref <relation-init-data>)):
;||    [component1] --(relation)--> [component2]
;||   component2 (frame-or-ref or (list frame-or-ref <relation-init-data>)):
;||    [component2] --(inverse-relation)--> [component1]
;||   relation (optional):
;||    Connecting relation, default = :connected (symbol)
;||   model-name (optional):
;||    Model name of the two components, default = current (symbol)
;||  RETURNS:
;||   Component2 (frame) if OK, otherwise NIL
;||  EXAMPLE:
;||   (dpp-relate 'pipe.s123 'pipe.s321 :in-out)
;||   ==> #<FRAME PIPE.S321 TEST>
;||
;||  Modified to allow relation initialization data to be passed to a
;||  relation's WHEN-RELATED method when the relation is
;||  established. This is done by extending the specification for the
;||  component argument to be either a single component or a list of a
;||  component followed by the component's relation initialization
;||  data, for example
;||  (dpp-relate
;||    (list (:? leg) :face 'top :vertex 5)
;||    (list (:? top) :face 'bottom :vertex 1)
;||    :mounted-on)
;||
```

* Two internal collinearity predicates revisited. One is used for checking the collinearity of three points and the other for the collinearity of two vectors. Both predicates were improved to better interpret the angle formed by the lines joining the three points or the two vectors. These predicates are used in the following Design++ Functions

```
DPP-COLINEAR-P (point1 point2 point3)
DPP-POINT-LINE-ORTHO-DISTANCE (point lpt1 lpt2)
DPP-DROP-POINT-ON-LINE (point lpt1 lpt2)
DPP-DROPPED-POINT-WITHIN-SEGMENT-P (point lpt1 lpt2)
DPP-COLLINEAR-VECTORS-P (v1 v2)
```

* A new Design++ Function DPP-PLANE-FROM-3-POINTS introduced for returning a plane defined by the 3 point arguments.

```
;|| DPP-PLANE-FROM-3-POINTS (p1 p2 p3)
;||  PURPOSE:
```

```
;||   Returns a plane containing the original three points as
;||   a list of components of a normalized vector perpendicular
;||   to it and it's distance to origin.
;||  ARGUMENTS:
;||   p1, p2, p3:
;||    Points on the plane (each a list of x, y and z)
;||  RETURNS:
;||   Plane (list of x, y, z and distance to origin), or NIL if
;||   points are coincident
;||  EXAMPLE:
;||   (dpp-plane-from-3-points '(0 0 0) '(1 1 0) '(1 0 0))
;||   ==> (0.0 0.0 -1.0 0.0)
;||
```

* A new keyword argument CURRENT-DIRECTORY added to DEF-EXTERNAL-SERVER, the function for defining a C/API client to be an external server for Design++. The new argument allows a directory to be sepecified as the current directory when the server program is started. CURRENT-DIRECTORY defaults to d++:bin;win32-i86.

  NOTE THAT THE CURRENT-DIRECTORY IS PASSED TO A SERVER'S STARTER FUNCTION. THIS MEANS THAT IF A SERVER HAS A CUSTOM STARTER FUNCTION INSTEAD OF THE DEFAULT STARTER-FN, THEN THE CUSTOM STARTER FUNCTION NEEDS TO BE CHANGED TO ACCEPT THE NEW THE KEYWORD ARGUMENT, CURRENT-DIRECTORY.

  Thus, the starter function is called with five arguments: the server-name, server-pathname, a list of arguments passed to start-<server-name> itself, :current-directory and :show-window.

DESIGN++ CORE

---

* Exiting Design++ fixed to properly handle the loading of <d++>\misc\d++-exit.lisp file.

* Detection of user specific initialization file, d++-init.lisp, made more robust.

* PARENT argument renamed to ASSEMBLY in the following Design Rule Macros
  - :N
  - :?1, :?2, :?3,...,:?10
  - :FIRST, :SECOND, :THIRD,...,:TENTH
  - :LAST
  - :ANY

  The name ASSEMBLY is a better fit than PARENT when referring to a component's assembly in a product structure as the 'parent' is typically used for referring a class parent in an inheritance hierarchy.

* Parsing of valueclasses
  - :EQUALS

:INSTANCE-OF
:SUBCLASS-OF
fixed to catch the error of passing more than one argument to the valueclass.

* Editing an attribute's comment, display name, and value display name fixed to mark the KB as having been modified. Thus, the user will be prompted upon exiting whether or not to save the KB.

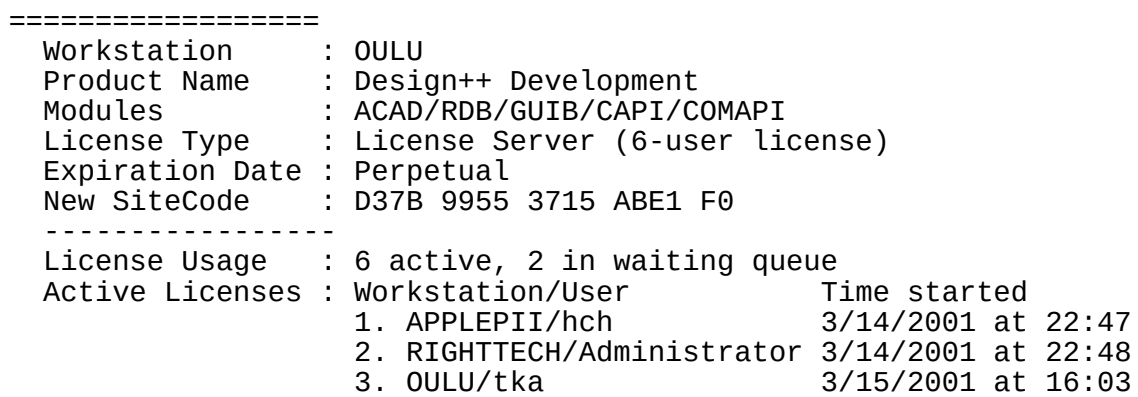
* Design Rule mechanism, including the Rule Editor, enhanced to recognize additional rule definition macros. Note that all the new rule macros need to expand to the classic :! macro.

* Modifying existing (FrOBS-level) slot/facet value sets by either appending new values or removing existing values fixed to be non-destructive. This eliminates the unwanted side effects caused by destructive value set modifications.

* Handling of modified component-attribute-value triplets in external data sources revisited:
  - Fixed to process only the primary component class and its instances in indirect component references. Assembly classes are now skipped and should no longer have their attribute values removed unnecessarily.
  - Processing of changed triplets optimized by taking into account the source of each attribute's value (local, rule, inherited, or external).
  - Determining when a triplet's value has changed fixed to coerce the value first to match the attribute's valueclass.

* Design Rule Editor (DRE) fixed to always retrieve the design rule as-is from a rule-inheriting class attribute and NOT to substitute
"(:! <name-of-the-rule-defining-class> ..." with
"(:! <name-of-the-rule-inheriting-class> ...".
Also, internal design rule representation unified for all the different methods of attaching a rule, i.e., DRE, UIP, Emacs, or evaluating a rule expression, (:! comp att body).

* Design++ License Manager (File>License Manager...) dialog enhanced to display also the network license usage information. Here's an example what the new display could look like.



```
                        4. MUDPII/aax              3/15/2001 at 16:18
                        5. ULLA/dmb                3/15/2001 at 16:19
                        6. PII300/lnicoles         3/15/2001 at 19:09
      Waiting Queue   : Workstation/User
                        1. UPPSALA/str
                        2. PEACHPII/aka
    ==================
```

The network license usage information is also included in the dialog informing the user that there is not enough licenses available when he tries to start Design++.

Note that for standalone licensing this patch does NOT change the License Manager display.

* A new Design++ relation MOUNTED-ON (and its opposite relation MOUNTS) introduced. It is an outcome of our efforts to make the geometric modeling significantly easier by automating the details of inter-component positioning and orientation through the use of predefined relations like MOUNTED-ON.

  The MOUNTED-ON relation comes with a symbolic topology language that allows components to be positioned relative to one another as they are related. For example, a table-leg can be positioned to a corner of a table-top by relating it with the MOUNTED-ON relation:

```
(dpp-relate
 (list (:? table-leg) :vertex 5)
 (list (:? table-top) :vertex 1
       :inside 20 10 0)
 :mounted-on)
```

  Or, with a rule for table's relations attribute:

```
(:! table relations
    (list :mounted-on
          (list (:? table-leg) :vertex 5)
          (list (:? table-top) :vertex 1
                :inside 20 10 0)))
```

  Based on the location definition, the MOUNTED-ON relation uses :when-related (and :when-unrelated) methods to make the related component (mounted) dependent on the location, rotation, and dimensions of the component (base) it is related to. You might just never have to write another geo_loc or geo_rot rule!

  This enhancement is delivered with a PDF document 'Mounted On Relation' (in the d++:documentation; directory), which gives an overview of the new relation concepts and the capabilities of the MOUNTED-ON relation. Take a look!

  The enhancement also includes 3 example projects (in the d++:projects; directory) using the MOUNTED-ON relation:

```
table-with-mounted-on-relation
```

spiral-staircase-with-mounted-on-relation
vessel-exercise-with-mounted-on-relation

To explore each project, first create a new model using the product structure and external data files provided, and then request the geometry. Before creating a model make sure that the 'Lock External Attribute Values' option is checked off in the 'Design++ Core' tab of UIP's 'File>Preferences...' dialog.

The emphasis is to demonstrate how using MOUNTED-ON relation can simplify the geometric modeling process. So, be sure to check (and modify) each project's design rules, especially the rules for substructure and relations attributes. Also, note the differences between this version of the vessel-exercise and the one that was used for training.

You may also check how changes are handled. With each demo project try moving and/or rotating the base (the component to which other components are mounted on). Also, changing the dimensions of either the base or mounted components is interesting to observe. In addition, specific to each project you may try the following changes.

For the table demo, modify the value of table's top-type (square or round).

For the spiral-staircase demo, modify the value of spiral_stairs' total_rise (range: 40 to 160).

For the vessel demo, modify the value of vessel's operating_pressure (100, 200, or 300), or the vessel's orientation (horizontal or vertical).

Note that the MOUNTED-ON relation is supported in both development and runtime versions. Also, models saved with one version can be loaded with the other version.

This enhacement represents the current state of development and should be considered as an alpha release. The reason for this early release is to collect feedback for further development, thus all feedback is greatly appreciated.

Enjoy!

* Erasing an attribute's comment in UIP's Component Editor modified to be handled as comment removal. If the attribute has comments in several languages, the user is prompted whether or not all the local comments should be removed, so that they can be inherited.

* Design Rule Macro :LOCAL-ROTATION (component &rest rotations) fixed to properly evaluate different types of rotations, like local bindings, special variables, constants, functions, and macros. See examples below.

Note that an expression with an axis keyword (:X, :Y, and :Z) as its

first element is always considered as axis-rotation pair, i.e., even if the keyword has a function binding, the expression is not evaluated.

```
(:! transformer geo_rot ;;EXPRESSIONS
    (let ((*forty-five* 45))
      (:local-rotation (:?1 wall)
                       (:x 45)
                       (list :Y *forty-five*)
                       ((if t :z :y) (- *forty-five*)))))

(:! transformer geo_rot ;;LOCAL BINDINGS
    (let ((y-rot '(:y 45))
          (z-rot (list :Z 45)))
      (:local-rotation (:?1 wall) y-rot z-rot)))

(defvar *rot-v* '(:y 35))
(:! transformer geo_rot ;;SPECIAL VARIABLE
    (:local-rotation (:?1 wall) *rot-v*))

(defconstant *rot-c* '(:y 75))
(:! transformer geo_rot ;;CONSTANT
    (:local-rotation (:?1 wall) *rot-c*))

(defun my-rotations (&optional (axis :X) (angle 45))
    (list axis angle))
(:! transformer geo_rot ;;FUNCTION
    (:local-rotation (:?1 wall) (my-rotations :y 78)))

(defmacro my-rot (axis angle)
  `(list ,axis ,angle))
(:! transformer geo_rot ;;MACRO
    (:local-rotation (:?1 wall) (my-rot :y 76)))
```

* Design rule parsing fixed to expand macro calls in their proper lexical environment, not in the environment from which the parsing was initiated. For example, parsing the rule

```
(:! floor ax
    (let ((count 0))
       count))
```

  failed to detect the local binding COUNT as it is also the name of a function.

* Attribute values retrieved from external data source files are no longer locked by default.

DYNAMIC CONFIGURATION ENHANCEMENTS

---

* Retrieving values from external data sources fixed not to issue warnings about non-existing indirect component references.

* A warning about non-existing component for design rule macro :N
  modified to include the assembly under which the referenced
  component was searched. This will help to identify the part of the
  model from which the missing component was expected to be found.

* Design++ Function DPP-RULE-ACTIVE-P fixed to detect design rule
  status also during attribute value redetermination.

```
;|| DPP-RULE-ACTIVE-P (component attribute)
;||  PURPOSE:
;||   Checks whether the design rule for a COMPONENT'S ATTIBUTE is
;||   currently active or not. An active rule is one that is being
;||   executed and has not yet returned.
;||  ARGUMENTS:
;||   component:
;||    Component (frame-or-ref)
;||   attribute:
;||    Attribute (symbol)
;||  RETURNS:
;||   T (rule is active) or NIL (rule is NOT active)
;||  EXAMPLE:
;||   (dpp-rule-active-p 'table.s1234 'geo_length)
;||   ==> NIL
;||
```

* Referencing to a non-existing component or attribute using design
  rule macros outside of a rule, e.g., from the Command Interpreter,
  fixed to display a warning dialog and then return with nil. Both
  references used to fail and enter the debugger.

* Redetermining an attribute without a design rule modified to try to
  retrieve its value from external data sources, as the dependencies
  could have been created as a result of indirect component
  referencing in the external data.

* The issue of when the user should be prompted for the value of a
  referred attribute revisited. Now, the user is prompted for the
  value also when processing delayed attributes (either in
  redetermination or in dynamic instantiation) IF the referred
  attribute does NOT have a design rule (and, of course, has no other
  source for its value).

* Warning about referencing to a non-existing attribute modified NEVER
  to display a dialog whether within a design rule or not. Warning is
  now printed only to the Command Interpreter.

* Component deletion fixed to disconnect all the relations required by
  the deleted component. It used to leave the relations in place.

* Prompting the user during redetermination modified NOT to prompt for
  a referred attribute value if the dynamic instantiator is known to
  restructure the model. This eliminates the user being prompted for
  attribute values of components which are marked for deletion and,
  thus, may not have their values available from external data files

or from other sources.

* Dynamic Instantiator fixed to round all negative integers to zero in substructure descriptions in *substructure*, role, and NR-<class name> attributes. Thus, no more anti-components!

* Dynamic Configurator optimized to handle API (C, COM, Lisp), including UIP, initiated deletion of model components, library classes, and design rules as a single transaction, i.e., changes are propageted into the model only after the operation has otherwise completed.

* Parsing of a component's relation specification with relation initialization data fixed for the Dynamic Configurator. This fixes the problem where the Dynamic Configurator failed to disconnect relations established via relation specifications with init data.

* Handling of relation specifications (in the *relations* attribute) with the first specification being empty fixed. Thus, specifications of the form

        (nil <rel-spec1>...<rel-spec>)

  are now handled OK.
  Also, user-initiated explicit removal of *relations* attribute's value, e.g., via UIP, fixed.

* Rule selection for change management fixed to avoid cycles during change propagation.

INSTALLATION
---

* AutoCAD ADS link is no longer supported, though this release still included the files for the ADS link.

* Install program has support for MicroStation V8, but currently it's not enabled as Design++ currently doesn't support MS V8.

* Some preliminary test has been conducted with Windows XP and it seems to be working fine.

* Alpha version Visio link is included. Note that Visio interface doesn't ask for the DPPCADEXECPATH as that is not used, instead Windows registry is used to start Visio though COM.

* The path for the licesence server can be specified when selecting network license and doing custom setup.

C/API
---

* Libraries are compiled with Microsoft Visual C++ 6.0sp5

* dppComponentGetAssemblyAndPartList fixed.

* Bug in debug output when using dppCommSetDebugEnabled(TRUE) usage fixed.

COMAPI
---

* Added an alias dppCommInitializeClient for dppStartCOM function.
  This name is same as used in C/API.

* Added an optional string type parameter to dppStartCOM function.
  If a client name is not defined as a string, then default name
  'dppCOMserver' is used.

* Added a new event dppEventProc to COM interface. It has 5 parameters:
  Long lMessageId, Long lMessageType, Long lMessageReturnType,
  Long lMessageLen, and String strMessage.

* Now all asynchronous messages sent by dppCOMserver are forwarded to
  client application dppAsyncNotify event handler as well
  as dppEventProc event handler. There is no need to modify
  existing client applications and developers can smoothly
  take advantage of new event.

CAD INTEGRATION MANAGER (CIM)
---

* dppCadGetSymbolLibraryName returns now dppStringArray *
  instead of string. The reason for the change is that previously
  the separator character was space character and so symbol library
  specification with space in name couldn't be used.

* Modified to handle string or list of strings in symbol library
specification
  and filter them trough translate-logical-dpp-pathname.
    (dpp-cad-register-query-symbol-library-function
       #'(lambda () '("abc" "d++:xyz" "projects:xxx" "D:/symbols")))
  Resulting specifications could be:
     abc d:/d++50/xyz d:/d++50/projects/xxx D:/symbols

* Some minor fixes.

* Primitive style and weight arguments were incorrectly specified as integers

  in header files. Fixed them to be of type 'dppChar *'

* Fixed handling of NULL timestamp

* In primitive symbolWithAttributes the attribute parsing skipped empty
  string values.

* Symbol name was always capitalized in symbol primitives.

* dpp-cad-start has new optional argument show-window.

* At project closing time, project specific settings are now reset.

* New functions
      dpp-cad-ready-p
      dpp-cad-register-query-show-window-function
      dpp-cad-register-project-closing-function
      dpp-cad-register-cad-has-registered-function
      dpp-cad-register-cad-has-unregistered-function

MICROSTATION LINK
----

* MicroStation Interface has preliminary support for MicroStation V8.
  Symbol 'msv8' is used as identification for MS V8.

* Implemented revert model functionality. If normal backup file exists
  link will switch to it, otherwise to backup file in MS_TMP. If no backup
  files exists, link will ask to do UNDO ALL.

* Implemented saveas functionality.

* Changed the about box not to show old static Design++ version text, but
  instead to show the MS version used for compilation and currently
  running MS version.

* Changed the autocompress feature to happen now after 1000 delete
  operations instead of the old setting of 300 create+delete operations.

* Fixed a bug, where directory <proj><msdir> was created instead of
  <proj>/<msdir>

AUTOCAD/ARX LINK
----

* ADS link is not supported anymore, though the interface files are
  still included in the distribution.

* Supports AutoCAD 2002 and 2000i

* Symbol handling modified to allow full pathnames to be specified.

* Symbol handling modified:
  - Symbol directory setting is queried from Design++ when drawing is opened.
  - Symbol is searched recursively from defined symbol directories.
  - Directory <project>/arx2000/"symbols" is used as default directory for

```
symbols.
  - Full pathnames can be used.
  - Symbol search order was incorrect when multiple symbol directories were
specified.
  - Symbol library specifications are filtered through
dpp-translate-logical-pathname.
  - Example with DPP=C:/d++ and project=geo_test
      (dpp-cad-register-query-symbol-library-function
         #'(lambda () '("abc" "d++:xyz" "projects:xxx" "D:/symbols")))
     Resulting directory specifications:
        d:/d++50/projects/geo_test/arx2000/abc
        d:/d++50/xyz
        d:/d++50/projects/xxx
        D:/symbols

* Fixed some bugs.

* DIMASO is not used in Acad 2002, instead using DIMASSOC.

* Created functions to execute AutoCAD commands from Design++. See
  documentation for details about
        DPP-ACAD-SEND-COMMAND
        DPP-ACAD-SEND-STRING-TO-COMMAND-LINE
        DPP-ACAD-EVAL-AUTOLISP

* New user functions:
        DPP-CAD-READY-P
        DPP-CAD-REGISTER-QUERY-SHOW-WINDOW-FUNCTION
        DPP-CAD-REGISTER-CAD-HAS-REGISTERED-FUNCTION
        DPP-CAD-REGISTER-CAD-HAS-UNREGISTERED-FUNCTION
        DPP-ACAD-REGISTER-QUERY-ARGUMENTS-FUNCTION

* Added some functionality to try to make sure that (dpp-eval-alisp) gets
  executed in case the command is lost or canceled in AutoCAD command line.

* Polyline argument types of style and weight are now handled correctly
  internally. Weight setting is recognized, but using it doesn't seem to
  be working either because of usage of 3dpolyline or ads_entmake.

* Design++ Function DPP-PROJECT-CAD-PATHNAME fixed to return a
  proper pathname for AutoCAD 2000.

* Better error reporting in case ARX link can't connect to Design++.


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

* Some bug fixes.

* License dialog is now resized correctly at update.

* The font used in prompt text can be specified for some prompt dialogs.
```

* Some prompt dialogs have optional checkbox.

* Prompt-and-read dialog has optional browse/help button. Argument
  browse-button-title specifies the title show in the Help button. If
  Help button is pressed value ^E/05 is returned.

* prompt_expression_with_dialog_box has optional third button.

* In some situations the correct font was not used.

* Y-OR-N dialog's Windows 'X' button is now handled correctly to mean
  'cancel'

DESIGN RULE EDITOR (DRE)
---

* Direct access into Design++ documentation for functions is available
through
  command 'Show Documentation'

* Some bug fixes.

* Fixed some font problems.

* Fixed to add the comment prefix, ";;;", to the beginning of each comment
  line. Previously the prefix was added only for the first comment line.

* Fixed to always retrieve the design rule as-is from a rule-inheriting
  class attribute and NOT to substitute
    "(:! <name-of-the-rule-defining-class> ..."
  with
    "(:! <name-of-the-rule-inheriting-class> ...".

EMACS INTERFACE
---

* Direct access into Design++ documentation for functions is available
through command
  d++-show-documentation, which is bound into 'Control-C Control-f'.

* elidoc mode to show Common Lisp function arglist in echo area is loaded by
default

* eldoc mode to show ELisp function arglist in echo area is loaded by default

* Visual Basic Mode editing is loaded by default

* Emacs version upgraded to from GnuEmacs 20.4 to GnuEmacs 21.1

* Fixed an Emacs deadlock problem caused by UIP initiated commands trying
  to raise the Emacs window. This problem prevented, e.g., UIP initiated

Design Rule file browsing in Emacs.

* Requesting (from UIP) the creation of a new design rule in Emacs and requesting to edit in Emacs a design rule that does not have a rule file associated with it fixed to work properly when Emacs in being run with the new Windows mode.

* Some support for XEmacs added. In order to use XEmacs, the D++.bat file needs to be edited to point to XEmacs executable, the GNUEmacs environment settings replaced and all ELC files need to be removed and recompiled in <d++>/emacs-interface

* Added support for ':rule' as rule macro name.

* Upgraded window mode code to latest version cua.el 2.8

* Some modification in exit handling, for better protection against accidental exits.

USER INTERFACE PROGRAM (UIP)
---

* On Windows platform the preferences are now stored in the Windows registry.

* Reallowed '#' and '/' in symbols and added check for ":' and '#' not to be allowed to be first character of the symbol.

* Fixed a bug in attribute origin display in component editor.

* Deleting a project deletes also the project specific user preferences.

* Fixed to update magnifier when :ONE-OF type value is changed.

* Double confirmation added for library deletion.

* Added 'Browse Relations' menu item into appropriate places to enable access to the new Relation Browser.

* Fixed some dialog placing problems.

* Multiple grapher windows (Model/Library editors) can now be opened.

* Part of the model/library graph can be opened into a separate subgrapher window.

DATABASE LINKS
---

* Documentation for obsolete standards and model read/write

functionality removed. Both ODBC and ORACLE links are now described in same document. New usage examples added.

ODBC LINK
---

* Variable length data is now handled correctly.

* Basic support for binary types added. When binary SQL data is converted to character C data, each byte (8 bits) of source data is represented as two ASCII characters. These characters are the ASCII character representation of the number in its hexadecimal form. For example, a binary 00000001 is converted to "01" and a binary 11111111 is converted to "FF".

MISC
---

* The program dppdiag is included in <d++>\bin\win32-i86\ directory. It can be used to get some system information and possibly to help in diagnosing some Design++ related problems. It is a console application and needs to be started from a command prompt or given argument '-pause' to keep it from closing its console window. Running dppdiag without any arguments shows some help information.

=====