Design^{++} 4.2 Release Notes for Windows NT

We are pleased to announce the release of Design^{++} 4.2 for Windows NT. This is the second major release of Design^{++} for the PC platform after the highly successfully release of Design^{++} 4.1.

Highlights

Design^{++} 4.2 includes several new modules and enhancements, the most important ones being the following:

COM/API

Design^{++} Component Object Model Application Programming Interface (COM/API) is a library of Windows COM functions that let any COM compliant application, like Visual Basic, communicate with Design^{++}. The emphasis is in supporting Visual Basic as a GUI development tool for Design^{++}.

AutoCAD 2000 Link

New ObjectARX-based AutoCAD link supports both AutoCAD release 14 and 2000. The old ADS-based AutoCAD Link is still supported with AutoCAD release 13 & 14.

MicroStation/J Link

MicroStation Link supports now also MicroStation/J. As a result, Design^{++} now integrates with MicroStation 95, MicroStation SE, and MicroStation/J.

Dynamic Configuration Enhancements

- Dynamic Configurator (DDC) has been optimized to reduce trashing. Also, model creation based on product structure files is now compatible with earlier Design^{++} versions.
- Intelligent Change Manager (ICM) and Execution Order Controller (EOC) have been optimized significantly. They also have more robust exception (cycle) and error handling.

- Design rule compilation has been upgraded to initiate proper change propagations in all situations.
- Substructure attribute is now inherited automatically to all library classes of type assemblies as they most likely will need it. This simplifies the use of the dynamic configuration capabilities.

Windows Mode in Command Interpreter and Design Rule Editor

Emacs-based Command Interpreter and Design Rule Editor come with a new Windows-like command set including standard Open, Close, and Print dialogs.

Multiprocessing Support Enhanced

The new native NT threads-based multiprocessing implementation fixes the infamous “sleeping” problem, where Design++ occasionally failed to detect client requests.

Network Licensing

Licensing mechanism has been extended to support network licensing. Now any workstation capable of running Design++ as a stand-alone installation can also host a license server installation. Any client workstation connected to the same local area network as the server will be able to run Design++.

Due to the network licensing support, SiteKeys issued for Design++ 4.1 need to be renewed. To smooth the transition each workstation is granted a new 10-day evaluation license for Design++ 4.2 regardless of their prior license status.

Automated Installation

Automated installation program has been upgraded to support license server and client installations. It also supports installation upgrades.

License Manager Dialog

To facilitate the handling of licensing issues, Design++ has a new License Manager that displays the current license status and allows on-the-fly license upgrades.

New Documentation - Available Online

Design++ User’s Manual has gone through a complete overhaul. In addition, all Design++ Function Manuals have been merged into a new Design++ Lisp/API Manual.

Starting with this release all manuals are available in PDF format on the delivery CD.

Year 2000 Compliant

The representation of the year in Design++ knowledge base time-stamps has been extended to use four digits. After this modification and related testing, Design Power believes that Design++ will conform with the year 2000 requirements.

Design++ Functions (Lisp/API)

▲ Design++ Function
DPP-RELATED-TO

modified to handle correctly relations and classes with identical names.

▲ Design++ Function
DPP-COLINEAR-P

fixed. Also, indirectly affected are the following other Design++ Functions.

DPP-POINT-WITHIN-SEGMENT-P
DPP-DROPPED-POINT-WITHIN-SEGMENT-P
DPP-DROP-POINT-ON-LINE
DPP-DROP-POINT-ON-LINE-SEGMENT
DPP-POINT-LINE-ORTHO-DISTANCE
DPP-POINT-TO-LINE-DISTANCE
DPP-PERPENDICULAR-TO-PLANE

▲ Design++ Function **DPP-POINT-TO-LINE-DISTANCE** fixed to handle correctly the two special cases: coincident line points and colinear points. Fixed also to make sure that the calculation remains within real numbers (and does not fall into complex numbers) when dealing with near-colinear points.

```
DPP-POINT-TO-LINE-DISTANCE (p lp1 lp2 &optional (ndecimals 5))
   PURPOSE:
      Returns the shortest distance between a 3D point and a 3D line
   ARGUMENTS:
     p:     3D point (list)
     lp1:   3D end-point of a line (list)
     lp2:   The other 3D end-point of a line (list)
     ndecimals (optional):
            Number of decimals in the result, default = 5 (integer)
   RETURNS:
      Point-line distance. There are two special cases. If the two
      line points are coincident, point-point distance is returned.
      If all three points are colinear, 0.0 is returned.
   EXAMPLE:
      (dpp-point-to-line-distance '(1 0 0) '(0 0 0) '(1 1 0))
      ==> 0.70711
      (dpp-point-to-line-distance '(1 0 0) '(0 0 0) '(0 0 0))
      ==> 1.0    ;Coincident line points
      (dpp-point-to-line-distance '(1 0 0) '(0 0 0) '(10 0 0))
      ==> 0.0    ;Colinear points
```

▲ A new structure specification keyword ':list' added for Design++ Functions **DPP-SHOW-GEOMETRY** and **DPP-REMOVE-GEOMETRY**. The new keyword allows a list of components to be given as the first argument.

```
DPP-SHOW-GEOMETRY (&optional (component (dpp-model-root))
 (structure :component))
   PURPOSE:
      Displays component/component-and-substructure/substructure/
      leaves geometry in CAD
   ARGUMENTS:
      component (optional):
            Component whose geometry is to be calculated
            (frame-or-ref) default = (dpp-model-root)
     structure (optional):
            :component, :component-and-substructure,
            :substructure, :leaves, or :list
            default = :substructure
   RETURNS: NIL
   EXAMPLE:
```

```
DPP-REMOVE-GEOMETRY (&optional (component (dpp-model-root))
(structure :component-and-substructure))
   PURPOSE:
       Removes component's and its substructure's geometry from CAD
   ARGUMENTS:
       component:
            Component whose geometry is to be removed
            (frame-or-ref)
            Default is model's root component.
      structure (optional):
            :component, :component-and-substructure,
            :substructure, :leaves, or :list
            default = :substructure
   RETURNS: NIL
   EXAMPLE:
```

▲ Design++ Function **DPP-COPY-DIRECTORY**

fixed to handle directory copying from one disk partition to another. For example, the following operation was not handled correctly

```
(dpp-copy-directory "H:/test/" "C:/temp/")
```

▲ Design++ Function **DPP-MAKE-DIRECTORY** enhanced to create recursively all the directories denoted by its directory-pathname argument.

```
DPP-MAKE-DIRECTORY (directory-pathname)
   PURPOSE:
      Creates recursively all the directories denoted by
      DIRECTORY-PATHNAME. Does not allow existing directories to
```

```
   be overwritten.
ARGUMENTS:
   directory-pathname (string or pathname):
         Directory structure to be created.
RETURNS:
   Returns DIRECTORY-PATHNAME if any directories were created,
   otherwise NIL.
EXAMPLE:
   (dpp-make-directory "C:/temp/my/very/own/directory/struc-
ture/")
   ==> "C:/temp/my/very/own/directory/structure/"
```

▲ The capability of saving custom Design++ images has been reintroduced for the Runtime version. Image are saved with the new Design++ Function **DPP-DISKSAVE-D++**.

DPP-DISKSAVE-D++ (path restart-function)

```
PURPOSE:
   Saves the current Design++ session into a re-startable
   custom image.
   Disksaving a custom application images is supported only
   with the Design++ Runtime version. Thus, an application
   should be developed and and compiled with the Development
   version and then loaded with the Runtime version to be
   disksaved into an image.
   The easiest way to test your custom image (without
   the Emacs interface) is to create a new copy of your
   Design++ runtime startup script, e.g. d++:bin;d++.bat,
   and replace the only occurrence of d++-run.dxl with
   the name of your new image.
ARGUMENTS:
   path (optional):
         File where the custom image is to be saved. Default
         path on Windows NT is "d++:bin;win32-i86;d++.dxl"
         and on Unix
         "d++:bin;<object-directory-name>;d++.image"
   restart-function (optional):
         Function to be called when the custom image is started.
         The restart function should perform all application
         initializations, like starting the application's GUI,
         before the control is given to the user.
         Note that unlike with Design++ 4.1, the restart function
         should NOT call the standard Design++ restart function.
         Defaults to NIL.
RETURNS:
   Nothing if successful, NIL otherwise. Note that it is not
   recommended to continue to use the Design++ session after
   a new image has been saved.
EXAMPLE:
(dpp-disksave-d++
   (dpp-translate-logical-pathname
         "d++:bin;win32-i86;pbar.dxl")
         #'pbar-restart-function)
```

▲ Design++ Functions **DPP-GET-EXTERNAL-DATA-FILES** and **DPP-ATTACH-EXTERNALS** modified NOT to filter out non-Lisp (*.lisp) files. Design++ **DPP-ATTACH-EXTERNALS** fixed also to return either T or NIL as documented.

```
DPP-GET-EXTERNAL-DATA-FILES ()
   PURPOSE:
      Returns list of external data files of the current project
   ARGUMENTS:
   RETURNS:
      List of external data files (list of strings)
   EXAMPLE:
      (dpp-get-external-data-files)
      ==> ("eqpt-data.lisp" "pipe-data.lisp" "struct-data.lisp")

DPP-ATTACH-EXTERNALS
(&optional (model-name (dpp-get-current-model)) extarnal-files)
   PURPOSE:
      Attaches external files to model
   ARGUMENTS:
      Optional:
            model-name (symbol): Model-name
            extarnal-files (list of strings):
                  External file names in current project's
                  externals directory.
   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.
```

▲ A new Design++ Function (macro!) **DPP-WITH-AUTO-DEBUG** introduced. It evaluates the expressions of the body and in case of an error automatically :zooms on the error in the Debugger. The output from the :zoom can be directed to a file. Before exiting back to Lisp top-level, the :when-exit expression is evaluated.

```
DPP-WITH-AUTO-DEBUG ((&key output when-exit count) &body body)
   PURPOSE:
      Evaluates the expressions of the body and in case of an error
      automatically :zooms on the error in the Debugger. The output
      from the :zoom can be directed to a file. Before exiting back
      to Lisp top-level, the :when-exit expression is evaluated.

      Note that in order to improve the performance Design++ 4.2
```

```
   Runtime version does NOT save call arguments by default.
   The drawback is that the amount of information available in
   the :zoom output is also reduced. The saving of call arguments
   can be turned on by evaluating expression
   (if (dpp-getenv "DPPNOEMACS")
    (tpl:do-command "args" :save t)
    (s::emacs-do-cmd
    ;;Be sure to flush the input buffer (#\Control-C #\Control-U)
    ;;first.
     (format nil "~C:args :save t ~C"
      (code-char 190) #\return)))
   in <d++>/misc/d++-startup.lisp file or with the top-level
   command
         D++(): :args :save t
ARGUMENTS:
   Keyword:
         output (pathname or stream):
               Defines where the output from :zoom is to be
               directed. Defaults to *standard-output*.
         when-exit (symbol or function):
               Function to be evaluated before exiting back
               to Lisp top-level.
         count (integer or T):
               Integer number of frames to print when
               :zooming on an error. Value T means print all
               applicable frames. Defaults to 10.
   body (expressions):
         A set of expressions to be evaluated (an implicit
         progn).
RETURNS:
   Unless there is an error returns what the last body expression
   returns. In case of an error returns what :when-exit
   expression returns.
EXAMPLE:
(in-package :design++)

(defun div-by-zero (n)
  (/ n 0))

(defun notify-error-and-zoom-output (pathname)
  (dpp-prompt
   t (format nil "An unhandled error condition has been~
                signalled:~%~A~%~%The output from Debugger~
```

```
                :zoom command has been directed to file: ~A"
          (with-open-file (stream pathname :direction :input)
            ;;Read the error condition from file.
            (read-line stream nil))
          pathname)))

(defun test0 ()
  (div-by-zero 1))

(defun test1 ()
  (dpp-with-auto-debug
   (:output "c:/temp/debugger-zoom-output"
    :when-exit #'(lambda () (notify-error-and-zoom-output
              "c:/temp/debugger-zoom-output"))
    :count 3)
   (div-by-zero 1)))

(defun test2 ()
  (dpp-with-auto-debug
   ()
   (div-by-zero 1)))
```

▲ Two new Design Functions **DPP-ASSEMBLY** and **DPP-ALL-ASSEMBLIES** introduced. Actually, they are synonyms for existing functions **DPP-PARENT** and **DPP-ALL-PARENTS** with identical functionality. The word '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.

DPP-ASSEMBLY
(component &optional class-name (member-descendants-p t))

```
   PURPOSE: Get product structure parent of component
   ARGUMENTS:
      component: A model component (frame-or-ref)
      class-name (optional):
            The name of a library class (symbol-or-ref)
      member-descendants-p (optional):
            Consider all descendants or just direct instances
            (T (default) or NIL)?
   RETURNS: parent component
   EXAMPLE:
      (dpp-assembly 'mixin_vessel.S5381)
      ==> #<FRAME PROCESS_PLANT.S5157 A-PLANT>

   Added two new optional arguments, CLASS-NAME and
   MEMBER-DESCENDANTS-P.
   If the new CLASS-NAME argument is given, then a component's par-
   ents are searched recursively until a parent, which is of type
   class, is found.
   The second new optional argument MEMBER-DESCENDANTS-P (defaults
   to T) determines whether all parents that are descendants of the
   class or just the direct instances of the class are considered
```

```
   while searching for the parent.
   Examples:
      (dpp-assembly (dpp-get-component 'leg-end) 'table)
      (dpp-assembly (dpp-get-component 'leg-end) 'parts)
```

DPP-ALL-ASSEMBLIES
(component &optional class-name (member-descendants-p t))

```
   PURPOSE: Get recursively all parents of component
   ARGUMENTS:
      component: A model component (frame-or-ref)
      class-name (optional):
            The name of a library class (symbol-or-ref)
      member-descendants-p (optional):
            Consider all descendants or just direct instances
            (T (default) or NIL)?
   RETURNS: List of all parent component references
   EXAMPLE:
      (dpp-all-assemblies 'mixin_vessel.S5381)
      ==> (#<FRAME PROCESS_PLANT.S5157 A-PLANT> ...)
```

▲ Design++ Function **DPP-VISIT-LIBRARY** modified not to visit each class more than once. The function used to follow each subclass link and, thus, visit those classes with multiple superclasses more than once. This modification fixes also 'Saving Rules To File...' in UIP's Library Editor not write each rule to a file more than once.

DPP-VISIT-LIBRARY (library-name fn &optional root)

```
   PURPOSE: Execute function to library classes.
   ARGUMENTS:
      library-name: Library name (symbol)
      fn: Function to be executed with library components
      root (optional): Root component in the library (symbol)
   RETURNS: NIL
   EXAMPLE:
      Following example returns all library components with
      'geo_type' attribute defined.

    (let ((r nil))
      (dpp-visit-library
       'plant #'(lambda (class)
          (when (and
           (dpp-attribute-exists-p class 'geo_type)
           (dpp-get-value-no-calc class 'geo_type))
      (push class r))))
      r)
```

▲ Design++ Function **DPP-VEC-NORM** fixed to convert integer arguments into double-floats just like some Lisp functions, e.g., sqrt. Without the conversion some

equality tests using the default *EPS* value of 1.0E-9, which is smaller that the precision of single-floats, caused problems.

```
DPP-VEC-NORM (vec)
   PURPOSE: Returns norm (length) of vector
   ARGUMENTS:
      vec: Vector (list of x, y and z)
   RETURNS: Positive scalar (float)
   EXAMPLE:
      (dpp-vec-norm '(1 1 1))
      ==> 1.7320508075688772
```

▲ New Design++ Function **DPP-CLEANUP-PROGRAM** introduced as an alias for sys:os-wait and sys:os-reap-subprocess. In order to clean up system resources, **DPP-CLEANUP-PROGRAM** needs to be called after a process started by **DPP-RUN-PROGRAM** with the :wait keyword argument nil has finished. Calling **DPP-CLEAN-UP-PROGRAM** returns also the exit status for the process.

```
DPP-CLEANUP-PROGRAM (&key wait pid)
   PURPOSE:
      If a process is started by the dpp-run-program with the wait
      keyword argument nil, then the process will remain in the
      system after it completes until either Lisp exits or Lisp
      executes dpp-cleanup-program to inquire about the exit
      status. To prevent the system becoming clogged with processes,
      a program that spawns a number of processes with :wait nil
      must be sure to call dpp-cleanup-program after each process
      finishes. Exactly what dpp-cleanup-program does depends on
      the status of spawned processes and the keyword arguments.

   ARGUMENTS:
      :pid
            The pid argument controls what processes might be
            considered on by dpp-cleanup-program. If pid is -1
            (the default), all processes are considered. If pid is
            0, only processes in the same process group (as the
            executing Lisp image) are considered. If pid is a
            positive integer, only the process with that process id
            is considered. By default (pid = -1) if there are any
            processes started by dpp-run-program with the argument
            :wait nil which have exited but for which
            dpp-cleanup-program has not been run, one of them is
            selected by the operating system and its status and
            process id are returned in that order as multiple
            values.
        :wait
            If there are no such processes which have exited but
            there are processes which are still running, then the
            behavior of dpp-cleanup-program depends on the :wait
            arguments. If it is T (the default),
            dpp-cleanup-program will wait (disabling
            multiprocessing, if necessary) until one of the running
```

```
          processes exits.

   RETURNS:
      If :wait was T then process's status and id are returned.
      If there are no running processes and :wait=NIL,
      dpp-cleanup-programs returns immediately with the values nil,
      nil. If :wait is NIL, dpp-cleanup-program will return two
      values: nil and the pid argument to dpp-cleanup-program
      immediately. 0 as the single returned value indicates that
      there are processes running but none to clean up, in contrast
      to nil -- no processes running, none to clean up, and multiple
      values -- a process was cleaned up.
```

▲ Design++ Performance Monitoring Functions/Macros

DPP-MONITOR
DPP-WITH-MONITORING
DPP-REPORT-MONITORS

re-introduced. They were accidentally left out from Design++ 4.2 Beta release. Note that the monitoring utilities are only supported with Design++ development version. For more information on optimizing your Design++ application, see file <d++>\documentation\optimizing-applications.txt.

Design++ Core

- Exiting Design++ fixed to wait also until the socket client that initiated the exit has properly exited.
- Design Rule compilation fixed to correctly handle quoted expressions within rules.
- Reading external data files modified to allow component-attribute-value triplets to spread over multiple lines. For example, the following is now a valid entry

```
mixin_vessel geo_loc (1.0 ;X
                      2.0 ;Y
                      3.0) ;Z
```

- The batch file for starting Design++, <d++>\bin\d++.bat, modified to use %HOMEDRIVE%%HOMEPATH% instead of C:\ as the default value for HOME if HOME is not set. Design++ user-interface resource files (*.vr) and Emacs startup file .emacs are created into HOME.
- Two new variables, DPPNOEMACS and DPPNOGUI, introduced for the d++.bat batch. The new variables, when set, allow Design++ to be started without Emacs and/or UIP.
- Design++ startup file <d++>\misc\d++-startup.lisp modified to load both Design++ ODBC link and Dynamic Configuration Enhancements (patch-config-0). Thus, the projects no longer have to take care of the loading in one of their function files. If the enhancements are not needed, the load expressions can simple be commented out. Note that the ODBC link overrides the Oracle RDB link.
- Design++ restart function modified to allow custom images to be started from any directory, not just from the default Design++ binary directory <d++>\bin\win32-i86\. Starting an image from a local disk partition can be significantly faster than from a file server.
- Two new Design Rule Macros **`:ASSEMBLY`** and **`:ALL-ASSEMBLIES`** introduced. Actually, they are synonyms for existing macros **`:PARENT`** and **`:ALL-PARENTS`** with identical functionality. The word ‘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.

`:ASSEMBLY (component &optional class member-descendants-p)`

```
A design rule macro for referring to the product structure
assembly to which COMPONENT belongs to. Note that :assembly
is a synonym to another design rule macro :parent with
identical functionality.

If optional argument CLASS is given, then COMPONENT’s
```

```
      assemblies are searched recursively until an assembly, which
      is of type CLASS, is found.

      The second optional argument MEMBER-DESCENDANTS-P (defaults
      to T) determines whether all assemblies that are descendants
      of the CLASS or just the direct instances of the CLASS are
      considered while searching for the assembly.

   Examples
      (:! floor abc (:assembly floor))
      (:! floor abc (:assembly (:assembly self)))
      (:! floor abc (:assembly floor building)) ;Name
      (:! floor abc (let ((building-class 'building))
            (:assembly floor building-class))) ;Local binding
      (defvar *building* 'building)
      (:! floor abc
            ;;Global (special) variable
            (:assembly floor *building*))
      (:! floor abc
            ;;Expression
            (:assembly floor (progn 'building)))
      (:! floor abc
            ;;First assembly which is a descendant of
            ;;plant_assemblies
            (:assembly floor plant_assemblies))
```

:ALL-ASSEMBLIES (component &optional class member-descendants-p)

```
      Returns recursively all assemblies in a product structure
      hierarchy (not just direct assemblies) of the given
      COMPONENT. When the CLASS is specified, only assemblies, which
      are member DESCENDANTS (not only direct members) of the given
      CLASS, are returned.

      Added a new optional argument MEMBER-DESCENDANTS-P (defaults
      to T) which determines whether to return all related
      components which are member descendants of the class or only
      those which are direct members of the class.
```

- Design++ Runtime version modified to run design rules with the interpreter when there is no compiler available. This allows design rule patches to be loaded into runtime images. The rule patches, i.e., files containing design rule definitions, need to be compiled with a development version before they can be loaded into a runtime image. Note that this does not allow rules to be edited with a runtime version.

- All required system DLLs have been moved under a single directory <d++>\system32dlls\.

- 2 Emacs related startup problems fixed. The first one was caused by Design++ trying to send output to Emacs before the interface had properly initialized. The

second one was caused by the initial input buffer not being properly flushed when the first prompt appeared.

Dynamic Configuration Enhancements

- Dynamic Configurator (DDC) has been optimized to reduce trashing.
- The attribute value prompt modified to use a component's real name unless *trace-rules-with-pnames* is T.
- Model creation based on product structure files is now compatible with earlier Design++ versions.
- Intelligent Change Manager (ICM) and Execution Order Controller (EOC) have been optimized significantly. They also have more robust exception (cycle) and error handling.
- Certain warnings about corrupted dependencies modified to be directed to the *standard-output* instead of a separate dialog requiring user acknowledgment.
- Reporting design rule failures during change propagation modified NOT to report transient failures caused by the changing model. Rules that truly fail during a change propagation are reported at the end of the propagation. Transient rule failures can still be traced by turning the rule trace on.
- Design rule compilation has been upgraded to initiate proper change propagations in all situations.
- Design rule compilation fixed to remove the attribute values for rule-inheriting classes with instances.
- Model creation fixed to handle correctly *substructure* and *relations* attributes with both an inherited value and a design rule.
- The issue of when the user should be prompted for an attribute value revisited. Now, during dynamic instantiation the user is asked for the value of an attribute, which does not have a design rule, regardless of where the attribute is referred from. Attribute values are never prompted from the user during change propagation.
- Handling of NR-<class-name> attributes fixed NOT to set the valueclass to integer for predefined (inherited) attributes.
- Handling of design rule references to non-existing attributes during change propagation fixed.
- Handling of GEO_TYPE's value removals fixed.

- To facilitate the creation of the *substructure* attribute, the attribute is now inherited automatically to all library classes of type ASSEMBLIES as they will most likely need it. If a PART requires the *substructure* attribute it still needs to be created manually or the PART's inheritance needs to be modified so that it becomes also an assembly.

- New keyword argument load-cad-drawing-p added to Design++ Function **DPP-CREATE-MODEL**.

```
DPP-CREATE-MODEL
(model-name &key description
structure-file structure-description
external-data-files (load-cad-drawing-p t)
(reset-cad nil) seedfile)
   PURPOSE: Create new model
   ARGUMENTS:
      model-name (symbol): Model name
      keywords:
            description (string): Description of model
            structure-file (pathname): Structure-file pathname
            structure-description (list or string):
                  Structure-description
            external-data-files (list of pathnames):
                  External-data-files to be added to model
            load-cad-drawing-p (T (default) or NIL):
                  Whether or not create a drawing in CAD. Note
                  that :reset-cad is ignored when load-cad-drawing
                  is NIL.
            reset-cad (T or NIL (default)):
                  One of nil, :remove-drawing,
                  :save-&-remove-drawing and :keep-drawing. If nil,
                  then user is asked with a dialog box to choose one
                  of the above options.
            seedfile (string):
                  Name of the seedfile. If NIL use default seedfile.
   RETURNS: Model-name
   EXAMPLE:
      (dpp-create-model 'newmodel)
      ==> NEWMODEL
      (dpp-create-model 'new-bike
            :description "This is a description"
            :structure-description
            '(bicycle (front_wheel_system) (back_wheel_system)
             (body) (crank_system)))
      ==> NEWBIKE
```

- Cycle detection during change propagation enhanced to catch also infinite propagation cycles that cannot converge because of inconsistent design rules, typi-

cally an application error. For example, the following simple two-attribute cycle will not converge due to inconsistent rules.

```
(:! comp a (:? self b))
(:! comp b (1+ (:? self a)))
```

The new detection capability is intended to assist application developers and is not on by default. To turn the enhanced cycle detection on, a new keyword argument with-cycle-detection is added to Design++ Function **DPP-TRACE-RULES**.

```
DPP-TRACE-RULES
(&key (with-pretty-names t)
(with-values 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 (default) or NIL)
      with-values (keyword):
            Whether or not the trace should also contain attribute
            values (T or NIL(default))
      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
```

▲ The deletion of NR attributes fixed to handle the rare case where an NR attribute is deleted from a library while the attribute is being used in an active model.

Emacs Interface

- Design++ 4.2 on Windows NT comes with a new Emacs version 20.4.1.
- The Emacs interface has a new default Windows mode, which supports:
  - Standard Windows dialogs for choosing printers and selecting filenames for save and open.
  - Windows style menubar
  - Windows style key-bindings supporting, e.g., standard Cut, Copy and Paste (C-x,C-c,C-v)
- Whether Emacs is to be run with the new Windows (default) mode or with the classic Emacs mode is determined during Design++ installation or reconfiguration. The Windows mode can also be enabled with command

  ```
  M-x dpp-win-mode
  ```

  and partially disabled with command

  ```
  M-x disable-dpp-win-mode
  ```

  which restores most of the classic Emacs key-bindings and menubars.
- The Emacs Windows mode provides a emulation of the standard CUA key bindings and manipulating the region where S-<movement> is used to highlight & extend the region. This package allow the C-z, C-x, C-c, and C-v keys to be bound appropriately according to the Windows GUI, i.e.

  ```
  C-z > undo
  C-x > cut
  C-c > copy
  C-v > paste
  ```

  The tricky part is the handling of the C-x and C-c keys which are normally used as prefix keys for most of Emacs' built-in commands. With this mode they still do! Only when the region is currently active and highlighted since the C-x and C-c keys will work as CUA keys

  ```
  C-x -> cut
  C-c -> copy
  ```

  When the region is not active, C-x and C-c works as prefix keys!

  This has a few drawbacks (such as not being able to copy the region into a register using C-x r x), but this mode automatically mirrors all region commands using the [C-x r] prefix to use the [M-r] prefix as well, so you can use M-r x to copy to a register.

The Windows mode is based on “the best of” pc-selection-mode, s-region, and delete-selection-mode packages with some extra features which we think are useful.

A few more details:

- When the region is highlighted, TAB and S-TAB will indent the entire region by the normal tab-width (or the given prefix arg).
- C-x C-x (exchange point and mark) no longer activates the mark (i.e. highlights the region). I found that to be confusing since the sequence C-x C-x (exchange once) followed by C-x C-x (change back) would then cut the region! To activate the region in this way, use C-u C-x C-x.
- [delete] will delete (not copy) the highlighted region.
- The highlighted region is automatically deleted if other text is typed or inserted.
- Use M-r as a prefix for the region commands instead of C-x r. The original binding of M-r (move-to-window-line) is now on M-r M-r.

List of some changed bindings in Windows mode and their classic bindings in Emacs:

Key	Windows mode	Classic Emacs mode
C-c	copy-region-as-kill	prefix key
C-x	kill-region	prefix key
C-z	advertised-undo	suspend-emacs-or-iconify-frame
C-v	yank	scroll-up-command
C-a	mark-whole-buffer	beginning-of-line
C-p	print-buffer	previous-line
C-s	save-buffer	isearch-forward
C-n	find-file	next-line
C-o	open-file	open-line
C-f	isearch-forward	beginning-of-line
C-h	query-replace	help prefix key
f5	insert-time-stamp	-
f1	help	-
S-insert	yank	beginning-of-line
M-insert	yank-pop	-
C-insert	copy-region-as-kill	copy-primary-selection
S-delete	kill-region	kill-primary-selection
C-delete	kill-line	delete-primary-selection
S-tab	back-tab-indent	tab-to-tab-stop

▲ show-paren-mode, which highlights matching parentheses, is now on by default.

▲ auto-fill-mode is now disabled by default for Lisp buffers.

▲ font-lock-mode is now on by default for Lisp buffers.

▲ The menubar menu has been reorganized.

- Environment variable DISPLAY is no longer set on Windows NT as it somehow conflicts with AutoCAD's license manager.
- Default directory is set to DPPPROJECTSPATH.
- The problem, where Design Rule compilation fails if the rule buffer has been fontified, i.e., font colors have been turned on, is now fixed.

  Note that you can fontify a single Emacs buffer with function key F8. To disable autofill mode and enable font-lock mode, i.e., to fontify buffers, for the default Lisp mode, you need to add the following expressions into your HOME_emacs file.

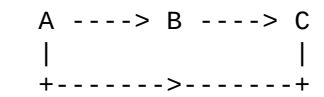
```
(defun dpp-setup-lisp-mode ()
   (auto-fill-mode -1) ;Disable autofill
   (font-lock-mode 1) ;Enable font-lock (to fontify)
)
(add-hook 'fi:common-lisp-mode-hook 'dpp-setup-lisp-mode)
```

User Interfaces

- Font specifications fixed for dialogs with font PHONETIC.FON which caused this font to be used instead of intended MS Sanf Serif.
- About dialog has a new button to launch the default browser pointing to www.dp.com.
- Design++ executables (.exe files) are now displayed in Windows Explorer and Taskbar with the Design Power logo (designpower.ico).
- When minimized Design++ executables are now placed on the Taskbar.
- At startup, if connecting with Design++ fails, the error message is shown in a dialog.

UIP

- Model Editor's 'Geometry' menu item enabled.
- Enabling/disabling of Magnifier's menubar items 'Remove' and 'Unlock' fixed.
- Preference dialog is now the leading dialog for the ColorChooser dialog so that the ColorChooser stays over the preference dialog.
- Icons in Magnifier now have tooltip helps.
- A new 'License Manager' item added to the main dialogs' File menu.
- Several Component Editor updating related problems fixed.
- Several dialog closing and deleting related problems fixed.
- Removing an attribute value through Component Editor fixed.
- Saving a project fixed NOT to prompt the user with a second set of dialogs to save unsaved libraries and/or model after the user has already selected not to save them with the first dialog.
- Inheritance cycle detection fixed for Library Editor. The editor did not allow a direct subclass link between two classes if there already existed an indirect inheritance link between the two. For example, adding a subclass link from A to C is now allowed.



Truly cyclic inheritance links are still blocked. For example, adding a subclass link from C to A is not allowed.

UIS

- To facilitate the handling of licensing issues, the new License Manager displays the current license status and allows on-the-fly license upgrades.
- For Design++ runtime version, displays a new permanent background Design Power logo dialog.
- Modified not use UIP's resource files anymore. Shared resource files used to corrupt in case of concurrent access.
- Dialog for Design++ Function

DPP-PROMPT-FOR-FILE

fixed to expose properly.

Design Rule Editor (DRE)

- On Windows NT, the default Emacs-like key-bindings are now disabled and Windows-style key-bindings are used instead. For example, now

```
Control-X is Cut,
Control-C is Copy, and
Control-V is Paste.
```

- Matching parentheses are highlighted when point is positioned either before the starting parenthesis or after the ending parenthesis. The old method of flashing parentheses when inserting one of the parentheses characters is removed.
- There is a new toolbar (with tooltips) for most common operations.
- The Open/Save file dialogs are fixed to show the correct default directories.
- The default text font on NT is now Courier.
- Handling of UNIX/DOS linefeed/return conventions fixed.
- Parsing fixed to handle correctly Design++ Functions with no arguments, like

DPP-PROJECT-PATHNAME

CAD Interfaces

- Design++ 4.2 integrates with AutoCAD releases 13c4a, 14.01, and 2000, and with MicroStation 95, SE, and J. The following table summarizes the supported CAD systems.

CAD	DPPCAD	in D++	Interface Dir.	Project Dir.
MS 95	mstation	:ms	ms-interface	mstation
MS SE	msse	:msse	msse-interface	msse
MS/J	msj	:msj	msj-interface	msj
ACADr13/ADS	acad	:acad	ac-interface	acad
ACADr14/ADS	acadr14	:acadr14	acadr14-interface	acadr14
ACADr14/ARX	arx	:arx	arx-interface	arx
ACAD2000/ARX	arx2000	:arx2000	arx2000-interface	arx2000

  To request the name of CAD system currently integrated with Design++, use Design++ Function
DPP-CAD-GET-CAD-SYSTEM

- The value of the exported variable *dpp-geo-level* is now the name of the geo_level attribute (GEO_LAYER or GEO_LEVEL).

- GEO_LEVEL, GEO_LAYER, and GEO_COLOR value strings can include space characters.

- Showing and removing geometry optimized by relying only on the existence of *geo-rep* attribute as an indication that the component has a geometric primitive associated with it.

AutoCAD/ADS Interface

- OSMODE is set off before executing AutoLisp free code.
- Fixed to allow Free functions to be called without any arguments.
- When exiting Design++, AutoCAD should now exit more reliably.
- dppHiddenLayer layer created only when needed.
- dppBlock_box block definition created only when needed, that is only when Design++ BOX primitive is used.
- dppBlock_box color set to bylayer.
- Added more error reporting into file operations.
- SEQEND was sometimes created to current layer instead of the designated layer of a primitive.
- Problem, which prevented project-specific AutoCAD startup files (startup.scr) from being loaded, is now fixed.
- Communication with AutoCAD optimized for a single component geometry show/remove messages.

AutoCAD R14/ARX Interface

Changes Since v0.6.6

- Symbol_with_attributes primitive fixed to handle attribute placement correctly.
- acaddpp now registers itself correctly for the drawing with ads_regapp.

ARX Link v0.6.10 13-Jul-98

- AutoLisp code is correctly initialized if the drawing is changed outside of the ARX link, e.g., within an external application or through the AutoCAD user interface.
- Fixed some default geometric primitive code, where SEQEND was created to the current layer instead of the layer of the object.

ARX Link v0.6.12 25-Sep-98

- Free geometry can consists of multiple objects. Free primitive can return object handles in a string list, where handles are separated by spaces. Note that function dppExtapMappingGetHandle now returns a string that can have multiple handles in it.
- CMDECHO disabled during ‘Show Axis’ command.
- During the creation of dimensions DIMASO is set to 1 to make sure that only one geometric object is created.
- Color of the block DPP_BOX (used in BOX primitive) is set to BYLAYER.
- Added error checking into script file handling with better error reporting.

ARX Link v0.6.13 12-Jan-99

- GEO_COLOR, GEO_LEVEL and GEO_FREE_TYPE value strings can contain space characters.
- Loading of AutoLisp code is now more robust.
- BYLEVEL is accepted as the color spec for GEO_COLOR for compatibility reasons.
- Xref rotations handling fixed.
- Some miscellaneous bugs fixed.

ARX Link v0.6.14 12-Feb-99

- CAD startup fixed to wait until AutoCAD has properly started.

- Creation/deletion of geometry fixed to force AutoCAD to update its graphics display after each component creation/deletion. By default, AutoCAD rel. 14 updates its display only after a certain amount graphics entities are created/deleted.
- Model deletion fixed to delete also the associated drawing file even if AutoCAD is not running.

ARX Link v0.6.15 20-Apr-99

- The project specific startup script file startup.src is not used anymore. Instead, when ever a drawing is loaded, also an AutoLisp file startup.lsp is loaded.
- OSMODE is set off before executing AutoLisp free code.

AutoCAD 2000/ARX Interface

▲ The current version (v0.7.2) of AutoCAD 2000/ARX link is a port of AutoCAD R14/ARX link (v0.6.15) to AutoCAD 2000. The link currnetly works only in a Single Drawing Interface (SDI) mode. Design++ identification symbol used for the link is :arx2000.

Porting Design++ AutoCAD R14/ARX project to use AutoCAD 2000

1. Check references to symbol :arx in functions and rules files and add an entry for the :arx2000. For example,

```
(or (eq (dpp-cad-get-cad-system) :arx)
    (eq (dpp-cad-get-cad-system) :arx2000))
```

2. Project's CAD files are in directory <d++-project>\project\arx2000
3. Design++ arx2000 interface directory is <d++>\arx2000-interface
4. Convert drawing files into AutoCAD 2000 format.
5. Check/update custom menus.
6. Upgrade ARX application (see below).

Porting ARX application into AutoCAD 2000/ARX

▲ Please start by reading the book: AutoCAD200 Migration Guide for Applications

▲ When converting an old ARX project to R2000 there are three ways to start:

1. (Recommended) Create a new project with ObjectARX2000 wizard and add your code into it.
2. Read from the ARX2000 online help the document of how to create a project: ObjectARX Reference-> Additional Information-> Compiler Support-> Building and Debugging ObjectARX Programs for Windows NT-> Creating an ObjectARX Application using the MSVC++ 6.0 IDE
3. Edit an existing AutoCAD R14/ARX project.
   - Upgrade VC++ compiler to VC++6.sp3
   - Remove the preprocessor definition RADPACK as it is not used anymore.
   - Change ARX include and library paths to point to ARX2000 directories, like C:\Program Files\ObjectARX 2000\inc and C:\Program Files\ObjectARX 2000\lib
   - Change Design++ ARX link include and library paths to point to directory <d++>\arx2000-interface\win32-i86
   - In the Projects->Settings dialog's Link tab clear the 'Entry-point Symbol' in the Output category.

- Change the project's dppextap.def file to use the PRIVATE keyword. For example,

```
DESCRIPTION       'Design++ project extension app'
LIBRARY           dppextap
EXPORTS           acrxEntryPoint PRIVATE
                  _SetacrxPtp PRIVATE
                  acrxGetApiVersion PRIVATE
```

- The R14 ARX source code can be made compatible with R2000 by adding the following migration headers and defining symbol (like XYZ_R14) in your old project:

```
#ifndef XYZ_R14
#include <migrtion.h> //from R14 -> R2000 migration defs.
#include <dbapserv.h>
#endif
```

  Otherwise, it is recommended to start using the new ARX function names instead of the old ADS function names.
- ARX library names have been changed, check the book 'AutoCAD2000 Migration Guide for Applications' Library Changes (page 14) for the new names.

▲ After the project compiles, follow other R14 to R2000 porting tasks documented in Acad2000/ARX manuals.

Known problems

▲ AutoLisp error handling does not return usable error codes.

▲ Some model handling commands (open/delete/close) are not synchronous. As a result, Design++ does not always wait for the processing to complete in AutoCAD.

▲ Multiple Document Interface (MDI) mode support is unfinished. The support for the MDI mode will be released later. It is worth noting that the MDI will be running in the application context and will not support ADS_COMMAND function. Thus, it is recommended to start migrating away from using ADS_COMMAND in Free primitives and user functions.

MicroStation Interface

- A new dppGetVersion function added. This allows external applications to check that they were compiled within an environment compatible for the current MicroStation version.
- Element size checked not to exceed MicroStation limit of 65534.
- Primitive ruledSurface fixed to return an error, if the input is more than 100 points, instead of trying to show a partial geometry.
- Fixed the handling of profile points when there are more than 101 points as most MDL functions accept no more than 101 points.

C/API

- Following functions work now with filenames which have spaces in them

dppProjectGetProductStructureFileList
dppProjectGetExternalDataFileList
dppModelGetSecondaryModelName

- Function

dppSystemIsConsistencyRedeterminatorEnabled

fixed. It used to always return FALSE.

- New functions added:

dppPortAlloc
dppPortRealloc
dppPortFree
dppPortStrdup

- The idea is that if C/API's portability manager allocates/frees memory, then also the client should use the same dppPort functions to free/Allocate memory. By using custom dppPort manager functions this is always possible but with default functions there is problem on NT if using static runtime libraries as every static library gets its own copy of runtime memory allocation functions. Data type bool is replaced with dppBool as bool conflicts with C++'s built-in bool type.

▲ Following functions are renamed:

dppNodeLocalGetUserDataDestructor
renamed to
dppNodeLocalGetUserdataDestructor
dppNodeLocalGetUserData
renamed to
dppNodeLocalGetUserdata
dppNodeLocalSetUserData
renamed to
dppNodeLocalSetUserdata
dppNodeLocalGetNodeId
renamed to
dppNodeLocalGetId

▲ Problems in the definitions of following functions fixed.

dppComponentLocalSetModelName
dppComponentLocalSetName
dppComponentLocalSetPrettyname

▲ Added following into the C/API header files to speed up NT compilations.

```
#if _MSC_VER >= 1000
#pragma once
#endif /* _MSC_VER >= 1000 */
```

▲ Function dppAttributeDetermineValue fixed to properly format the determined attribute value. For example, it used to return attribute value (“ABC” “DEF” “GHI”) incorrectly as (ABC DEF GHI).

COM/API (changes since pre-release)

▲ Support for asynchronous messages added

▲ Systematic Memory handling error causing memory leaks in dppautomation.exe fixed

▲ Visual Basic example project and documentation had systematic error in handling memory in regards of the use of dppNew and dppDelete functions. Normally dppNew functions are only needed in order to create iterators.

▲ A bug in C/API message id handling fixed.

CAD Interface Manager

- Expanded CAD Integration Manager (CIM) provides framework for seamless bi-directional integration with various CAD systems.
- CIM is now loaded in <d++42>\misc\d++-startup.lisp if AutoCad/ARX link is selected as the CAD system.

CIM 1.10.1 17-Mar-98

- Generalized model handling, by removing assumptions about current drawing in operation. Argument list has changed for following functions.

dppOpenDrawingProc
dppNewDrawingProc
dppCloseDrawngProc
dppRevertDrawingProc
dppSaveDrawingProc
dppSaveAsDrawingProc

- Function

dppCadCadRegisterPrimitive

removed and introduced new functions listed below. Benefit from this is that now compiler can check for function prototype being correctly defined.

dppCadRegisterBoxPrimitiveProc
dppCadRegisterCylinderPrimitiveProc
dppCadRegisterExtrusionPrimitiveProc
dppCadRegisterFreePrimitiveProc
dppCadRegisterLinePrimitiveProc
dppCadRegisterLinearDimensionPrimitiveProc
dppCadRegisterPolyLinePrimitiveProc
dppCadRegisterRuledSurfacePrimitiveProc
dppCadRegisterSpherePrimitiveProc
dppCadRegisterSurfaceRotationPrimitiveProc
dppCadRegisterSweepPrimitiveProc
dppCadRegisterSymbolPrimitiveProc
dppCadRegisterSymbolWithAttributesPrimitiveProc
dppCadRegisterTextPrimitiveProc
dppCadRegisterXrefPrimitiveProc

- Changed the internal definition of following functions so that compiler can check for function prototype being correctly defined

dppCadRegisterModifyProc
dppCadRegisterOpenDrawingProc
dppCadRegisterNewDrawingProc
dppCadRegisterCloseDrawingProc
dppCadRegisterDeleteDrawingProc
dppCadRegisterRevertDrawingProc
dppCadRegisterSaveDrawingProc
dppCadRegisterSaveAsDrawingProc
dppCadRegisterShutdownProc
dppCadRegisterRestartProc
dppCadRegisterHighlightProc
dppCadRegisterHandleByNameProc
dppCadRegisterNameByHandleProc
dppCadRegisterMappingAddProc
dppCadRegisterMappingDeleteProc
dppCadRegisterDeleteComponentProc
dppCadRegisterUserFunctionProc
dppCadRegisterOperationProc
dppCadRegisterAsyncNotifyProc
dppCadRegisterEventProc

- Xref scale is now defined to be 3Dpoint instead of double.
- New Design++ Functions:

```
dpp-cad-start
dpp-cad-restart
dpp-cad-quit
```

- Fixed the handling of some special geo_rep attribute values:
  - :error, geometry creation failed and CIM primitive function returned NULL as the handle value.
  - :calculated, CAD probably will finish geometry creation later and the handle value :calculated is returned for now.
  - :done, CAD link was not on when value of geo_rep was calculated.

▲ New functions for user function handling:

bool dppCadUnRegisterUserProc(char * tag)
bool dppCadGetUserProc(char * tag)
dppChar dppCadRegisterUserFuctionProc(dppChar *tag, dppChar * args)

▲ Function

dppCadGetStartupReason

added. It can be used to query the reason for the CAD startup request issued by Design++.

▲ Primitives with names ending with ‘2’ are removed.

CIM 1.10.2 1998-04-02

▲ Design++ Functions

dpp-cad-send-request
dpp-cad-eval-user-message
dpp-cad-send-user-message

changed to use (read-from-string ans nil nil) for backward compatibility, so that they now do not return strings but the value inside the string.

CIM 1.10.3 7-Jul-1998

▲ Exported defparameter *dpp-geo-level* added. It will also be available with non-CIM -based CAD interfaces. This enables the creation of rules that use level/layer and work with any MS/ADS/ARX -based CAD links.

▲ Function

dppCadGetAutocalc

added for querying the on/off status of the autocalc feature.

CIM 1.11.0 24-Sep-98

▲ Data type bool replaced with dppBool due to changes in C/API.

▲ Some cleanup in the header files.

CIM 1.11.1 20-Oct-98

▲ Documentation was updated.

▲ Include directory was cleaned of obsolete files.

CIM 1.12.0 11-Jan-99

▲ String values of geo_level, geo_color, geo_style, and geo_free_type modified to allow space characters.

CIM 1.12.2 12-Jan-99

- Loading of the design rules for the Geometries system KB fixed for Design++ Runtime version.
- Drawing file deletion fixed to handle the deletion even if the CAD is not running.
- A missing default function binding added for

cad-interface-init-function

(defined with cim-define-cad-system).

- The geo_rep/handle value handling modified. See updated CIM documentation for details.
- User Function

dppCadGetUserProc

fixed to return a proper function pointer.

ODBC Link

- Design++ startup file <d++42>\misc\d++-startup.lisp modified to load both Design++ ODBC link and Dynamic Configuration Enhancements (patch-config-0). Thus, the projects no longer have to take care of the loading these in one of their function files. If the enhancements are not needed, or cause problems with existing applications, the load expressions can simple be commented out. Note that the ODBC link overrides the Oracle RDB link.
- Fixed an ODBC (Solid) Interface problem which prevented simultaneous DDE use. The problem was caused by the ODBC Interface having been compiled as a Console application which is not supported by C/API.

GUI Builder

- The new GUIB-V2 is not necessarily compatible with the GUIB version 1 that was shipped with Design++ 4.1. You can downgrade back to GUIB version 1 by extracting files from <d++>\bin\win32-i86\guib-v1.zip to directory <d++>\bin\win32-i86. Later, you can upgrade back to GUIB version 2 by extracting files from <d++>\bin\win32-i86\guib-v2.zip.
- GUI Builder V2 fixes and enhancements:
  - Two exit problems fixed. GUIB used to crash sometimes when the last window was closed. Also, exiting GUIB used to crash it.
  - Some error messages are now shown in the GUIB trace window instead of Galaxy debug window.
  - Restarting GUIB fixed
  - Some serious memory leaks fixed.
  - Some problems with graph updating fixed.
  - Some problems with handling of '.' and '\' in filenames fixed. Reopening the trace dialog works now after it it has been closed.
  - Trace output is now dumped to file, either HOME\guib_trace.log or .\guib_trace.log
  - A warning dialog box is now shown when there a stack over/under flow. It used to just abort.
  - Listboxes now send the Accept event when a cell value has been edited.
  - Comboboxes now allow the insertion of new values not already in the value list.
  - A bug, where accept handler sending an event that modifies other text fields crashes GUIB, fixed
  - A related bug, where the message "Undefined dialog reference :DLG:X" was displayed, fixed.
  - A bug, where empty column names resulted in one less listbox columns as requested, fixed.
  - Some focus problems are fixed.

- Labels of dialogs are now printed in bold.
- Style of labelitem works correctly .
- Some event handling bugs fixed.

▲ Things to consider when converting from Design++ 4.1 GUIB to Design++ 4.2:

- Changes in event handling might now cause loops like update -> select -> update -> ...
- Because values in listbox cells can now be changed directly, every listbox has to be checked. If event “OnDoubleClick” is used, add a style style { readonly } for the listbox. Or remove OndoubleClick, and let the user change values manually.
- Function concat works now like described in the documentation. The previous version allowed some other ways to use it also. So, it is recommended to check all concat functions.
- Dialog labels are now printed in bold and, thus, need more space. Now labelitem styles work correctly. For example,

```
labelitem XXX
style { BOLD }
```

  returns a bold labelitem and needs more space.
- If there are no CancelHook in a dialog, the CloseWindowButton in upper right corner does not work. Add a CancelHook if needed.

Happy with Design++