

Walkinside™ SDK User's Guide

Version 4.1, October 2006

Notices

© 2002-2005 VRcontext s.a./n.v.. All rights reserved.

Walkinside™ SDK User's Guide

This manual, as well as the software described in it, is delivered under license and may be used or copied only in accordance with the terms of this license. The information contained in this manual is subject to change without notice. VRcontext s.a./n.v. assumes no responsibility or liability for any errors or inaccuracies that may appear in this manual or in the information delivered by the Walkinside software.

Walkinside, Walkinside Viewer, Walkinside logo, VRcontext and VRcontext logo are registered trademarks or trademarks of VRcontext s.a./n.v. in the United States and/or other countries.

OpenGL is trademark of SGI in the United States and/or other countries.

FontGen is trademark of Steve Williams.

Microsoft, Windows 98 SE, Windows Me, Windows NT, Windows 2000 and Windows XP are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

MicroStation/J, MicroStation 8, Triforma, PlantSpace and SmartSolids are either registered trademarks or trademarks of Bentley Systems Inc. in the United States and/or other countries.

PDS and SmartPlant Review are either registered trademark or trademark of Intergraph Corp. in the United States and/or other countries.

PDMS is either registered trademark or trademark of AVEVA in the United Kingdom and/or other countries.

Autocad is either registered trademark or trademark of Autodesk in the United States and/or other countries.

All other trademarks are the property of their respective owners.

VRcontext s.a./n.v., Av. Tedesco 7, B-1160 Brussels, Belgium.

Table of Contents

1 WALKINSIDE CONSOLE DOCUMENTATION

- 1.1 CONSOLE VARIABLE LISTING
- 1.2 CONSOLE COMMAND LISTING

2 WALKINSIDE DATABASE REFERENCE GUIDE

- 2.1 INTRODUCTION
- 2.2 INSTALLATION
- 2.3 FUNCTION REFERENCE

3 WALKINSIDE EXPORTER REFERENCE GUIDE

- 3.1 INTRODUCTION
- 3.2 INSTALLATION
- 3.3 FUNCTION REFERENCE
- 3.4 ADDITIONAL NOTES
 - 3.4.1 *Order of operations*
- 3.5 COORDINATE SYSTEM
- 3.6 GEOMETRY OPTIMIZATION CONTROL
 - 3.6.1 *General Seccion:*
 - 3.6.2 *OptimizeGeometry seccion:*

4 REGISTRATION

5 WALKINSIDE END USER LICENSE AGREEMENT

1 Walkinside Console Documentation

NOTES:

- Variables marked with "Internal use only" are either defined inside the .VR file or set by the viewer itself at runtime. Changing their values in *user.ini* will not change anything.
- All coordinates, distances and other measurements are in centimeters.

1.1 Console variable listing

NAME	TYPE	DESCRIPTION
avii_framerate	Int	Video recording framerate
avii_quality	Int	Video compression level
avii_windowHeight	Int	Vertical resolution of recorded videos
avii_windowWidth	Int	Horizontal resolution of recorded videos
avis_codec	String	Default video recording codec
gb_disableCloudPointsOpt	Bool	Internal use only
gb_fullScreen	Bool	Internal use only
gb_impostors	Bool	Internal use only
gb_loadBalancing	Bool	Internal use only
gb_showEffects	Bool	Internal use only
gb_showGUI	Bool	Internal use only
gb_stereo	Bool	Run in stereo ?
gb_stereoAutoFocus	Bool	Internal use only (Set with graphical user interface in walkinside)
gb_stereoFocusSmoove	Bool	Internal use only (Set with graphical user interface in walkinside)
gb_stereoSwapEyes	Bool	Internal use only (Set with graphical user interface in walkinside)
gb_thirdPerson	Bool	Internal use only (Set with graphical user interface in walkinside)
gb_useFrustumTrick	Bool	Internal use only
gb_videoTesselateAll	Bool	Use highly detailed tessellation for movie creation.
gb_wireFrame	Bool	Start viewer in wireframe?
gf_cloudPointsParam	Float	Internal usage
gf_cloudPointsSize	Float	Internal usage
gf_fov	Float	Vertical field of view in degrees
gf_rejModifier	Float	Internal use only (rejection distance modifier.)
gf_stereoEyeSep	Float	Internal use only (Set with graphical user interface in

		walkinside)
gf_stereoParallax	Float	Internal use only (Set with graphical user interface in walkinside)
gf_targetFrameRate	Float	Framerate to aim for when using r_loadBalancing
gf_thirdPersonDist	Float	Internal use only.
gf_zfar	Float	Distance to far clipping plane
gf_znear	Float	Distance to near clipping plane
gi_collisionCacheSize	Int	Size (in cells) of cache to use for collision data streaming
gi_screenDepth	Int	Internal use only
gi_screenFrequency	Int	Internal use only
gi_screenHeight	Int	Internal use only
gi_screenWidth	Int	Internal use only
gi_texDownscale	Int	Scaling factor to use for textures. This reduces a lot the video memory used for texture data.
glb_disableVAR	Bool	Disable use of GL_NV_vertex_array_range (i.e. video and AGP memory on GeForce)
glb_useAGP	Bool	Allow use of AGP memory?
glb_useDisplayLists	Bool	Use display lists?
glb_useVideo	Bool	Enable / disable video memory
glf_maxVidMemPercent	Float	Maximum fraction of available system memory to use as AGP memory
gli_requiredVidMem	Int	Internal use only – approximate amount of video memory needed to fit the whole model
gli_vidMemAmount	Int	Amount of video memory available in megabytes
gv_pointParams	Vector	
mb_showLogo	Bool	Internal use only
mb_showMap	Bool	Internal use only
mb_showOcean	Bool	Render ocean?
mb_useSpecular	Float	Set to 1.0 to enable specular lighting
mf_ambient	Float	Define the ambient color of elements (value for intensity range 0-1)
mf_mapX1	Float	Internal use only
mf_mapX2	Float	Internal use only
mf_mapZ1	Float	Internal use only
mf_mapZ2	Float	Internal use only
mf_oceanLevel	Float	Ocean level
mf_scale	Float	
mf_startPitch	Float	Internal use only (player view direction starting up

		walkinside)
mf_startRoll	Float	Internal use only (player view direction starting up walkinside)
mf_startYaw	Float	Internal use only (player view direction starting up walkinside)
mf_unitScale	Float	Internal use only – scale of units relative to a centimeter
mi_maxElementID	Int	
mi_numberVertices	Int	Should be changed to gl_requiredVidMem
ms_dbPluginName	String	Internal use only
ms_logoFile	String	Internal use only
ms_mapFile	String	Internal use only
ms_name	String	Internal use only
ms_skybox	String	Relative path to current skybox (e.g. “.textures/sky0”)
ms_unitName	String	Internal use only
mv_boundingBoxMax	Vector	Internal use only
mv_boundingBoxMin	Vector	Internal use only
mv_globalOrigin	Vector	Internal use only
mv_lightPos	Vector	Vector defining the global light position
mv_origin	Vector	internal use only
mv_startPos	Vector	internal use only
mv_unitx	Vector	internal use only
mv_unity	Vector	internal use only
mv_unitz	Vector	internal use only
ob_debug	Bool	internal use only
ob_disableDB	Bool	Completely disable database querying?
ob_filePath	String	internal use only
ob_genericCones	Bool	internal use only
ob_genericElbows	Bool	internal use only
ob_interactive	Bool	internal use only
ob_loadCollisions	Bool	If false no collision dat will be loaded in memory
ob_loadSolids	Bool	Load solids from VR files?
ob_loadStatic	Bool	Load static triangles from VR files?
ob_logFile	Bool	internal use only
ob_noMove	Bool	Disable/enable player movement.
ob_noTurn	Bool	Disable/enable player rotation.
ob_pingpong	Bool	Enable ping pong balls?
ob_runEvents	Bool	Internal use only
ob_sphereinternal	Bool	
ob_viewFrustum	Bool	
of_refRotationSpeed	Float	the speed at witch rotations take place for references.
pf_pitch	Float	Internal use only – player start pitch angle

pf_roll	Float	Internal use only – player start roll angle
pf_yaw	Float	Internal use only – player start yaw angle
pv_pos	Vector	Internal use only – player starting position
pv_posExtRead	Vector	
pv_posExtWrite	Vector	
sb_disableSound	Bool	Disable/enable audio?
sb_disableVoice	Bool	Disable/enable speech
wb_collisions	Bool	Internal use only
wb_enableJoystick	Bool	Internal use only
wb_gravity	Bool	Internal use only
wb_invertMouse	Bool	Invert the mouse's Y axis?
wb_thirdPersonCollisions	Bool	internal use only
wf_mouseSmooove	Float	Smoothing factor for mouse movement (higher = smoother)
wf_runSpeed	Float	Tony's running speed
wf_walkSpeed	Float	Tony's walking speed
ws_languagedoc	String	internal use only
ws_md3Mesh	String	internal use only
ws_md3Skin	String	internal use only
wv_charSize	Vector	internal use only

1.2 Console command listing

COMMAND	ARGUMENTS	DESCRIPTION
bots_toggleSpeed		Makes bots run instead of walk, or vice versa
camctrlreset		
cmdlist	Filename	Saves a list of all console commands to the specified text file
console		Displays/hides the console
cvarlist	Filename	Saves a list of all console variables to the specified text file
echo	Any string	Writes the string to the log file
exec	Filename	Reads and executes console commands from the specified text file
exit		Exits Walkinside
flood	Amount	Increases/decreases the ocean level by the specified amount
fly		Gravity off, collisions on
ghost		Gravity off, collisions off
jumpto	X, Y, Z	Moves Tony to the specified position
load		
modelTranslate		
playsound	Filename, Loop (optional)	Plays a sound, loops it if the second argument is 1. Filename is relative to sounds subdir.
print	String	Prints the value of the specified console variable to the log
quit		Exits Walkinside
restart		Restarts Walkinside and automatically reloads the current VR file.
savecvars	Filename	Creates a file like user.ini
screenshot		Takes a screenshot
setBool	Name, Value	Sets a boolean variable
setFloat	Name, Value	Sets a float variable
setInt	Name, Value	Sets an integer variable
setString	Name, Value	Sets a string variable
setVector	Name, Value	Sets a vector variable
shot		
spawn		
stopsound	Filename	Stops a looping sound that was started earlier
viewreset		Takes Tony back to his original position
walk		Gravity on, collisions on
waterlevel	Level	Sets the ocean level
select	Id, filename	Select the element with <id> in <filename>. The filename is the same as used in the references/levels walkinside dialog (no path).

2 Walkinside Database Reference Guide

2.1 Introduction

It is possible to attach database information to 3D elements while exporting Models, and view this information in Walkinside. The database handling is plugin based. This means the VRExporter will load dynamically a specific database library and all database information passed to the VRExporter will be forwarded to the database plugin. When viewing a model in Walkinside the corresponding Database plugin will be automatically loaded.

The database plugin is a DLL with the extension DB.

2.2 Installation

Add the “*wi_database.h*” and “*wi_database.def*” files included with this document to your C++ project. This header file defines the functions that need to be implemented.

The Walkinside Database plugin has to be located in the plugin directory, of Walkinside. E.g. C:\Program Files\VRContext\Walkinside\Plugins\pds.db

2.3 Function reference

This section lists all functions published by Walkinside Database, and 2 functions published by Walkinside Exporter concerning database information.

3.a Exporter Functions

```
int wieUseDataBase (const char *name);
```

If there is database information associated with the geometry elements, define witch database plug-in to use to handle the information. This is the file name of the plugin without extension. E.g. “pds” to use “pds.db”

```
int wieConnectDataBase (const char *rawBytes, int size);
```

This function is used to pass a memory block defining some settings to establish a connection. A copy of this memory block will be passed to the database plugin used. There are no restrictions for the type of information, as long the plugin can handle it.

```
int wieProcessNewElement(char *rawBytes, int size);
```

This function should be called for each “element” in the file. The definition of an element is up to you – usually each element will correspond to a “primitive” in the 3D modeling environment, such as a box, a sphere, a Bezier surface or a teapot.

Size: is the number of bytes passed to the exporter.

rawBytes: is a pointer to a memory block where the database information is stored.

A copy of this memory block will be passed to the database plugin used. There are no restrictions for the type of information, as long the plugin can handle it.

3.b Database Functions

```
const char * widb_GetPluginName();
```

Return the name of this plugin.

Functions for walkinside export support

```
bool widb_ExportFileName(const char *vrFileName);
```

The full pathname of the vr file being created. This is the first function called by the vrexporter.dll. Initializations should be done here. Return *false* if plugin can not be used.

```
bool widb_HasInfoFor(int id);
```

Return *true* if there exists database information for the element with identifier *ID*, else return *false*. The exporter will combine elements to one element if possible, to optimize for bigger frame rates in the viewer. Elements cannot be combined if there is valid database information present.

```
bool widb_PushFileName(const char *fn);
```

The file name that is currently being processed by the exporter. As mentioned in the “vrexporter_ref.doc” Walkinside can handle files that are referenced multiple times, without duplicating the data set. These files can be referenced hierarchically.

So typically the plugin should store these file names on to a stack.

Return *false* if there is no need of database information for this file.

Note: See *wieProcessNewRefFile* in the “vrexporter_ref.doc”

```
bool widb_PopFileName();
```

No more data is received for the file that was last pushed.

See *pushFileName(const char *fn)*

```
bool widb_ConSettings(void *dbInfo, int size);
```

Information needed to connect to a database (e.g.: database name, username ...).
The type of dbInfo is left to the implementation of the function connection settings.

Warning: This Function is not supported yet by *vrexporter.dll*. But will be in the near future. See also supporting SQL Query's.

```
bool widb_AddElement( int id,  
                     void *dataBuffer,  
                     int sizeBuffer,  
                     float position[3]);
```

This passes the data needed for an element, to retrieve the database information in the viewer.
(dataBuffer could be plain text, or record ID's of ODBC database, ..)

id : the element identifier.

dataBuffer : could be of any type. This is left to the implementer of this plugin.

(= copy of *rawBytes* parameter of *wieProcessNewElement*)

sizeBuffer : the size of memory allocated for dataBuffer.

(= *size* parameter of *wieProcessNewElement*)

position : the position of the element in the reference file.

See: *wieProcessNewElement(int size, char *rawBytes)*

```
const char * widb_Save();
```

This function is called to terminate the export process of database information. All data can be saved and destroyed from memory. If there is a file that should be packed in the VR Model, return its full pathname. Else, return a *NULL* pointer.

Functions for walkinside viewer support

Initialisation of the plugin

```
bool widb_init(    const char *tempPath,
                  const char *vrPath,
                  const char *vrFileName);
```

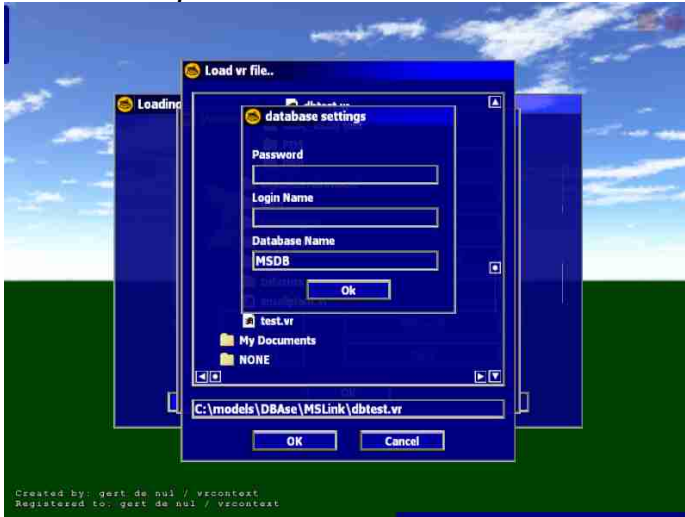
Initialize the database plugin to use with loaded model in Walkinside Return true if succeeded else return false.

tempPath : path where all files containing in the VR Model are extracted. If a Database file was stored in the VR Model. This will be the path where it should be located.

vrPath : path where the VR file is located

vrFileName : VR Model file name.

Get some user input after initialisation



```
int widb_GetNBInputParam();
```

Return the number of parameters needed to establish a connection. In this example there are 3 parameters.

```
const char *widb_GetNameInputParam(int id);
```

Return the name of the parameter corresponding to the "id". In this example "*widb_GetNameInputParam(1)*" -> "Login Name"

```
const char *widb_GetDefaultInputParam(int id);
```

Return the default value of the parameter corresponding to the “*id*”.
In this example “*widb_GetNameInputParam(2)*” -> “MSDB”

```
bool widb_SetInputParam(int id, const char *value);
```

The user input value of the parameter corresponding to the “*id*”, is passed. (It is possible that the user modify's the default values.)

Connecting to the database

```
bool widb_Connect();
```

Establish the connection to a Database. Return true if succeeded else return false.

```
bool widb_Disconnect();
```

Disconnect from the Database. Return *true* if succeeded else return *false*.

Searching information corresponding an element

```
bool widb_SearchElementInfo(int id);
```

Search information about an element with a specific identifier. Return *true* if there was information found else return *false*.

The reviewer of a model can use the question mark icon in the Walkinside Viewer to get information about an element. This action by the user will trigger this function.

```
unsigned int widb_InfoGetID();
```

Return the element *ID* for witch *searchElementInfo* was last called.

```
const char * widb_InfoGetFileName();
```

Return the reference filename the searched element belongs to. Walkinside viewer needs this information to reduce the data set to search for the element.

```
const char * widb_InfoGetData();
```

Return the information of the searched element as a character array. This result is shown in a Walkinside window, after a user selected an element with the question mark icon.

```
bool widb_InfoHasChanged()
```

Return *true* if the information has been changed since the last “*widb_InfoGetData()*” call, else return *false*.

An Database option in Walkinside is to check each time when a new frame is drawn if the selected information has been changed. This makes it possible for plugins to respond to realtime database information changes. (EG. Pressure of tubes, temperature of equipment, etc...)

Searching for elements corresponding to some information

```
bool widb_DoSearch()
```

This function should return TRUE if the search window of walkinside should be used. If this function return's FALSE the database module should display it's own GUI for the user to be able to search trough the database.

Note: If the database module only returns false to walkinside and does not display it's own search window, the walkinside user will feel as if the search function is disabled.

```
int widb_Search(const char *txt);
```

Return the number of elements found, matching the search parameter *txt*. The format of parameter *txt* is dependent of the plugin used (this could be an SQL query or just a plain txt). Instead of selecting an element to get information from, a user can do a search. In an edit box in Walkinside the user can type in some text. This input used as a parameter for this method.

Note: The results of this search are accessed with iterator type methods as explained next.

```
bool widb_ResultHasMore();
```

Return *true* if the result set from the last call to *search*, still contains results.

```
bool widb_ResultNext();
```

Go to the next result in the result set. Return *true* if succeeded else return *false*.

```
bool widb_ResultReset();
```

Go to the first element in the result set of the search.

```
unsigned int widb_ResultGetID();
```

As in *infoGetID*, get the identifier of the element currently selected by the iterators *resultNext*, *resultReset*.

```
const char * widb_ResultGetFileName();
```

As in *infoGetFileName*, return the reference filename of the element currently selected by the iterators *resultNext*, *resultReset*.

```
const char * widb_ResultGetData();
```

As in *infoGetData*, return the information of the element currently selected by the iterators *resultNext*, *resultReset*.

Using Walkinside SQL support dialogs.

The following Functions are for Database plugins who are supporting SQL Query's. This methods are still in a beta stage. Use dummy implementations for these functions.

```
bool widb_SupportQueryys();
```

Return *true* if this plugin implements Query support functions, else return *false*.

```
const char *** widb_GetQuerylist( int *numSqlData,
                                  int *numStatements);
```

Return all the SQL statement information in a 2 dimensional array. Walkinside uses these values to display a list of query's to the user.

numSqlData : Contains after the call, the number of data information defining the Query.

numStatements : Contains after the call, the number of Query statements currently present.



In this example **numSqlData* = 4, **numSqlStatements* = 1

Note: still beta.

```
char ** widb_CheckQueryfromInput( const char *inQuery, int
*num);
```

```
char ** widb_CheckQueryfromList(int inQuery, int *num);

char * widb_CompleteQuery(          const char **sqlParameters,
                                   int numParameters,
                                   const char *sqlStatement);

int widb_AddQuery(char *query,
                 char *data,
                 char *username,
                 char *queryName);

bool widb_RemoveQuery(int id);

bool widb_SetDefaultQuery(int id);

char * widb_GetQuery(int id);
```

3 Walkinside Exporter Reference Guide

3.1 Introduction

Although Walkinside only ships with an exporter for MicroStation, it is possible for developers to use Walkinside Exporter for their own applications. This allows 3D models to be viewed in Walkinside without the need to load them into MicroStation.

To use Walkinside Exporter with your application, you need to parse a 3D model and pass the data to the exporter. For MicroStation, this is done by an MDL application. For other file formats, you can either write a plugin for the host application or a standalone application that reads the 3D model files directly.

Walkinside Exporter is a DLL that exports a number of functions which allow you to describe a 3D model to it. Any programming language capable of loading and calling functions in a DLL is therefore capable of controlling Walkinside Exporter. The remainder of this document will assume that you are using C++.

3.2 Installation

To get started with Walkinside Exporter, you need to make sure that your application can find the Walkinside Exporter DLL. The DLL is named "*vr exporter.dll*" and is normally installed in your MicroStation directory (e.g. "*C:\Bentley\Program\MicroStation*"). It is not necessary to move the DLL into your system's search path, as you can load it dynamically from any location you like.

You will also need to add the "*wi_exporter.h*" and "*wi_exporter.cpp*" files included with this document to your C++ application.

3.3 Function reference

This section lists all functions published by Walkinside Exporter. In addition to the basic C++ types, the functions use the following type definitions (found in *wi_exporter.h*):

```
typedef double WIEVector2[ 2 ];
typedef double WIEVector3[ 3 ];
typedef double WIEMatrix4x3[ 4 ][ 3 ];
```

These represent a 2D vector (X, Y), a 3D vector (X, Y, Z) and a 4x3 matrix, respectively.

Also note that most functions return an integer. This integer will be one of the following values (defined in *wi_exporter.h*):

```
#define WIE_ERROR                -1
#define WIE_CANCELLED            0
#define WIE_SUCCESS              1
#define WIE_UNKNOWN_MATERIAL    2
```

A function will return *WIE_SUCCESS* if it succeeded, or *WIE_CANCELLED* if the exporter was prematurely terminated by the user. Functions may also return *WIE_ERROR* in the case of an internal error in Walkinside Exporter. In this event, more detailed information about the error may be found in the exporter's log file found in your Walkinside installation directory. Other return codes will be described below where appropriate.

```
bool wieLoadExporter(char *dll = "vrexporter.dll");
```

This function loads Walkinside Exporter from the DLL specified by the argument. The argument may contain a full pathname to the DLL. If the path is omitted, Windows will attempt to find the DLL in your system's search path. If the argument is not specified altogether, the default filename is used.

The function will return *true* if the DLL is loaded successfully, or *false* if the loading fails. This function must be called before any attempts to use Walkinside Exporter. If the function fails, you should not attempt to continue!

```
void wieUnloadExporter();
```

This function unloads the Walkinside Exporter DLL. It should be called before your application exits.

```
int wieSetSettings (char *drive, char *path, char
*filename, char *fileext, double unitscale, char
*unitname, char *modelname);
```

This function describes your 3D model to Walkinside Exporter. The first four arguments are the model's filename, split up into drive, path, name and extension. Walkinside Exporter will create a file which has the same name and path as the input model, but which has the extension ".VR".

The *unitscale* argument specifies a scale factor for the coordinate system of your model. Walkinside represents all coordinates in centimeters internally, but can still present information on screen in the units originally used in the 3D model. To this effect, you must specify a scale factor to convert internal units (centimeters) to external units. If your model is specified in meters, for example, the scale factor must be 0.01. Additionally, the *unitname* argument specifies the name of the external units (e.g. "m" for meters).

Finally, the *modelname* argument specifies a logical name for the 3D model. This does not have to be a filename, so you may use a more descriptive name if you please.

```
void wieStartup();
```

This function will start up Walkinside Exporter and create the progress dialog.

```
void wieInitialize();
```

This function marks the start of the data processing stage.

```
void wiePushMatrix(WIEMatrix4x3 matrix);
```

If your 3D model stores hierarchical transformations, you can have Walkinside Exporter resolve them automatically. This function takes a matrix and pushes it onto an internal matrix stack, after

multiplying it with the current top element on the stack. All geometric primitives you pass to the exporter are transformed using the top matrix on the stack. For readers familiar with OpenGL, `wiePushMatrix()` would be the equivalent of `glPushMatrix()` followed by `glMultMatrixd()`.

Note that the matrix is 4x3, not 4x4. The bottom row of the matrix is always assumed to be `[0 0 0 1]`. Walkinside again follows the OpenGL conventions, whereby matrices are column major and not row major. Hence, the “translation part” of the matrix is in the last column, not in the last row.

```
void wiePopMatrix();
```

This is the counterpart of `wiePushMatrix()` – it will pop the top matrix off the matrix stack.

```
void wieInitProgress(int numrefs);
```

Since a 3D model may consist of multiple files that reference each other, Walkinside Exporter needs to know up front how many files there will be – this enables it to correctly display its progress dialog. Call this function to specify the total number of files referenced by your 3D model. Note that the number specifies the number of unique files on disk – not the total number of file references in your model (i.e. if the same file is used twice, it only counts as one).

```
void wieProgressStep(double percentage);
```

The export process consists of two main stages: in the first stage, the host application parses the 3D model and hands the information to Walkinside Exporter, and in the second stage Walkinside Exporter processes the information and outputs a .VR file.

Walkinside Exporter can display a progress bar for its own processing, but it cannot do the same for the host application’s parsing progress. If you want Walkinside to display progress information for the host, call this function to specify your current file progress. The argument sets the percentage of the work that you have done on the current file, specified as a fraction (i.e. a number from 0 to 1).

```
int wieProcessNewRefFile(char *drive, char *path, char *filename, char *fileext, char *logicalname, char *modelname, WIEMatrix4x3 matrix, double scale);
```

As mentioned above, a 3D model may consist of several files. Walkinside Exporter can maintain the file structure of your original, so you can switch individual files on and off in Walkinside Viewer, or reference the same file multiple times without having to duplicate the data on disk.

This function will mark the start of a new reference file. It should be followed by the specification of the data in the reference file. The arguments are similar to those of `wieSetSettings()`. There is an additional *logicalname* argument which specifies a name by which to identify the file in Walkinside’s user interface screens. Furthermore, there is a *matrix* argument which allows you to transform the file.

The matrix argument should be specified in absolute world space. As a consequence, nesting reference files does not implicitly concatenate the current matrix with its parent one.

Addendum:

The matrix on the stack defined by subsequent calls to `wiePushMatrix()` and `wiePopMatrix()`, will be applied to the matrix argument of `wieProcessNewRefFile`.

So it is recommended, to specify a single push matrix that contains the axis orientation / space coordinate rotation to convert between a CAD model's system to Walkinside's one. And do calls to `wieProcessNewRefFile` in this scope of the stack

Example:

```
WiePushMatrix(mat_coordinate);
WieProcessNewRefFile("C:", "\\wieDemoll", "wieDemo", "dat", "", "", identity, 1)
WiePushMatrix(mat_A);
    // -> export geometry
wiePopMatrix();
WieProcessEndRefFile();
And not
WiePushMatrix(mat_coordinate);
WiePushMatrix(mat_A);
    WieProcessNewRefFile("C:", "\\wieDemoll", "wieDemo", "dat", "", "", identity, 1)
    // -> export geometry
wiePopMatrix();
WieProcessEndRefFile();
```

```
int wieProcessEndRefFile();
```

This function is the counterpart of `wieProcessNewRefFile()`, and marks the end of a reference file. Calls to `wieProcessNewRefFile()` and `wieProcessEndRefFile()` may be nested in order to create hierarchical file structures, provided that you specify all data in a given reference file before you process its nested references.

To reference a file a second time without duplicating its data, simply call `wieProcessNewRefFile()` and `wieProcessEndRefFile()` without specifying any data in between them. To process a reference file a second time, the *drive*, *path*, *filename*, *fileext* and *modelname* must match the ones of the reference that is being shared.

```
int wieProcessPlayer(WIEVector3 position, double pitch,
double yaw, double roll);
```

Walkinside supports full collision detection, but it needs to perform extra processing on the 3D model in order to enable this functionality. To tell Walkinside Exporter that you want to use collision detection with your model, call this function. It will tell Walkinside where your avatar will initially appear in the model. If you do not call this function, collision detection and all features that rely on it will not be available for your model. Note that you must call this function after `wieInitialize()` but before you specify any geometry data or reference files.

This function takes the 3D coordinates of the starting position as an array of *double* values. You can also specify the initial orientation of the avatar as a set of Euler angles (pitch, yaw and roll).

```
int wieSetColor(int red, int green, int blue, int level,
int color);
```

Walkinside assigns a color to each element in a file. In the absence of texture maps, these colors are used to display the elements on screen. Furthermore, each element can be placed on a certain "level" (or "layer").

This function sets the color and level to be used for all elements specified after the function is called. In addition to its red, green and blue components (each in the range of 0 to 255), the color must also be assigned a unique identification number.

You must also specify the level number. Walkinside Viewer allows you to switch individual levels on and off, so levels are also a convenient way to create logical groups of elements in your files.

```
int wieProcessMaterial(WIEMaterialInfo *info);
```

This function specifies the material properties of subsequent elements. The properties are stored in a *WIEMaterialInfo* structure which is defined as follows:

```
typedef struct __mapInfo
{
    char    texName[255];
    double  weight;
    double  angle;
    double  size_x;
    double  size_y;
    double  offset_x;
    double  offset_y;
    double  useFlip;
    double  scaleMode;
    double  mapMode;
    double  useTransparantBG;
} WIEMapInfo;

typedef struct __materialInfo
{
    struct
    {
        char    materialName[255];
        double  levelNR;
        double  colorNR;
        double  usePattern;
        double  useBump;
    } global;

    struct
    {
        double  color[3];
        double  specularColor[3];
        double  ambient;
```

```

    double diffuse;
    double specular;
    double finish;
    double transmit;
    double reflect;
    double refract;
    double useCast_shadows;
    double useRadiosity;
} prop;

WIEMapInfo bump;
WIEMapInfo pattern;
} WIEMaterialInfo;

```

The structure contains two *WIEMapInfo* structures which store information about the texture map and the bump map. The structure also contains translucency information.

```

int wieProcessUseMaterial(char *materialname, int level,
int color);

```

This function binds the material with the given name to all elements that are defined after this function is called. The material must have been previously specified with *wieProcessMaterial()*. *wieProcessUseMaterial()* overrides the data specified by *wieSetColor()*. Hence, it is only necessary to call *wieSetColor()* for those parts of your model for which you do not have full material definitions.

If *materialname* does not correspond to the name of a material previously defined with *wieProcessMaterial()*, the function will return *WIE_UNKNOWN_MATERIAL*. If this happens, the previously bound material will continue to be used. If you have a description of the missing material available, you can respond to this error by calling *wieProcessMaterial()* and then binding the material again.

```

int wieUseDataBase (const char *name);

```

If there is database information associated with the geometry elements, define which database plug-in to use to handle the information.

```

int wieProcessNewElement(int size, char *rawBytes);

```

Once you've gone through all the setup steps above, you can begin to specify the actual geometry data in your 3D model. This function should be called for each "element" in the file. The definition of an element is up to you – usually each element will correspond to a "primitive" in the 3D modeling environment, such as a box, a sphere, a Bezier surface or a teapot.

The *size*, *rawBytes* arguments is a buffer defining the Database information. This is only dependent on the database plug-in used. Pass 0, *NULL* to specify that there is no database information associated with the element.

```
int wieBeginSurface();
int wieEndSurface();
```

These functions are used to export planar surfaces with holes. All polygons exported using *wieProcessPolygon()* and *wieProcessPolygonWithVertexNormals()* between the calls *wieBeginSurface()* and *wieEndSurface()* are handled as contours of one surface. Walkinside Exporter will detect the inner and outer contours automatically.

```
int wieProcessPolygon(int vertices, WIEVector3
xyz[], WIEVector3 normal, WIEVector2 uv[]);
```

This function specifies a polygon to be added to the current element. The *vertices* argument specifies the number of vertices in the polygon. The *xyz* argument is an array of 3D vectors that define the vertex coordinates (its length must be *vertices*). The *normal* argument points to a single 3D vector which specifies the polygon's normal vector. Finally, *uv* points to an array of 2D vectors which specify the texture coordinates for the polygon. The texture coordinates are optional and should be *NULL* if not used.

```
int wieProcessPolygonWithVertexNormals(int vertices,
WIEVector3 xyz[], WIEVector3 normal[], WIEVector2 uv[]);
```

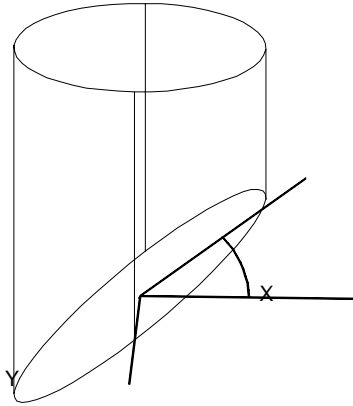
This function is equivalent to *wieProcessPolygon()*, except that the *normal* argument points to an array of normal vectors – one for each vertex. This allows for smooth shaded polygons, whereas *wieProcessPolygon()*, only allows for flat shading. The distinction was made because not all 3D applications can provide per-vertex normal vectors.

```
int wieProcessCone(WIEVector3 p1, double r1, WIEVector3
p2, double r2, WIEVector3 axis, double angle);
```

This function specifies a cone. A cone is specified by its two endpoints *p1* and *p2*. It can have different radii at both ends, which are specified by *r1* and *r2*. Furthermore, the top and bottom faces of the cone can be arbitrarily rotated. The rotation is specified by *axis* and *angle*. The angle is specified in radians.

```
int wieProcessSlopedCone(WIEVector3 p1, double r1,
WIEVector3 p2, double r2, WIEVector3 axis, double angle,
WIEVector2 baseAngles, WIEVector2 topAngles);
```

This function specifies a sloped cone. A sloped cone is specified like a normal cone, by its two endpoints *p1* and *p2*. It can have different radii at both ends, which are specified by *r1* and *r2*. Furthermore, the top and bottom faces of the cone can be arbitrarily rotated. The rotation is specified by *axis* and *angle*. The angle is specified in radians. The extra parameters *baseAngles* and *topAngles* defines the angles of the sloped top and base to the X axis and Y axis.



In this example only the base is sloped, and $baseAngles[0] = 4.Pi/9$, $baseAngles[1] = 0$.

```
int wieProcessElbow(WIEVector3 center, WIEVector3 axis,
double angle, WIEVector3 pstart, WIEVector3 paxis, double
pangle, double pradius);
```

This function specifies an elbow (i.e. a torus section). An elbow is created by sweeping a circular profile around a central axis (an operation which is commonly called “lathing” in 3D modeling software). The sweep axis is specified by *center* and *axis*, and the amount to sweep is determined by *angle*. The starting point of the sweep (i.e. the center of the circular profile) is specified by *pstart*, and its size is determined by *pradius*. Furthermore, the profile can be rotated, just like the top and bottom faces of a cone, by specifying *pxaxis* and *pangle*.

```
int wieProcessSweptArc(WIEVector3 arcenter, double
arcR1, double arcR2, double arcStart, double arcSweep,
WIEVector3 arcAxis, double arcAngle, WIEVector3
sweepOrigin, WIEVector3 sweepAxis, double sweepAngle);
```

This function specifies a “swept arc”. A swept arc is a more general case of an elbow, and is the most complex element directly supported by Walkinside. Where an elbow is created by sweeping a circle around an axis, a swept arc is created by sweeping a segment of an ellipse.

The *arcenter* argument corresponds to the *center* parameter of an elbow. The *sweepOrigin*, *sweepAxis* and *sweepAngle* correspond to *pstart*, *axis* and *angle*, respectively. *arcAxis* and *arcAngle* correspond to *pxaxis* and *pangle*.

What’s left are the arguments that specify the ellipse segment to be swept. The two radii (long and short) of the ellipse are defined by *arcR1* and *arcR2*. The start and end angles of the ellipse segment are defined by *arcStart* and *arcEnd*.

```
int wieStop();
```

This function is the counterpart of *wieInitialize()*, and marks the end of the data input stage. Once this is called, Walkinside Exporter will no longer accept new data and will begin to create a .VR file.

```
int wieShutDown();
```

This function is the counterpart of *wieStartup()*, and shuts down Walkinside Exporter. If the 3D model conversion was successful, Walkinside Viewer (if it is installed on your system) will automatically launch and open your model.

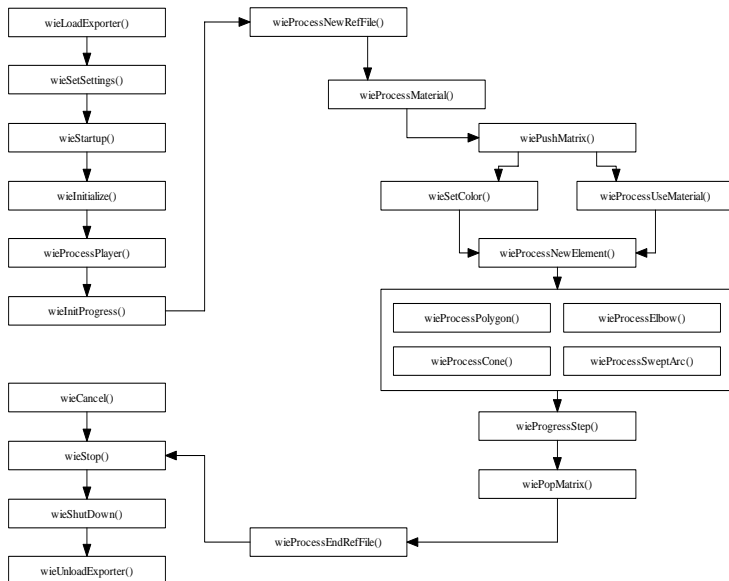
```
void wieCancel();
```

This function interrupts the export process and shuts down Walkinside Exporter. If you call *wieCancel()* or if the user presses the "Cancel" button in the Walkinside Exporter progress dialog, all the functions described above will return *WIE_CANCELLED* to indicate that you should stop passing data to Walkinside Exporter.

3.4 Additional notes

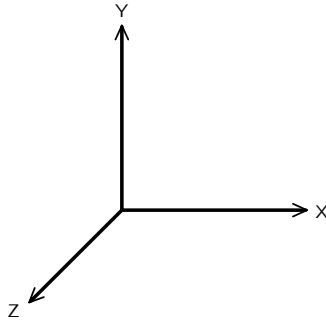
3.4.1 Order of operations

The following diagram illustrates the usual order of operations when working with Walkinside Exporter. Note that some functions are optional and some may be called multiple times. This is not indicated in the diagram but should be clear from the descriptions in chapter 3.



3.5 Coordinate system

The coordinate system used by Walkinside Exporter is as follows:



The X axis points to the right, the Y axis points up, and the Z axis points out of the screen. You should adjust your 3D models accordingly. This coordinate system matches the one used by OpenGL.

Also note that using extremely large coordinate values may lead to precision issues in Walkinside Viewer. If your 3D model is located very far from the global origin, we recommend that you use *wiePushMatrix()* to specify a transformation matrix that will place the model near the origin.

3.6 Geometry optimization control

When Walkinside Exporter is first launched the *vrexporter.ini* file will be created in the installation directory. This is the content of the file:

```
[General]
startViewer=TRUE
[OptimizeGeometry]
enabled=TRUE
colinearityAngle=5.00000
```

This value can be changed with any text editor application as Notepad to user specific ones.

3.6.1 General Section:

- “StartViewer” can be TRUE or FALSE. It controls if Walkinside Viewer is launched after conversion.

3.6.2 OptimizeGeometry section:

- “Enabled” can be TRUE or FALSE. If FALSE, all geometry optimizations are disabled. Useful if optimizations create corrupted models.
- “ColinearityAngle” specifies the threshold angle in degrees used to consider two lines colinear. The bigger it is, the less polygons the exported models contains, but might cause some corruption in special cases. Then, before disabling the geometry optimizations, it is recommended to fit this value to the bigger value possible that does not create corruption.

4 Registration

You need to be a registered user of Walkinside before you can start using it.

! You receive your license file via email. You need to copy it in the Walkinside directory, which is by default in **C:\Program Files\VRcontext\Walkinside**.

When you receive the license file, it can be named "ABC123.lic" or something similar – you need to rename the file to **vrcontext.lic** and copy it to the installation directory of Walkinside.

Do not hesitate to make a backup of your license file.

If you have downloaded Walkinside from our Web site, it will not run until you license it.
--

To launch the Walkinside Viewer DP or the Walkinside Viewer EP, you can either double click the Walkinside icon on your desktop or browse the "Start/Programs/Walkinside" menu and select the Walkinside Viewer DP or the Walkinside Viewer EP.

Your registered user name and company name will be shown on the screen during the review of the model. The "About" menu tells you the exact Walkinside version number, the registered User and Company name and the license expiration date (if any).

The "3D Model Info" menu tells you the Model title, the name of the registered user and company having made the conversion and the creation date.

5 Walkinside End User License Agreement

Please read this document carefully before using the software. If you do not agree to the terms of this license, do not use the software.

1. Grant of License. This License Agreement ("License") grants you the following rights with respect to the VRcontext ("VRCONTEXT") software products ("SOFTWARE"):

1.1 Walkinside Viewer DP software ("VIEWER DP")

(a) You may use one copy of the VIEWER DP ONLY on a single computer system.

(b) You may make one copy of the VIEWER DP onto electronic medium for backup purposes.

This copy must include the VRCONTEXT copyright notice and must include all of the object code for the VIEWER DP.

(c) You may transfer the VIEWER DP and all rights under this License to another party together with a copy of this License and all written materials accompanying the VIEWER DP, provided the other party agrees to accept the terms and conditions of this License and that it is removed from the computers from which it is transferred prior to installation by the other party.

1.2 Walkinside Viewer EP software ("VIEWER EP")

(a) You may use one copy of the VIEWER EP ONLY on a single computer system.

(b) You may make one copy of the VIEWER EP onto electronic medium for backup purposes.

This copy must include the VRCONTEXT copyright notice and must include all of the object code for the VIEWER EP.

(c) You may transfer the VIEWER EP and all rights under this License to another party together with a copy of this License and all written materials accompanying the VIEWER EP, provided the other party agrees to accept the terms and conditions of this License and that it is removed from the computers from which it is transferred prior to installation by the other party.

2. Restrictions. The SOFTWARE contains trade secrets and to protect them you may NOT decompile, reverse engineer, disassemble, or otherwise reduce or translate the SOFTWARE to a human perceivable form. You may not modify, adapt, translate, rent, lease, loan, resell for profit, distribute, network or create derivative works based upon the SOFTWARE or any part thereof. This is a restricted use license, not a sale of the SOFTWARE. Although you may distribute the SOFTWARE to any number of computer systems, each installation must retain all trademarks and copyright notices incorporated in the SOFTWARE.

3. Termination. This license is effective until terminated. This License will terminate immediately without notice from VRCONTEXT if you fail to comply with any of its provisions. Upon termination you must destroy the SOFTWARE and all copies thereof, including removal from all computer systems on which the SOFTWARE is installed, and you may terminate this License at any time by doing so.

4. Export Law Assurances. You agree that neither the SOFTWARE nor any direct product thereof will be transferred or re-exported from the U.S., directly or indirectly, into any country prohibited by the U.S. Export Administration Act and regulations thereunder or will be used for any purpose prohibited by the U.S. Export Administration Act and regulations thereunder.

5. Warranty Disclaimer, Limitation of Remedies and Damages. THE SOFTWARE AND ACCOMPANYING WRITTEN MATERIALS ARE LICENSED "AS IS." IN NO EVENT WILL VRCONTEXT, OR ITS OFFICERS, EMPLOYEES, OR AFFILIATES BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE OR ACCOMPANYING WRITTEN MATERIALS EVEN IF VRCONTEXT OR AN AUTHORIZED VRCONTEXT REPRESENTATIVE

HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU. VRCONTEXT SHALL HAVE NO LIABILITY TO YOU FOR ACTUAL DAMAGES FOR ANY CAUSE WHATSOEVER, REGARDLESS OF THE FORM OF THE ACTION.

6. General. If you are a U.S. Government end-user, this license of the SOFTWARE conveys only "RESTRICTED RIGHTS," and its use, disclosure and duplication are subject to Federal Acquisition Regulations, 52.227-7013. This License will be construed under the laws of the State of California, except for that body of law dealing with conflicts of law, if obtained in the U.S., or the laws of jurisdiction where obtained if obtained outside the U.S. If any provision of this License is held by a court of competent jurisdiction to be contrary to law, that provision will be enforced to the maximum extent permissible, and the remaining provisions of this license will remain in full force and effect.

VRcontext s.a./n.v., Brussels, Belgium.