GDL-Related Changes in ArchiCAD 19

May 22, 2015

Ver. 1.1

GRAPHISOFT®
A NEMETSCHEK COMPANY

GRAPHISOFT®

Visit the GRAPHISOFT website at *www.graphisoft.com* for local distributor and product availability information.

GDL-Related Changes in ArchiCAD 19

Trademarks

Contents

I. Goal of this document

This document summarizes all the changes in the GDL language, between version 18 and version 19, which affect existing elements. This document does not cover additional features; it focuses only on the changes that affect compatibility.

II. Why the changes in GDL interpretation?

We see big value in compatibility, and we are proud that even the newest ArchiCAD version can read elements which were written in GDL 1.0 more than 20 years ago. It's important to us that existing content should work with upcoming ArchiCAD versions as well, without any modifications.

On the other hand, as ArchiCAD's feature set has grown over the years, GDL coding requirements have also expanded. Thus, in some cases it is necessary to optimize the existing GDL code. In ArchiCAD v19, we introduced a new technology – called Predictive Background Processing – to further speed up model generation. This means that object scripts will be generated more often, typically in the background while the user is working on something else in the foreground. To make this background processing possible, GDL objects must follow certain rules, described in Chapter III of this document. Objects that don't follow these rules will run in ArchiCAD 19 the same way as they did in version 18. The only difference is that they cannot take advantage of the speed enhancement.

Another type of situation arises if an object's GDL code is written incorrectly, resulting in errors. "Incorrect" means that, although the code can perform the desired feature in some workflows, in other workflows it will produce a wrong result. Because we cannot ensure that users will use an object in a particular way, ArchiCAD should not support such incorrect GDL code. The object itself can still be loaded and used in the project, but any incorrect code will be ignored by ArchiCAD.

Because of this, we have changed the way "User Globals" work in GDL. The original goal of the User Globals feature was to provide a small data storage for an object, in which the calculated value could be stored and reused by related scripts. Because it was not explicitly forbidden by the GDL engine, this feature could be also used to transfer data from one object to another. This unsupported usage worked in some special workflows, but it did not work in most situations, so the result was unreliable. In version 19, as a consequence of the more intensive multiprocessing, the result will be even more unreliable. To avoid end user disappointment and to provide robust, predictable object behavior, we decided to prohibit this type of usage in v19. You can find more information about this in Chapter IV of this document.

There is another situation in which object behavior is not 100% predictable: it relates to the usage of requests and globals in the parameter script. It's a great feature in GDL that an object can reflect changes in its environment (e.g. a window can vary its symbolic representation as the scale value changes). But this feature can be achieved with several different programming methods. If used properly, this feature ensures that only the element's appearance (2D/3D scripts) follows the changes in the environment; the stored data of the element does not change, regardless of where it's viewed from. This method has been supported earlier and will be supported in the future as well, but modifying the object's parameters will not be supported starting from v20. We believe this change is necessary to ensure long-term reliable GDL usage in ArchiCAD. We also realize that existing GDL objects, written by 3rd party developers or customers, will now have to be modified, which takes time. Therefore we will introduce this change only starting with version 20. You can find more information about this in Chapter V.

For the future, we remain committed to our basic principle: to ensure compatibility to the maximum extent possible. To support the smooth introduction of changes, we are constructing a website dedicated to GDL, as part of the graphisoft.com domain, where we will inform the GDL community about such developments in advance.

III. Background processing

Some GDL statements will work unchanged, but are not compatible with background processing. If such statements are contained in an element's GDL 3D/master script, the element will convert in foreground only, causing delays in switching to a view.

List of non-compatible statements:

1. Text command
2. Any GDL add-on usage, except:
    a. Polygon Operations add-on
    b. Property Add-on
    c. Text I/O, Data I/O, XML add-ons, if the file is in the loaded library, and opened for reading only.
3. The following requests: CUSTOM_AUTO_LABEL, ZONE_COLUS_AREA, MATCHING_PROPERTIES, ASSOCEL_PROPERTIES, STYLE_INFO, MATCHING_PROPERTIES, TEXTBLOCK_INFO, FONTNAMES_LIST
4. Uses variable name macro
5. Uses variable name request
6. Calls a non-compatible macro

IV. User Globals

1. What has changed in v19 in the GDL environment?

GLOB_USER globals behavior:

- The Master GDL element can set these user globals; that value will be the default for all elements
- In a single interpretation (an element and all its macros), this global can be set to a different value, and read by all subsequent scripts (similarly to passing on values to the macro, but it can return them as well). The next interpretation (another element and its macros) will again use the default set in the Master GDL; there is no data exchange between the different interpretation instances.

2. What was the reason for this change?

Predictability of GDL elements:

- This change is a bug fix. Due to an earlier bug, the state of these globals was not reset to the default defined in the Master GDL, potentially leading to erroneous data exchange among library parts.
- This made the entire system unreliable, because if multiple library parts (e.g. A and B) were setting these values, and another library part (e.g. C) read them, the results were unpredictable, depending on the order in which the elements' scripts were executed:

- interpretation order A, B, C or B, C, A:
  - B is the last to set the global before C, so C reads the value set by B
- interpretation order B, A, C or A, C, B:
  - A is the last to set the global before C, so C reads the value set by A
- interpretation order C, A, B or C, B, A:
  - only the Master GDL set the global before C, so C reads the value set in the Master GDL

This bug has been fixed: all elements always read the value set by the Master GDL, unless their own scripts (caller element or called macro) modify that value.

3. How can you recognize if you are using such elements?

If you had a 3rd party library, in which setting parameters in a particular object affected many others, and it has now stopped working, chances are the library needs updating. Contact the developer of that library for an updated version.

4. How can you recognize if you are developing such elements?

If your library employs any kind of centralized control for object parameters (that is, setting a parameter of one object affects many others in the same library), search for GLOB_USER in all the scripts.

The use of user globals for data transfer between an element and its macros is obsolete, slow to interpret, and hard to debug. The best practice is:

- For data transfer from the caller element to the macro, simply add the parameters to the macro call
- For data transfer from the macro to the caller element, add the returned parameters to the macro call
- To centrally set a value for all parts of a library:
  - Use the Library Master object to set fixed values and value lists for all of your objects
  - Use the Library Globals set in Model View Options to have editable parameters affecting all Library parts using that Model View.

V. Requests and Globals in the Parameter Script

1. What has changed in v19 in the GDL environment?

Some requests and GDL globals usage have been limited in the following parameter-script type scripts:

- The elements' own parameter script
- The elements' master script, when it is running as a parameter script
- Master.gdl, Master.gsm, and Library Master files

If they are still used in parameter-script context:

- in v19 they will generate a warning in the GDL Editor
- in v20 and up they will be assigned a fixed default value

For a complete list of affected globals and requests, see Chapter VII.

2. What was the reason for this change?

The change was implemented to improve Teamwork stability and documentation consistency.

From now on, only direct user input, changes applied by the AC engine (e.g. migration), or add-on activity should modify objects' stored parameters.

If a GDL object autonomously tries to modify its own parameters (e.g. based on view-dependent data like Scale, or User-dependent data), this can cause:

- Issues in Teamwork
  - an element owned by someone else runs its parameter script, and tries to modify itself based on the current view, even though it should not
  - such requests are hard to filter and prone to errors, leading to instability and some cases even data loss in Teamwork.
- Unpredictable schedules
  - View-dependent globals/requests are not valid for all possible views (e.g. camera and sun position have no meaning in schedules), so these values are often not set
  - If an object uses such values, the results of the schedules will depend on the last-opened view
    - E.g. if you open a 1:50 view before opening the schedule, the result of the schedule would be different than if you had opened a 1:100 view.
- Inconsistencies in documentation
  - To optimize performance, the parameter script is only re-run when absolutely necessary.
    - To avoid slow-downs, a view change does not trigger a parameter script run for all elements

        - By default, a re-run of the parameter script should not be necessary; an element's own parameters should not change merely due to navigation between views

- This means that if a parameter depends on a view-specific value, it might not show values according to the current state of the element, but rather the values that were true the last time the parameter script was run
    - E.g. if an element's parameter is set to 0 or 1 depending on scale, said parameter will not contain the proper value according to the current scale, but rather according to the value that was valid the last time its settings dialog was opened.

3. How can you recognize if you are developing such elements?

- Warnings during script check for developers. Using the Check Script in the GDL editor:
    1. Open the object you want to check in the GDL editor
    2. Switch to the parameter script
    3. Press the *Check Script* button

    Warnings will notify you if there are any view- or project-dependent globals/requests in use in the parameter script (or in the master script when running as a parameter script)
- For more complex libraries, it is usually better to store the XML source of your libraries, and compile a new version when needed. If you compile your library with the AC19 LP_XMLConverter, you can set the checking parameters to check all your objects, and print the warnings and errors in your build-log.

VI. Frequently Asked Questions

1. I'm an architect, and I noticed that some of my objects stopped behaving correctly in V19. What should I do?

- If you notice any changes, please contact the developer from whom you bought the library part for an update.
- You can also check the elements yourself by opening the GDL editor and pushing the "Check Script" button for each object. If you have any warnings or errors, you should contact your library developer.

2. I'm a developer, and I develop scale-sensitive objects. What should I do?

- Scale sensitivity should be handled in the objects' 2D and 3D scripts, as these scripts affect the objects' appearance in the different views. Because an object can have different appearances in different views, the scale information shouldn't be stored in the objects' parameters.
- You can also do common calculations in the Master Script, if you use the result as a **variable.** But do not store view-dependent values in **parameters.**

3. I'm a developer, and I want central control for some parameters of my objects. What should I do?

- To centrally set a value for all parts of a library:
  - Use the Library Master object to set fixed values and value lists for all of your objects
  - Create Library Globals (any object under the *Library Global Settings* subtype) that can be set in the Model View Options to have editable parameters affecting all Library parts using that Model View.

4. How do I use the font list available on the user's computer?

- You can still use the corresponding request in the parameter script; this has not changed.
- The ArchiCAD 19 Library uses the "fontType" parameter name, so if you use "fontType" for your font parameter as well, the value list will be automatically handled by the ArchiCAD Library Master. But this requires that the ArchiCAD 19 Library always be loaded together with your object.
- If you want to have a central font list that works for your entire library, the best way to set the value list of your font type parameters is to make your own Library Master object (any object under the Library Master subtype, which works the same as a master GDL), use the fontnames_list request there, and add a value list to your common font type parameter. This way, the presence of the ArchiCAD 19 Library is not required.
- The Library Master object is available in AC18 and up.

5. Scheduling the location/coordinates of the Coordinate Dimension object was used to document setout points. Has a new method been added to deal with this?
    - Scheduling of these parameters was not reliable earlier, because the parameter script did not run when generating the schedules, so the results were inconsistent with the current state if someone moved the project origin at least once.
    - We have changed the Coordinate Dimension element so that it provides reliable coordinate information on floor plan and in 3D, but scheduling is not possible anymore.
6. I often used GLOB_SCALE to control paper and model dimension options for text (i.e. in zone stamps) along with GLOB_MODPAR_NAME. The last text size parameter to be updated would also recalculate the other. Maybe this wasn't the best way of working in Teamwork... but any ideas or alternatives?
    - hotspot2 command has a new working method: it can edit lengths in paper size, automatically calculating the current size. Please check the GDL Manual. This solves the positioning problems of texts or labels, but paper size and model size parameters can't be linked to each other.
    - Best practice is to have one (paper size or model size) as the core value
        - If it is paper size, then the model size should only be calculated as a variable on the fly during 2D/3D scripts
        - If it is model size, then the current paper size should not be shown at all (or if really necessary, only as a static string on the interface/in the 2D script)

VII. List of View- and Project-Dependent Globals and Requests

View-dependent GDL globals

In v19, using these globals in the parameter script will result in a warning in the GDL Editor. (The end user will not see the warning.)

In v20, if these globals are still used in the parameter script, their result will always be the value shown here.

GLOB_CONTEXT	2
GLOB_VIEW_TYPE	2
GLOB_SCALE	100.0
GLOB_DRAWING_BGD_PEN	19
GLOB_FRAME_NR	-1
GLOB_FIRST_FRAME	0
GLOB_LAST_FRAME	0
GLOB_EYEPOS_X	-5.0
GLOB_EYEPOS_Y	-5.0
GLOB_EYEPOS_Z	1.7
GLOB_TARGPOS_X	0.0
GLOB_TARGPOS_Y	0.0
GLOB_TARGPOS_Z	1.7

Project-dependent GDL globals

In v19, using these globals in the parameter script will result in a warning in the GDL Editor. (The end user will not see the warning.)

In v20, if these globals are still used in the parameter script, their result will always be the value shown here.

GLOB_NORTH_DIR	90.0
GLOB_PROJECT_LONGITUDE	0.0
GLOB_PROJECT_LATITUDE	0.0

GLOB_PROJECT_ALTITUDE	0.0
GLOB_PROJECT_DATE	[0, 0, 0, 0, 0, 0]
GLOB_WORLD_ORIGO_OFFSET_X	0.0
GLOB_WORLD_ORIGO_OFFSET_Y	0.0
GLOB_CUTPLANES_INFO	[1.0, 3.0, -0.1, -0.1]
GLOB_STRUCTURE_DISPLAY	0
LAYOUT_CURRENTREVISION_OPEN	false
GLOB_HSTORY_ELEV	0.0
GLOB_HSTORY_HEIGHT	3.1
GLOB_CSTORY_ELEV	0.0
GLOB_CSTORY_HEIGHT	3.1
GLOB_CH_STORY_DIST	0
GLOB_SUN_AZIMUTH	240.0
GLOB_SUN_ALTITUDE	35.0

Requests deprecated in the parameter script

In v19, they will result in a warning if used in the parameter script.

In v20, if these are still used in the parameter script, their result will always be 0 (or an empty string).

NAME_OF_PROGRAM

RGB_OF_MATERIAL

RGB_OF_PEN

PEN_OF_RGB

ID_OF_MAIN

FULL_ID_OF_PARENT

NAME_OF_PLAN

NAME_OF_MATERIAL

NAME_OF_FILL

NAME_OF_LINE_TYPE

NAME_OF_STYLE

STORY

CLEAN_INTERSECTIONS

HOME_STORY_OF_OPENING

INTERNAL_ID

CUSTOM_AUTO_LABEL

ZONE_COLUS_AREA

WINDOW_DOOR_SHOW_DIM

WINDOW_NAME_OF_LISTED

MATCHING_PROPERTIES

ASSOCLP_NAME

ANCESTRY_INFO

CONSTR_FILLS_DISPLAY

MATERIAL_INFO

BUILDING_MATERIAL_INFO

COMPONENT_VOLUME

COMPONENT_PROJECTED_AREA

TEXTBLOCK_INFO

WINDOW_SHOW_DIM

DOOR_SHOW_DIM

HOMEDB_INFO

FLOOR_PLAN_OPTION

CLASS_OF_FILL

VIEW_ROTANGLE

MODEL_TEXT_SIZE_UNIT

LAYOUT_TEXT_SIZE_UNIT

PROGRAM_INFO

STYLE_INFO

ANGULAR_DIMENSION

ANGULAR_LENGTH_DIMENSION

RADIAL_DIMENSION

LEVEL_DIMENSION

ELEVATION_DIMENSION

ZONE_RELATIONS_OF_OWNER

WINDOW_DOOR_ZONE_RELEV

WINDOW_DOOR_ZONE_RELEV_OF_OWNER

CALC_AREA_UNIT

CALC_ANGLE_UNIT

ASSOCEL_PROPERTIES

WORKING_ANGLE_UNIT

LAYOUT_LENGTH_UNIT

MODEL_LENGTH_UNIT

ASSOCLP_PARVALUE_WITH_DESCRIPTION