April 23, 2010

Design^{++} V8*i* (SELECTseries 1) Release Notes

Welcome to Bentley Design^{++} V8i (SELECTseries 1) release. This is the first follow-up release for Design^{++} V8i, which was released in October 2008 as part of Bentley's V8i software portfolio for infrastructure professionals.

Design^{++} V8i (SELECTseries 1) is supported on standard Windows PC platform running Windows XP, Windows Vista, Windows 7, or Windows Server 2003/2008. Windows 2000 is no longer supported. Design^{++} is a 32-bit application, but it does run on 64-bit Windows 7 and Vista either stand-alone or with a compatible 32-bit CAD application.

Highlights

New Integrated Development Environment

Design^{++} developer's interface is replaced with a new Integrated Development Environment (IDE) implemented with .NET using Windows Forms. IDE modernizes and integrates various old GUI components (UIP, DRE, Relation Browser), many of which were based on now obsolete technology. User Interface Server (UIS) is also reimplemented.

While IDE is the new default developer interface, the old UIP is still available as a backup. Design^{++} can be configured to start with either IDE (default) or UIP, which can also be started from IDE's File menu.

IDE can also be started from Command Interpreter with expression (start-ide).

Controlling Design^{++} from CAD

MicroStation and AutoCAD links are enhanced to allow Design^{++} to be started directly from an existing CAD session. Once started Design^{++} is able to connect to the active CAD model. This extension enables new types of applications where Design^{++} is controlled from a running CAD session instead of Design^{++} requiring its own CAD session.

This release includes a helper utility DppStart for starting Design^{++} and several VBA/COM, C#/.NET, and AutoLisp examples for controlling Design^{++} from MicroStation and AutoCAD.

Subpart Initialization Introduced

Design rule macro :CREATE is extended to allow subpart attributes to be initialized as subparts are created. Subpart attributes can be initialized either with fixed values or by providing the attributes local, instance-level design rules. Changes in attribute value initializations trigger the normal consistency maintaining change propagation.

Extended Assembly Attribute Referencing Introduced

A new reserved word ASSEMBLY (similar to SELF) is introduced for the attribute value referencing design rule macro :?. Using ASSEMBLY as the component reference triggers the referenced attribute to be searched from the current component's assembly hierarchy. This simplifies assembly referencing as the rule no longer needs to know where the attribute is defined in the assembly hierarchy.

Dot Notation Based Component Referencing Extended

Dot Notation based component referencing is extended to allow more comprehensive component search based on its index and class. This is compatible with the classic design rule component index referencing and related macros, such as :?<n>, :n, :any, and :last. The extended component search is specified with new brackets [] syntax. Below are some examples of the extended syntax and how they transform to the classic design rule syntax.

```
?pump[1]                    -> (:n pump 1)
?pump[1 devices]            -> (:n pump 1 devices)
?pump[:any]                 -> (:any pump)
?pump[:last]                -> (:last pump)
?pump[:all]                 -> (:instances pump)
```

The extended component referencing is also available in the typical component-attribute referencing:

```
?pump[1].geo_loc            -> (:? (:n pump 1) geo_loc)
?pump[:last].geo_loc        -> (:? (:last pump geo_loc)
```

Database Link Enhanced

The database link now supports direct database file specification in the connection string avoiding the need to specify global DSN name. The database link is also modified not to exit when changing the connection string. New utility functions, like dpp-db-with-connection-settings, have been added for easier database connection management.

Support for MicroStation V8i (SELECTseries 1) Added

Design^{++} MicroStation link now supports MicroStation V8i (SELECTseries 1).

Support for AutoCAD 2010 and 2011 Added

Design^{++} AutoCAD link now supports AutoCAD 2010 and 2011.

Supported CAD versions

Design^{++} V8i (SELECTseries 1) integrates with the following CAD versions.

MicroStation: V7 (aka J), V8 XM Edition, V8i, and V8i (SELECTseries 1)

AutoCAD (32-bit only): 2004, 2005, 2006, 2007, 2008, 2009, 2010, and 2011

Installation

- Installation of .NET 3.5SP1 and JRE 1.6.20 are now handled as installation prerequisites through a web downloader.

- Reconfigure now uses settings from the previous install (from DppInstallInfo.ini) as defaults, instead of default install settings.

- This release fixes a problem where leftover registry entries from previous uninstall of patched Design^{++} could fool install to think that Design^{++} is already installed.

Design^{++} Core

- **Introducing subpart initialization:** Design rule macro :CREATE is extended to allow subpart attributes to be initialized as subparts are created. Subpart attributes can be initialized either with fixed values or by providing the attributes local, instance-level design rules.

  :CREATE macro has a new optional argument attribute-value-initializations, which takes attribute value (or rule) initializations as a list of attribute value pairs.

  :CREATE (class-or-role &optional nr attribute-value-initializations)

  Furthermore, to simplify the specification of attribute value initializations, a new :INIT macro is introduced. It takes any number of attribute value pairs and returns a valid attribute-value-initializations expressions for :CREATE macro. Most importantly, it reasons which values are provided dynamically as expressions, local bindings, or global variables/constants and need to be evaluated. Note that attribute names are never evaluated. See the example below.

  A local rule initialization is defined as a list starting with the ':!' design rule specifier followed by a rule body. Note that any self references in the rule body refer to the subpart to which the rule is assigned to, not the assembly creating and initializing the subparts.

  Changes in attribute value (or rule) initializations trigger the normal consistency maintaining change propagation.

  Note that if initialization value conflicts with attribute’s valueclass the initialization is skipped.

  Below is an example of a table's substructure rule creating and initializing 1 table-top and 4 legs. It uses the :INIT macro to specify the attribute value initializations. Note how the legs' XY-dimensions are tied to table-top's XY-dimensions with local design rules.

```
(:! table substructure
    (let ((table-top-width 150))
      (:create
       square-top 1
       (:init
        ;; Local binding:
        geo_width table-top-width
        ;; Expression - evaluated unless quoted:
        geo_height (- table-top-width 50.0)
        ;; Local rule:
        geo_length (:! (/ ?self.geo_height 10.0))
        ;; List data - NOT evaluated:
        geo_loc (0.0 0.0 0.0)
        ;; String data - NOT evaluated:
        geo_color "green"))
      (:create
       leg 4
       (:init
        geo_width
        (:! (let ((table-top (first (:mounted-on self))))
              (* 0.05 ?table-top.geo_width)))
        geo_height (:! ?self.geo_width)
        geo_length 75.0
        geo_color "red"))))
```

Legs access the table-top through :mounted-on relation specified in table's relations rule, which ties the location and rotation of each leg to those of the table-top.

```
(:! table relations
    (let ((legs (:parts ?table leg)))
      (loop for leg in legs
          collect
            (list
             :mounted-on
             (list leg :vertex (+ (:index leg) 4))
             (list (:? square-top) :vertex (:index leg)
                   :inside 15 10 3)))))
```

Interestingly, using the new attribute value initialization and :mounted relation, the whole table can be specified with just 2 rules.

Extended Assembly Attribute Referencing Introduced: A new reserved word ASSEMBLY (similar to SELF) is introduced for the attribute value referencing design rule macro :?. Using ASSEMBLY as the component reference triggers the referenced attribute to be searched from the current component's assembly hierarchy. This simplifies assembly referencing as the rule no longer needs to know where the attribute is defined in the assembly hierarchy.

The attribute search starts from the current component's direct assembly and moves towards the model root until an assembly with the referenced attribute is found. Proper dependencies are created along the search path.

For example, let's look at the following product structure of a car.

```
car +-- body
    +-- chassis +-- axle.1 +-- wheel.1 +-- rim
                |          |           +-- tire
                |          +-- wheel.2 +-- rim
                |                      +-- tire
                +-- axle.2 +-- wheel.1 +-- rim
                           |           +-- tire
                           +-- wheel.2 +-- rim
                                       +-- tire
```

Using the extended assembly referencing, a tire can refer to the axle-weight attribute of its assembly with a simple expression

```
     (:?  ASSEMBLY axle-weight)
```

It does not need to know that the axle-weight is defined by its axle. Similarly, the tire can refer to the overall weight of the car with expression

```
     (:? ASSEMBLY total-weight)
```

Without the extension, the tire would have to use an expression, like

```
     (:? (:assembly self axle) axle-weight)
```

which requires that it knows where the axle-weight is defined. Furthermore, the expression would have to be modified if the axle-weight attribute was moved, for example, to a new axle-system assembly consisting of the 2 axles.

Dot Notation based component referencing is extended to support the new extended assembly referencing:

```
 ?assembly.axle-weight -> (:? ASSEMBLY axle-weight)
```

Note that this does not change the expansion of direct assembly reference:

```
 ?my.assembly.axle-weight -> (:? (:assembly my) axle-weight)
```

- **Multi-language handling:** Handling of multi-language Design^{++} prompts, attribute comments, and display names is modified to use :english as the backup language. That is, if a prompt, comment, or display name does not include a version for the requested language, then the :english version, if available, is returned. Empty string is returned only when the :english version is also missing.

  User can also specify the backup language, which defaults to :english. The backup language can also be set to :ignore or nil, which ignores the default backup language and allows testing the availability of a prompt, comment, or display name for the requested language.

  For this purpose, a new optional argument backup-language, which defaults to :english, is added to the following Design^{++} Lisp/API functions.

  DPP-GET-COMMENT
  DPP-COMPONENT-DISPLAY-NAME
  DPP-ATTRIBUTE-DISPLAY-NAME
  DPP-GET-DISPLAY-VALUE

Also, Lisp/API function DPP-SET-LANGUAGE is modified to allow the current language to be set to any language known by Design^{++} user interface. The list of the 14 known languages is returned by DPP-GET-AVAILABLE-LANGUAGES.
With these changes applications can provide a limited number of foreign language attribute comments while continuing to use the default :english versions for all the other non-translated prompts and comments.

- Parsing of relation specifications with references to deleted frames is fixed.

- Design rule after-method calling mechanism is fixed to allow proper dependencies to be created.

- Lisp/API function DPP-CLASS-DESCENDANT-INSTANCES is fixed to filter out duplicate component instances. It used to return duplicate instances for certain class hierarchies with multiple inheritance.

- Documentation strings for 330+ Design^{++} Lisp/API functions and design rule macros are updated.

- Reverting libraries and models are revisited.
    - Reverting a library is modified to revert also all of its depending libraries and models. This assures that any changes caused by the revert are properly reflected also in the depending libraries and models.
    - Reverting a model is modified to revert also all of its parent libraries and all of their depending libraries and models. This assures that any model changes stored in its parent libraries are properly reverted as well.
    - UIP revert notification is modified to notify UIP on all the reverted libraries and models.

- Design rule compilation fixed to update UIP (DRE) with the compilation status OK or Failed and with any compiler provided warnings and errors.

- Library version-unit (<LIBRARY-NAME>_INDEX), which is for internal system use only, is now filtered out from the list of classes accessible to the user through UIP.

- Updating an attribute's calculation order during redetermination is fixed to make sure that the attribute's referenced components still exist as some of them may have been deleted during redetermination.

- Lisp/API function DPP-DEPENDENCIES is fixed to create dependencies even when called outside the scope of a design rule.

- The special DEF-EXTERNAL-SERVER macro, which is provided for defining C/API clients as external servers for Design^{++}, is revisited to exit the external server in case its def-external-server is redefined. This is to guarantee that old servers not consistent with the new server definition are not left running.

- An internal utility checking an attribute facet value status (local, inherited, or both) is fixed to handle correctly multi-parent inheritance structures.

- The KB format has changed slightly. That is, the project index frame in the project index kb has been renamed to avoid naming conflicts with user library classes. The renaming happens automatically plus the project index kb is not even visible for the user. BUT, it means that Design^{++} V8i (SELECTseries 1) projects are NOT BACKWARD COMPATIBLE with Design^{++} V8i. If this becomes a problem, a Design^{++} V8i patch can be released to restore the compatibility.

Design Rule Language

- Dot Notation based component referencing is extended to allow more comprehensive component search based on its index and class. This is compatible with the classic design rule component index referencing and related macros, like :?<n>, :n, :any, and :last. The extended component search is specified with new brackets syntax []. Below are some examples of the extended syntax and how they transform to the classic design rule syntax.

```
?pump                        -> (:? pump)
?pump[]                      -> (:? pump)
?pump[1]                     -> (:n pump 1)
?pump[1 devices]             -> (:n pump 1 devices)
?pump[1 devices nil]         -> (:n pump 1 devices nil)
?pump[:any]                  -> (:any pump)
?pump[:any devices]          -> (:any pump devices)
?pump[:any devices nil]      -> (:any pump devices nil)
?pump[:last]                 -> (:last pump)
?pump[:last devices]         -> (:last pump devices)
?pump[:last devices nil]     -> (:last pump devices nil)
?pump[:all]                  -> (:instances pump)
```

The extended component referencing is also available in the typical component-attribute referencing:

```
?pump[1].geo_loc             -> (:? (:n pump 1) geo_loc)

?pump[:last].geo_loc         -> (:? (:last pump geo_loc)
```

User Interface Server (UIS)

- A new .NET based version
- The name of the UIS executable is changed to dpp-uis.exe

Developer's User Interface (UIP)

- Added synchronization to UIP startup and exit
- Fixed a problem in the model attachment dialog where removing an library attachment and then adding it back crashed UIP.

- Unhighlight component wasn't implemented. Now by default, the component highlighting is turned ON. Also, raise windows only when doing component highlight, not in unhighlight
- Fixed Querytool's GEO_COLOR value color entries to be in lowercase.
- Removed the Querytool valueclass menu entry "Clear Valueclass".
- In the Querytool's Lisp search the frame and the root fields where swapped.

Design Rule Editor (DRE)

- Design rule compilation is fixed to allow rules to be compiled only for classes in currently active libraries. Compiling rules for system library classes is not allowed.
- Handling of substructure rules containing local design rule initializations for subparts is fixed.
- Editing and compiling design rules for a class with the same name as the current project is fixed.
- Rule compilation is fixed to locate the rule even if the cursor is not inside the actual rule. The compilation used to fail with invalid rule error.
- Reactive menu is fixed to handle expressions containing the '&' character, such as (defun x (&optional y) y)
- Command 'File>Open Rule File' is fixed to handle files with 'TAB' characters. This used to crash occasionally. Note that this change has increased the file load time somewhat, especially for large files.
- Design Rule Editor's Dottify command ('Edit>Indent>Dottify the Rule') is modified not to extend the transformation to function calls, which often resulted in complicated, even if syntactically correct, expressions. The new simplified Dottify command performs only the following transformations.

```
(:? comp)                          -> ?comp
(:? self)                          -> ?self
(dpp-model-root)                   -> ?root
(:assembly self)                   -> ?assembly
(:instances comp)                  -> ?comp[:all]
(:n comp 1 devices nil)            -> ?comp[1 devices nil]
(:any comp devices nil)            -> ?comp[:any devices nil]
(:last comp devices nil)           -> ?comp[:last devices nil]
(:? comp attr)                     -> ?comp.attr
(:? self attr)                     -> ?self.attr
(:? (dpp-model-root) attr)         -> ?root.attr
(:? (:assembly self) attr)         -> ?assembly.attr
(:parts comp)                      -> ?comp.parts
```

```
(:all-parts comp)                          -> ?comp.all-parts
(:assembly comp)                           -> ?comp.assembly
(dpp-component-name (:? comp) :internal)-> ?comp.name
(dpp-component-name (:? comp) :pretty)  -> ?comp.prettyname
(:index comp)                              -> ?comp.index
(:class comp)                              -> ?comp.class
```

- Use DRE rule window size when using pretty print to reformat the rule.
- Added handling for back-quote syntax character '@'.

SAL – Standard Source Code Annotation

Most of the documented Design^{++} C/C++ API functions have now SAL (Microsoft's standard source code annotation language) annotations in them. The annotations provide a consistent way to annotate buffer parameters or return values for a function. When using Visual Studio 9.0 with /Analyze option to compile files, the compiler can use the annotations to check user's code for errors. For more detail on SAL see: http://msdn.microsoft.com/en-us/library/ms235402.aspx

For example, the SAL annotation for MicroStation link dppextap function dppExtapPrint

```
 void dppExtapPrint(_In_z_ _Printf_format_string_ const char *fmt, ...);
```

specifies that dppExtapPrint is similar to printf and so compiler can check that the function call parameters match the format string.

Following example specifies that the argument is normal C string and can't be NULL and that the return value should be checked.

```
 _Check_return_ char* dppExtapEval(_In_z_ const char *message);
```

Now if you have following code and compile it with VS 9.0 and use /analyze option

```
char * cmd = NULL;
char * ans = dppExtapEval(cmd);
dppExtapPrint("Answer was\n", ans);
```

you will get following kind of warnings:

```
dppextap.cpp(733) : warning C6271: Extra argument passed to
'dppExtapPrint': parameter '2' is not used by the format
string

dppextap.cpp(732) : warning C6387: 'argument 1' might be '0':
this doesnot adhere to the specification for the function
'dppExtapEval': Lines: 731, 732
```

C/API

- Function dppSystemGetProcessList is fixed to correctly test the end of the input.
- Function dppLibraryClose has a new argument closeDependingKbs.

- Function dppProjectFindObject is fixed to limit the object search for current libraries and model only. It used to extend the search for all currently loaded KBs, including system KBs.
- Added connection existence test into Eval and Send. A new error code dppCommNOCONNECTION is returned if Eval or Send is called without connection.
- Fixed dppCommShutdownClient to handle connection cleanup correctly. Now a connection can be restarted with dppCommInitializeClient.
- Removed functions dppSystemCopyFileToServe and dppSystemCopyFileFromServer.
- Renamed version number definitions in Design^{++} header files to use consistent terminology.
- Modified dppExtapStringToPoint to return dppBool.

COM/API

- Added the argument closeDependingKbs into function dppLibraryClose.

Starting Design^{++} From CAD

- MicroStation and AutoCAD links are extended to allow Design^{++} to be started directly from an existing CAD session. Once started, Design^{++} is able to connect to the active CAD model. This extension enables new types of applications where Design^{++} is controlled from a running CAD session instead of Design^{++} requiring its own CAD session.
- For MicroStation V8i (SELECTseries 1) .NET example documentation, see <d++>/msv8i-interface/Examples/DPPAddIn/ReadMe.txt
- For MicroStation V8i (SELECTseries 1) VBA example project, see <d++>/msv8i-interface/Examples/dppstart.mvba
- Following changes were mainly done to ease the starting of Design^{++} from CAD:
    - Following environment variables are introduced for customizing Design^{++} startup.
        - DPPSTARTUPPROJECT - Project to load at startup
        - DPPSTARTUPFUNCTION - Function to execute after project load
        - DPPSTARTUPFINISHEDFILE - File were socket port number is stored after the startup
    - Added experimental CAD drawing control disabling feature, which can used through variable DPPUSEACTIVEDOCUMENT.
    - When starting Design^{++}, if DPP is not set, use image path to construct it. Now Design^{++} can be started with default settings by clicking <d++>\bin\win32-i86\d++-dev.exe
    - If environment variable HOME is not set, set it with USERPROFILE value.
    - Fixed test of DPPWITHEMACS to test if it is 'yes' and that no value means 'no'

    - Make sure that the *language* and DPPLANGUAGE have a value.
    - Make sure that DPPTMP has a valid value.

CAD Integration Manager (CIM)

- Error Project can now be changed without restarting CAD. The following two new events are used to tell the CAD link that project has changed
    - dppBool dppCadRegisterOpenProjectProc(dppOpenProjectProc f)
    - dppBool dppCadRegisterCloseProjectProc(dppCloseDrawingProc f)
- Fixed spelling of typedef dppCloseDrawngProc.
- Fixed connection close cleanups to allow reconnecting.
- The prototype createViewProc has a new argument dppStringArray * properties.

MicroStation Link

- Both MicroStation V8i and V8i (SELECTseries 1) are supported with the same link version.
- Reviewed dppextap user functions for BoolInt and StatusInt usage as BoolInt value TRUE is not same as StatusInt value SUCCESS:

  Changed to return BoolInt instead of void:
    - dppExtapUnitSet
    - dppExtapUnitSetPerMs

  Changed to return BoolInt instead of StatusInt
    - dppDllExport BoolInt dppExtapUnitConvDppToMs(double* d)

  Fixed to return correct BoolInt values TRUE/FALSE instead of StatusInt values SUCCESS/ERROR
    - dppExtapUnitConvMsToDpp
    - dppExtapUnitConvPointDppToMs
    - dppExtapSetActiveTransformation
- Fixed documentation and header file to correctly state that it returns StatusInt dppExtapMappingGetElement
- Changed the information commands (Attributes, Axes, Magnifier, and Find) to use mdlLocate_allowLocked which allows selection of locked elements and considers all models. Move and Copy commands accept only unlocked element from current model.
- By default the example UI module DPPUI doesn't open the palette anymore, just the toolbar.

- With MicroStation V8 versions changed the printouts to go to MicroStation Message Center instead of the console. This way the console doesn't open by default.
- Removed unneeded defines STRCPY and STRCMPI from header file dppdefs.h
- Fixed some issues when no cell library is attached to the drawing.
- When creating a cell and no cell library is attached, try attaching default library.
- Modified the cell library search to check from specified symbols directory, <project>/<cad>/symbols directory, and then finally from MS_CELL path.
- Projects can be changed without restarting MicroStation.
- Ensure that CAD link system directory <d++>\<ms-interface>\win32-i86 is in MS_MDL
- Modified the AutoDrafter hidden line view to use the hiddenline removal files instead of the view presentation setting.
- Allow unloading and reloading of MSDPP
- PRIM_SPHERE changed to create an arc and rotate it as rendering works better this way than with the previous usage of rotating a sphere.
- Use the module location to find DPP value if DPP is not set.
- Ask projects path from Design^{++} if DPPPROJECTSPATH is not set
- Use MicroStation Alert dialogs for Design^{++} error dialogs
- Projects can be changed without restarting MicroStation.
- Version number defines in dppdefs.h have been renamed to match the naming convention used in Design^{++} header files.
- Use Alert dialogs for Design^{++} error dialogs.

AutoCAD Link

- AutoCAD link now supports AutoCAD 2010 and 2011.
- If AutoCAD 2010 or 2011 doesn't have VBA module installed and VBA use is tried, show the standard AutoCAD VBA module notification dialog.
- As 64-bit AutoCAD is not supported, make sure that in 64-bit Windows the link doesn't try to start 64-bit AutoCAD.
- Projects can be changed without restarting AutoCAD.
- Modified to close the drawing and to open temporary drawing when Design^{++} closes a model. Previously the old drawing was left open.
- Use the module location to find DPP value if DPP is not set.

- Ask projects path from Design^{++} if DPPPROJECTSPATH is not set
- Removed following obsolete functions
    - dpp-acad-enable-compatibility-mode
    - dpp-acad-disable-compatibility-mode
- Added AutoLisp functions dpp-disconnect and dpp-connect

Visio Link

- Design++ V8i (SELECTseries 1) integrates with the following Visio versions (32-bit only): 2000, 2000 SR1, 2002, 2003, 2007, and 2010.
- Projects can be changed without restarting Visio.
- Fixed dpp-visio-get-last-error-message to handle strings correctly.

AutoDrafter

- The keyword FIRST of AutoDrafter's component specification language fixed.

Database Link

- The database link now supports direct database file specification in the connection string avoiding the need to specify global DSN name. For example:

  ```
  "DRIVER=Microsoft Access Driver (*.mdb);DBQ=D:\\test.mdb;"
  ```

  ```
  "FILEDSN=D:\\test.dsn"
  ```

- The database link is also modified not to exit when changing the connection string.
- The database link user functions have been modified. The new functions names are listed below. For more information, see the documentation.
    - dpp-db-settings
    - dpp-db-connect
    - dpp-db-disconnect
    - dpp-db-connected-p
    - dpp-db-sql
- A new user function dpp-db-with-connection-settings is introduced for easier access of multiple databases. For example:

  ```
  (dpp-db-with-connection-settings
    ("DRIVER=Microsoft Access Driver (*.mdb);DBQ=D:\\test.mdb;")
    (dpp-db-sql "select field4 from dpp_test_table"))
  ```

  ```
  ➔ ((1.23456) (12.3456) (123.456) (1234.56) (12345.6))
  ```

User Interface Builder (GUIB)

- Most GUIB files are merged into main GUIB module file dppgint.exe:

  In Design^{++} V8i (SELECTseries 1) the following GUIB V1 files are delivered:

db_attr.dll	; Attribute broker definition
db_buck.dll	; Bucket broker definition
db_comp.dll	; Component broker definition
db_file.dll	; File broker definition
db_model.dll	; Model broker definition
dppcapi42_mt.dll	; CAPI DLL
dppgint.exe	; Main GUIB program file
dppgint.lib	; Export library for broker creation
dppgint.vr	; Dummy Galaxy resource file
guib.vr	; GUIB Galaxy resource file
guic.exe	; GUIB source file compiler
stdfld.dll	; C MODULE extension with input validation function
vgalaxy5.vr	; Galaxy 2.6 resource file
guibdefs.h	; Standard GUIB definitions.

  In the new GUIB V1.2 only following files are required:

  - dppgint.exe
  - vgalaxy5.vr

- GUI Builder Compiler's (GUIC) include file search path handling fixed. GUIC failed to find GUIB include files from certain directories.

- Lisp/API function DPP-GUI-POST-EVENT-DIALOG has been fixed to work asynchronously and not to receive an unwanted answer messages from GUIB.

Emacs Interface

- Emacs version has been updated to GNU Emacs 23.1.

JAVA/API

- Fixed threading problem where incoming answer messages could be lost.

Miscellaneous

- Changed to use Lisp online documentation from Franz.
- Various version number defines in C/C++ header files renamed to use same terminology, for example:
  - #define CAPI_MAJOR_VER 8
  - #define CAPI_MINOR_VER 11
  - #define CAPI_PATCH_VER 7
- Design^{++} startup environment variable DPPWITHDEVGUI is renamed to DPPDEVGUI. It specifies whether or not to start Developer GUI, and which version: IDE or UIP. DPPDEVGUI takes one of the values: [IDE, UIP, or disabled]. IDE is the default value.
- Secondary Model concept is declared obsolete. Related operations are removed from UIP and documentation; otherwise the underlying functionality is available for compatibility reasons.
- Systematic handling of obsolete functions is introduced. Now, obsolete functions generate both compile and runtime warnings.