GOZ BRUSH SDK

Pixologic™
makers of ZBRUSH®

I ADDING GOZ SUPPORT FOR A NEW APPLICATION

1. SOME NOTES BEFORE STARTING:

• GoZBrush is extensible and can support up to 32 different 'GoZ-enabled' applications.

• GoZBrush uses a "public" folder to exchange files/data between ZBrush and external applications. Let's define **[PIXOLOGIC_PUBLIC]** as this public folder:

- "/Users/Shared/Pixologic" on MacOSX
- "C:\Users\Public\Pixologic" on Windows

• Each 'GoZ-enabled' application can be added or removed easily, as it simply consists of a subfolder for each application in "[PIXOLOGIC_PUBLIC]/GoZApps" folder.
Let's call it an "**Application subfolder**".

• Additionally, GoZBrush provides a folder "[PIXOLOGIC_PUBLIC]/GoZBrush" which contains common GoZ files, such as the GoZBrush config file, or the list of SubTools exchanged, etc…
Let's call it the "**GoZBrush subfolder**".

• And all the SubTools / meshes, and texture data go in the subfolder "[PIXOLOGIC_PUBLIC]/GoZProjects/Default".
Let's call it the "**GoZProjects subfolder**".

2. ORGANIZATION OF THE DOCUMENT:

• In the first part, we will describe briefly the GoZ SDK, which consists mainly of this documentation which explains how to create GoZ support for a new application, plus some useful source code to help you write GoZ support for the application of your choice.

• In a second part, we will detail the "GoZBrush subfolder" and its content.

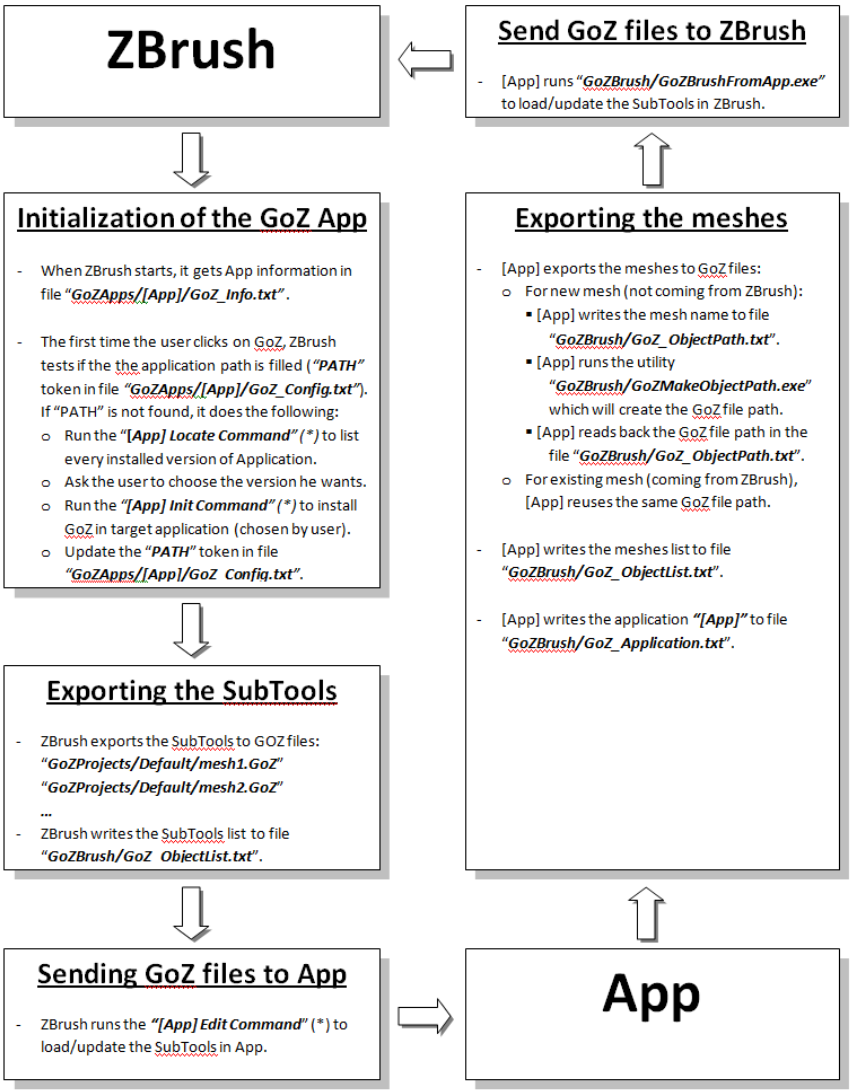
• In a third part, we will show the content of the "Application subfolder" and explain each file it must contain. You will find detailed information for each file you need to provide.

• A short appendix will define some terms/concepts we introduce for GoZ description purpose.

3. QUICK OVERVIEW OF GOZ WORKFLOW:

On the following pages you will find two diagrams:

- One shows the complete GoZ workflow between ZBrush and a GoZ Application,
- The other summarizes the workflow from a GoZ Application to ZBrush.



Run program
Read/write file
GoZ_ObjectPath.txt
GoZMakeObjectPath.exe
GoZ_ObjectPath.txt
App
GoZ_ObjectList.txt
GoZ_Application.txt
mesh1.GoZ
mesh2.GoZ
...
ZBrush
GoZBrushFromApp.exe

II THE GOZ SDK

The “GoZ SDK” is not really a true SDK, as it does not rely on a plugin system with libraries and DLLs.

Instead, it relies on utilities/tools provided in a separate subfolder for each application, plus 3D data files (“.GoZ” or any other extension supported by ZBrush). The GoZBrush system will call these utilities/tools, and read/write the 3D data files to communicate with the target application.

As there is no plugin system, the SDK content consists mainly of this documentation which explains in detail all the steps required to create a GoZ support for the application of your choice (let’s call it the ‘GoZ-enabled’ application).

This documentation also contains a brief description of the binary GoZ file format.

In addition, we provide some source code which may be useful when writing GoZ support for the application of your choice.

In this first part, we will:

- Summarize the major steps required to add GoZ support for your application.
- Give an overview of the internal GoZBrush behavior.
- Explain the binary GoZ file format.
- Describe the main functions provided in the source code.

1. THE MAIN STEPS/REQUIREMENTS TO ADD GOZ SUPPORT FOR A NEW APPLICATION

- Writing import/export code for GoZ binary format (which is very easy)
- Providing several utilities(.exe/.app or .bat/.sh) which are required by the GoZ process:
    - One utility to search for any installed target application: then in ZBrush the user will choose the path among all the returned target application paths found.
    - One utility to install any file needed in the target application path (for example, it can be scripts for target application, the GoZ icon, etc…)
    - One utility to read GoZ files (using the import code from point 1) and open the GoZ files in target application. This utility must launch the target application if and only if it is not running, and for each GoZ file it must either create a new mesh or update existing mesh depending on whether the GoZ file was already opened in the target application or not yet.

- For the target application, providing a command (such as a button, menu entry, etc…) which is responsible for exporting the selection to GoZ files (using the export code above) and then calling an existing utility(GoZBrushFromApp) which will launch/update ZBrush.

2. Overview of the overall GoZ process

In ZBrush, the user clicks on a GoZ button (GoZ, All or Visible):

- If ZBrush detects a new GoZ-enabled application support, it calls the utility which searches for all installed target applications, and ask the user to choose the application path he wants to use (an additional 'browse' button is proposed to easily handle custom installations). Then after the user has chosen a valid application path, ZBrush calls the utility responsible for installing the required files in the target application.
- ZBrush saves all the SubTools required (depending on which GoZ button the user clicked on: GoZ, All or Visible) to GoZ files, then it calls the utility responsible for opening the GoZ files in the target application.

In the target application, the user calls the GoZ command (via a GoZ button, or a GoZ menu, or whatever other way):

- The target GoZ command will save the selected meshes as GoZ files (being careful to reuse the same GoZ file name for meshes already "linked" to ZBrush – "linked" means either already sent to ZBrush previously or coming from ZBrush).
- Then the ZBrush utility (GoZBrushFromApp utility) will launch ZBrush if it is not running (and only in that case), and for each GoZ file saved in point 1, it will either update the existing matching SubTool (if the GoZ file is already linked to a SubTool in ZBrush), or create a new SubTool(this will "link" the GoZ file to the created SubTool).

3. The binary GoZ file format

The GoZ file format is a simple binary file format, which stores one SubTool (or mesh), with extra optional information like UVs, maps (color/normal/displacement), polygroups (ID per polygon), masks (masking alpha value per vertex), and/or polypainting (MRGB value per vertex).

It also stores some basic transformation flags such as axis swaps, texture flips, etc…

The binary format itself is pretty simple:

- First, a 32-bytes file header, starting with `'GoZb'` and followed by 28 bytes of text (for example "File generated by ZBrush 4.0")
- Then follow several data blocks, one for each type of data, plus one for the "end-of-file". Each data block starts with a 16-bytes block header:
  - [4 bytes] tag: it identifies the block type
  - [4 bytes] size: the block size, including the 16-bytes block header
  - [4 bytes] count: the data count (depending on the type: face count, vertex count, …)
  - [4 bytes] modifier: an extra floating value, which may be used or not, depending on the block type (for example, the displacement factor in a displacement map

block)

The other content of a data block depends on the block type.

While parsing, any unsupported block can be skipped easily, as you get its size in the header.

- The last block in a GoZ binary file is a “end-of-file” block. It simply consists on a 16-bytes sized block, which tag is `GoZ_TAG_END_OF_FILE`.

You will find detailed information on the different blocks directly in the source code provided.

4. The source code provided

The source code provided is straightforward:

4.1 “GoZ_Mesh.h”/“GoZ_Mesh.cpp”

This is a basic parser/writer for GoZ file format. Usage is simple:

- First, create a GoZ mesh: `GoZ_Mesh *mesh = new GoZ_Mesh;`
- Fills data (minimal data are required to write a valid GoZ file using GoZ_Mesh class):
  - `mesh->m_name` (cstring)
  - `mesh->m_vertexCount` (the number of vertices)
  - `mesh->m_vertices` (3 floats per vertex)
  - `mesh->m_faceType` (recommended is `GoZ_TAG_FACE4_LIST_FORMAT_1`)
  - `mesh->m_faceCount` (the number of faces)
  - `mesh->m_vertexIndices` (depending on the format used in ‘faceType’. If recommended format is used: 4 integers per quad/face, triangle represented by 4th index=-1)

- Then writes the mesh: `mesh->writeMesh("file.GoZ");`
- For reading a mesh:
  - Call `mesh->readMesh("file.GoZ");`
  - Then get public data from the GoZ_Mesh class.

4.2 "GoZ_Binary.h"

This is the complete description of the GoZ file format (with comments).

All the block types (or tags) are defined in this file, along with comments to describe the data associated to each tag, and how those data are organized in the block.

4.3 "GoZ_Utils.h"/"GoZ_Utils.cpp":

These files provides some functions that you may need to use when writing GoZ support for the application of your choice:

- Several low-level functions to read/write a GoZ binary file
- Several functions to read/write preferences (PFF file format)
- Other useful misc functions

4.4 "main.cpp"

A sample main function is provided just to show how to use the utility source files provided.

III THE "GOZBRUSH SUBFOLDER"

This folder contains several files relative to GoZBrush: some internal files used only by ZBrush itself, plus common files and utilities/applications that may be used by every 'GoZ-enabled' application:

- A file "GoZ_Config.txt" which contains general GoZ options and preferences.

- A file "GoZ_Help.txt" which contains the help displayed in ZBrush to the user, the first time he clicks on a GoZ button.

- A file "GoZ_ProjectPath.txt" which stores the path to the folder where all the ".GoZ" and texture files are written.

- A temporary file "GoZ_ObjectList.txt" which will store, at each GoZ exchange, the list of objects/SubTools/meshes to exchange between ZBrush and any 'GoZ-enabled' application.

- A temporary file "GoZ_Application.txt" which stores the active 'GoZ-enabled' application.

- A temporary file "GoZ_ObjectPath.txt" which is used by the 'GoZ-enabled' application with the utility "GoZMakeOjectPath" to create a "Unique object identifier", which is a unique link between an object/mesh from the 'GoZ-enabled' application and a ZBrush SubTool.

- A utility/application "GoZLocateApp.exe" / "GoZLocateApp.app" which searches for every versions of an application in the default application folder(s).

- A utility/application "GoZBrushFromApp.exe" / "GoZBrushFromApp.app" which, during a GoZ exchange, opens the objects exported from a 'GoZ-enabled' application into ZBrush.

- A utility/application "GoZMakeObjectPath.exe" / "GoZMakeObjectPath.app" which generates a "Unique object identifier", which is a unique link between an object/mesh from the 'GoZ-enabled' application and a ZBrush SubTool.

- A subfolder "Scripts" which contains internal ZScripts used by ZBrush for GoZ purposes.

1. GoZ_Config.txt

This file stores general GoZ settings, and uses the PFF file format (see Appendix).

Here are the main settings/preferences:

- `PATH`: The full path to ZBrush application
- `IMPORT_AS_SUBTOOL`: The GoZ preference “Import as SubTool”.
- `SHOW_HELP_WINDOW`: A boolean indicating whether or not showing the GoZ help note when the user click on any GoZ button.

Note:

The full path to ZBrush is not initialized during ZBrush installation, but is updated each time ZBrush is launched. So, GoZ requires ZBrush to be launched right after installation in order to work properly when the user want to edit in ZBrush an object coming from a ‘GoZ-enabled’ application.

2. GoZ_Help.txt

This file is for internal use only and must NOT be modified.

It contains the help text displayed to the user the first time he clicks on the GoZ button.

3. GoZ_ProjectPath.txt

This file stores the ‘Project path’, which is the path to the folder used to store the GoZ files exchanged between ZBrush and the ‘GoZ-enabled’ applications.
Each ‘GoZ-enabled’ application must read/write the GoZ files into this ‘Project path’, as ZBrush does.

The default ‘Project path’ is “[PIXOLOGIC_PUBLIC]/GoZProjects/Default”.

Note:

Changing the ‘Project path’ to another folder was not tested at all, so it may lead to some issue. Be aware that if you need to change it for internal use, you do so at your own risk.

4. GoZ_ObjectList.txt

This file is a temporary file used during an exchange between ZBrush and a ‘GoZ-enabled’ application, and contains the list of “objects” to exchange.

On the ZBrush side, an “object” is a SubTool, and on the ‘GoZ-enabled’ application side, it’s a mesh.

Some notes and requirements about this text file:

- It contains one line per object, and for each line it stores the full path to the GoZ file for that object, but without the extension. Let’s call it the “Unique object identifier”.

  For example, when send SubTools ‘sphere1’ and ‘cube1’ from ZBrush to another application:

  ○ ZBrush first saves the 2 SubTools as:
  “[PIXOLOGIC_PUBLIC]/GoZProjects/Default/Sphere1.GoZ”
  “[PIXOLOGIC_PUBLIC]/GoZProjects/Default/Cube1.GoZ”

  ○ then it writes the GoZ_ObjectList.txt as:
  [PIXOLOGIC_PUBLIC]/GoZProjects/Default/Sphere1
  [PIXOLOGIC_PUBLIC]/GoZProjects/Default/Cube1

- Exception for an empty list: the file must contain just an empty line (Carriage Return).

5. GoZ_Application.txt

This file stores the active ‘GoZ-enabled’ application, i.e. the application which is used / will be used to exchange objects with ZBrush.

The user can change the active ‘GoZ-enabled’ application:

- In ZBrush: by clicking on the “R” button at the right of the line containing all the GoZ buttons, it opens a dialog where the user can choose which ‘GoZ-enabled’ application to work with.

- In any ‘GoZ-enabled’ application: when the user clicks on GoZ, it first changes the active ‘GoZ-enabled’ application to this application, and then it opens/switches to ZBrush.

6. GoZ_ObjectPath.txt:

This is a temporary file used by the target ‘GoZ-enabled’ application to create a link between an object/mesh from the application, and a SubTool in ZBrush.

More details are provided in the “GoZMakeObjectPath.exe/ GoZMakeObjectPath.app”subsection.

7. GoZLocateApp.exe / GoZLocateApp.app

This is a common utility which can be used as a generic application search utility: it searches for all versions of an application in the default application folder(s) of the user's computer ("/Applications" on MacOSX, and both Win32 and X64 application folders on Windows).

It requires 2 parameters:

1. The 'GoZ-enabled' application identifier. In fact, the application identifier is the name of the "Application subfolder" (see more details in part 2 of this document).
   It is necessaryto write the list of the application paths found, as this list is written in the file "GoZ_AppPathList.txt" in the 'GoZ-enabled' "Application subfolder".

2. The short name of the application to search (for example "Cinema4D*.app"). The wildcards '*' and '?' are supported.

As a result, it creates the file "GoZ_AppPathList.txt" in the 'GoZ-enabled' "Application subfolder": for each path found it writes a line with the full path to the application found, and if no path was found, it just writes an empty line (Carriage Return) in the file.

8. GoZBrushFromApp.exe / GoZBrushFromApp.app

This utility MUST be called by every 'GoZ-enabled' application when the user clicks on the GoZ button in the 'GoZ-enabled' application to edit his mesh into ZBrush.

In some respects this utility is similar to the "Editing utility" (described in the part 2 of this document), except that it "brings" the object(s) to ZBrush instead of to the 'GoZ-enabled' application.

More precisely, this utility first verifies if ZBrush is running and if not, it launches a new ZBrush instance. Then, it opens the objects (listed in the file "GoZ_ObjectList.txt") in ZBrush.

9. GoZMakeObjectPath.exe / GoZMakeObjectPath.app

This utility/application may be used by any 'GoZ-enabled' application to create a "Unique object identifier" in order to "link" an object/mesh to a ZBrush SubTool.

How to use this utility:

- Writes the object or mesh short name into the text file "[PIXOLOGIC_PUBLIC]/GoZBrush/GoZ_ObjectPath.txt" (for example "sphere").
- Run this utility: it will create a "Unique object identifier" (for example "[PIXOLOG-

IC_PUBLIC]/GoZProjects/Default/sphere1").

- Then listen to the file modifications, or reads the file until its content was changed: the file will contain the "Unique object identifier".

Note:

As it must return a unique identifier, it automatically handles cases where an identifier matching the object name already exists; in this case, it ensures that the returned identifier is unique by adding/increasing an occurrence number at the end until it is unique.

What to do with the returned "Object unique identifier":

- This "Unique object identifier" is exactly is the same which is used in the file "GoZ_ObjectList.txt".
- Just appends ".GoZ" (or another extension if you exchange with ZBrush in another file format) at the end of this identifier, and you get directly the full path to the GoZ file to be exported from the 'GoZ-enabled' application (for that object).
- Be careful to link this identifier to the object on the 'GoZ-enabled' application side, so that you can directly retrieve the object given the identifier, and the reverse, retrieve the identifier linked to an object. On its side, ZBrush takes care to "link" an identifier to a SubTool.

10. Scripts subfolder

This subfolder is for internal use only and must NOT be modified.
It contains some ZScripts used internally by GoZBrush.

IV THE "APPLICATION SUBFOLDER"

The "Application subfolder" describes a 'GoZ-enabled' application, and must contain:

- The subfolder name itself, which is considered by ZBrush as the unique application ID.
  For this reason, it must contain NEITHER space NOR tabulation.

- A file "GoZ_Info.txt" which describes some information required by GoZBrush about the 'GoZ-enabled' application. Detailed information on this file will follow in asubsection.

- An application/utility which searches on the disk for all installed versions of the 'GoZ-enabled' application.
  Let's call it the "**Locate utility**".

- An application/utility which installs any file required for GoZ support in the target 'GoZ-enabled' application.
  Let's call it the "**Install utility**".

- An application/utility which opens or updates the SubTool(s) exported by ZBrush into the target 'GoZ-enabled' application.
  Let's call it the "**Editing utility**".

- Optional configurations file ("GoZ_Config.txt") which contains optional user preferences specific to the 'GoZ-enabled' application.

- ZBrush will store in this file the full path to the active version of the 'GoZ-enabled' application. It will create the file if it does not exist.

- A script or plugin for the target 'GoZ-enabled' application, which will be responsible for sending objects/meshes selected in the 'GoZ-enabled' application to ZBrush.
  Let's call it the "**Export plugin**".

- Whatever files required by the target 'GoZ-enabled' application to support GoZ.

1. GoZ_Info.txt

This describes the 'GoZ-enabled' application, and uses the PFF file format (see Annex).

Some preferences are required in this file:

- NAME: The full name of the application (it can contain spaces)

- EXTENSION: The extension of the exchange file format to use (the preferred format is ".GoZ")

- TAMPLATE: The export template file used by ZBrush to export a SubTool.
  For ".GoZ" extension, the preferred template file is "GoZ Complete Binary.GoZ"
  You will find all the template files in "[ZBrush install folder]/ZStartup/ExportTamplates/".

- EDIT_COMMAND: The application or command (.exe/.app or .bat/.sh file) to launch in order to edit the SubTool(s) from ZBrush into the external 'GoZ-enabled' application.

Some other preferences are optional but recommended:

- GOZ_VERSION: Sets the highest GoZ version supported. Default value is 1.

- LOCATE_COMMAND: the name of the "Locate utility".
  It can be either an application (.exe/.app), or a .bat/.sh batch.

- INIT_COMMAND: The application or command (.exe/.app or .bat/.sh file) to call in order to install some required files (for example target script files) in the target 'GoZ-enabled' application path.

Any 'GoZ-enabled' application may need to set some Import/Export options before importing/exporting files. By default, no import/export option is set.

These Import/Export options are in relation with the Import/Export preferences in ZBrush:

```
IMPORT_FLIP_X
IMPORT_FLIP_Y
IMPORT_FLIP_Z
EXPORT_FLIP_X
EXPORT_FLIP_Y
EXPORT_FLIP_Z
IMPORT_SWITCH_YZ
EXPORT_SWITCH_YZ
IMPORT_FLIP_NORMALS
EXPORT_FLIP_NORMALS

IMPORT_POLYGROUPS
```

```
IMPORT_MAT_AS_GROUPS
GROUP_NSIDED_POLYS

NORMAL_MAP_FLIP_X
NORMAL_MAP_FLIP_Y
NORMAL_MAP_FLIP_Z
NORMAL_MAP_FLIP_XY
NORMAL_MAP_FLIP_VERT

IMPORT_MFLIP_X
IMPORT_MFLIP_Y
IMPORT_MFLIP_Z
IMPORT_MSWITCH_YZ

EXPORT_MFLIP_X
EXPORT_MFLIP_Y
EXPORT_MFLIP_Z
EXPORT_MSWITCH_YZ
```

2. The "Locate utility"

This application/utility is responsible for searching all versions of the GoZ-enabled application in the user's computer. It must return a list of all the paths found in the text file "GoZ_AppPathList.txt": for each path found it must write a line with the full path to the application found, and if no path was found, it must write an empty line (Carriage Return) in the file.

ZBrush will call this "Locate utility":

- For each GoZ-enabled application: the first time the user clicks on the GoZ button.
- When a new GoZ-enabled application is added: the next time the user starts ZBrush and click on GoZ button.
- When the user goes to the GoZ preferences, and clicks on a GoZ-enabled application to change the version he wants to use.

Then it will propose that the user to chooses between all the paths in the list, or selects a specific location through a Browse button.

To avoid spending too much time searching for applications, especially those not installed on the user's computer, a good compromise is to automatically search only in the default application folder. For custom installs, the user will choose the version he wants by using the "Browse" button.

For your convenience, you can use a default "Locate utility", which searches for all

occurrence of a given application name in the default application folder(s). The application name may contain the wildcards '?' and/or '*'. This utility is "GoZLocateApp.exe" and is located in the "GoZBrush subfolder".

For example, the default Cinema4D "Locate utility" on Windows is a batch file which calls this utility:

..\..\GoZBrush\GoZLocateApp.exe Cinema4D "Cinema 4D*.exe"

3. The "Install utility"

This application/utility is responsible for installing any file required by a GoZ-enabled application in order to make GoZBrush work with this GoZ-enabled application.

Such files are specific to the GoZ-enabled application, and can be for scripts, icons or whatever file you need on the GoZ-enabled application side to make it work with GoZBrush.

In particular, this "Install utility" must copy the "Export plugin" in the 'GoZ-enabled' application.

Some notes and requirements about the "Install utility":

- First of all, it must get the full path to the 'active version' of the GoZ-enabled application where to install the files, in the file "GoZ_Config.txt". More information on the configuration file format can be found in the Appendix (PFF – Preferences File Format).

- Then, it must install the required files.

- And after the installation is done, it must create a file "GoZ_InstallLog.txt":
  - If installation went fine: an empty file.
  - If an error occurred during installation: a text message that will be displayed to the user to tell him that installation failed and if possible the reason why it failed, or what he can do to fix the problem.

- Consider that it is called with administrator privileges, as ZBrush takes care to ask the user to login as a user with administrator privileges when running this "Install utility".

4. The "Editing utility"

This application/utility is responsible for opening the SubTool(s) coming from ZBrush into the 'GoZ-enabled' application.

Some notes and requirements about the "Editing utility":

- It must ensure that the 'GoZ-enabled' application version used matches the 'active version' (the one selected by the user in ZBrush).
- It must take care to not launch several instances of a same 'GoZ-enabled' application but instead reuse the instance currently running.
- The list of SubTools to open/edit in the 'GoZ-enabled' application is given in the

file “GoZ_ObjectsList.txt” in the “GoZBrush subfolder” (more information on this file is provided in the first part of this document).

5. GoZ_Config.txt

This is the configuration file specific to the ‘GoZ-enabled’ application, and uses the PFF file format (see Appendix). It may contain any specific preference/option you may need to define.

ZBrush will add 2 preferences in this file:

- `PATH`: the full path to the active ‘GoZ-enabled’ application.
- `APPS_UNLOCKED`: only on MacOSX.
  To avoid problems with applications marked as unsafe after being downloaded, ZBrush will unmark such programs. It sets this preference after it has “unlocked” every application it has found in the “Application folder”.

6. The “Export plugin”

This is a script (or a plugin, or something similar) which is responsible for sending a selection of objects/meshes from the ‘GoZ-enabled’ application into ZBrush, by clicking on a GoZ button (or a menu entry, or any other widget of your choice) in the ‘GoZ-enabled’ application.
This plugin should be installed in the ‘GoZ-enabled’ application by the “Install utility”.

This plugin must perform the following:

1. For each selected object, first check if the object is already “linked” to a “Unique object identifier”. If not, then create a “Unique object identifier” for this object by using the “GoZMakeObjectPath” utility (see more information in first part of this document).

2. Once each object is linked to a “Unique object identifier”, create the file “GoZ_ObjectList.txt” in the “GoZBrush subfolder” and write the “Unique object identifier” for every selected objects (one identifier per line, see more information in first part).

3. And of course for each object, write the “.GoZ” file (or any other file format supported by ZBrush, depending on the file format exchange you choose in “GoZ_Info.txt”). Pay attention to save to the right file, which is the “Unique object identifier” + the extension (“.GoZ” or the extension stored in the “GoZ_Info.txt”).

4. Finally, simply call the utility “GoZBrushFromApp” which will open the objects in ZBrush.

V APPENDIX - PFF: PREFERENCES FILE FORMAT

Several GoZ files are used to describe/store preferences.

For example, each application must provide a GoZ_Info.txt file to describe the GoZ enabled application. Another example is GoZ_Config.txt in the GoZBrush folder, which stores some user GoZ preferences.

GoZ uses a common format for such files; let's name it PFF ("Preferences File Format").

This is very simple file format: it consists of a text file, where each line contains one preference definition with the following syntax:

```
PREFERENCE = value
```

Here are some example files:

1. GoZBrush/GoZ_Config.txt:

```
PATH                = "C:\progr...ZBrush 4\ZBrush.exe"
IMPORT_AS_SUBTOOL   = FALSE
SHOW_HELP_WINDOW    = TRUE
```

2. GoZApps/Cinema4D/GoZ_Info.txt:

```
NAME                = "Cinema 4D"
GOZ_VERSION         = 1
SHOW_HELP_WINDOW    = ".GoZ"
EXTENSION           = "GoZ Binary For Cinema4D.GoZ"
LOCATE_COMMAND      = "GoZLocateCinema4D.bat"
INIT_COMMAND        = "GoZInitCinema4D.exe"
EDIT_COMMAND        = "GoZBrushToCinema4D.exe"
```