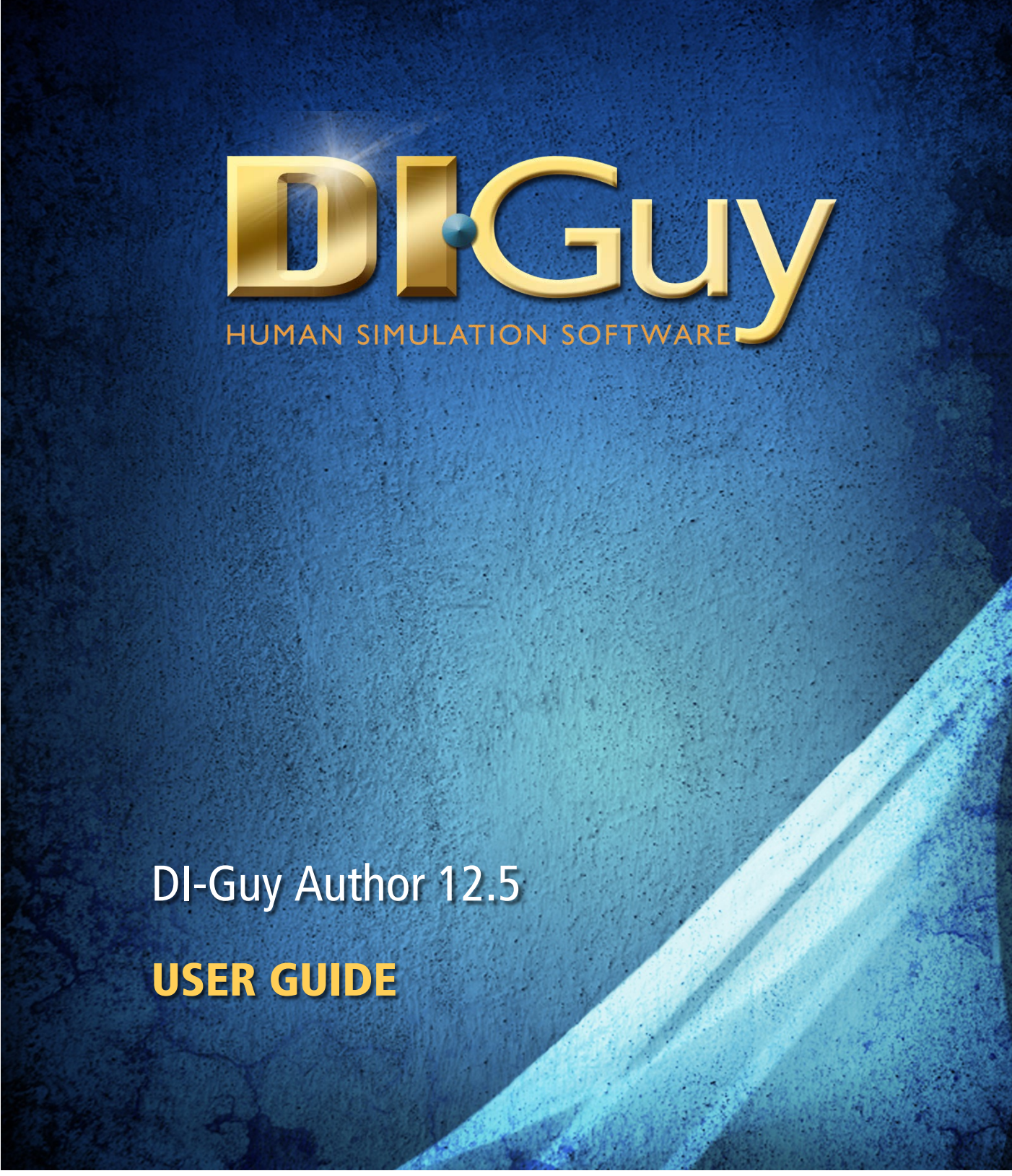
DI-Guy
HUMAN SIMULATION SOFTWARE
DI-Guy Author 12.5
USER GUIDE

DI-Guy Author User Guide

Version 12.5

Boston Dynamics

78 Fourth Avenue

Waltham, MA 02451 USA

For questions regarding *DI-Guy SDK*,

send email to diguy@diguy.com.

Part No. DI-Guy Author 12.5-User Guide

DI-Guy Software License Agreement

This is a legally binding agreement between you ("LICENSEE") and Boston Dynamics, with its principal place of business at 78 Fourth Avenue Waltham, MA USA ("Boston Dynamics"). By breaking the seal on the package containing the DI-Guy products (DI-Guy SDK, DI-Guy Scenario, DI-Guy AI and DI-Guy Motion Editor), and/or by installing and/or using the enclosed software, data, models, documentation, or related recorded information, regardless of its form or the method of the recording (the "SOFTWARE"), LICENSEE is agreeing to be bound by the terms and conditions of this agreement, including the software license and disclaimer of software warranty below. Please read this document carefully before opening the package and using the SOFTWARE. If LICENSEE does not agree with the terms and conditions of this agreement, LICENSEE should promptly return the unopened package and the SOFTWARE to the place where it was obtained, and LICENSEE will be given a full refund of any license fee that LICENSEE paid.

1. Grant of License: Subject to the terms and conditions of this Agreement, Boston Dynamics hereby grants to LICENSEE a non-transferable and non-exclusive right to use and execute the SOFTWARE on a single computer system at one time. LICENSEE agrees that it will not reverse engineer, decompile or disassemble any portion of the SOFTWARE, nor extract data of any kind from the SOFTWARE, including but not limited to motion data, 3D models, textures, texture movies, image sequences, terrain databases and the like. If LICENSEE disposes of any media or apparatus containing SOFTWARE, LICENSEE will ensure that it has completely erased or otherwise destroyed any SOFTWARE contained on such media or stored in such apparatus. LICENSEE may not distribute, lease, transfer, export, loan or otherwise convey the SOFTWARE or any portion thereof to anyone.

2. Copying Restrictions: In order to effect LICENSEE's license rights hereunder, it may install the SOFTWARE by copying it onto the hard disk drive or into the CPU memory of a computer system for use thereon, and may make full or partial copies of the SOFTWARE, but only as necessary for backup or archival purposes. LICENSEE agrees that (i) LICENSEE's use and possession of such copies shall be solely under the terms and conditions of this Agreement, and (ii) LICENSEE shall place the same proprietary and copyright notices and legends on all such copies as included by Boston Dynamics on the media containing the authorized copy of the SOFTWARE originally provided by Boston Dynamics.

3. Ownership: LICENSEE agrees and acknowledges that Boston Dynamics transfers no ownership interest in the SOFTWARE, in the intellectual property in SOFTWARE, nor in any SOFTWARE copy to LICENSEE under this Agreement or otherwise, and that Boston Dynamics and its licensors reserve all rights not expressly granted hereunder. After LICENSEE pays any applicable license fees, LICENSEE will own the media on which the SOFTWARE was originally provided hereunder and on which LICENSEE subsequently copies the SOFTWARE, but Boston Dynamics and its licensors shall retain ownership of all SOFTWARE and copies of the SOFTWARE or portions thereof embodied in or on any media.

4. Transfer Restrictions: LICENSEE may not sublicense, transfer, or assign this Agreement or any rights or obligations under this Agreement, in whole or in part.

5. Export Restrictions: LICENSEE agrees not to export or re-export, directly or indirectly, from the United States through any country or to any country of ultimate destination defined by the United States Government or any agency thereof as an "Excluded Territory". LICENSEE will not export, directly or indirectly, the Licensed Programs or Documentation to any country from which the United States Government or any agency thereof requires an export license or other governmental approval at the time of export without first obtaining such license or approval. The U.S. Government's list of "Excluded Territories" is subject to change without notice and is therefore not included herein.

6. Confidentiality: By accepting this license, you acknowledge that the SOFTWARE and accompanying materials, and any proprietary information contained in the media, are proprietary in nature and contain valuable confidential information developed or acquired at great expense. You agree not to disclose to others or utilize such trade secrets or proprietary information except as provide herein.

7. Enforcement of Terms: If LICENSEE fails to fulfill any of its obligations under this Agreement, Boston Dynamics and/or its licensors may pursue all available legal remedies to enforce this Agreement, and Boston Dynamics may, at any time after LICENSEE's default of this Agreement, terminate this Agreement and all licenses and rights granted under this Agreement. LICENSEE agrees that Boston Dynamics' licensors referenced in the SOFTWARE are third-party beneficiaries of this Agreement, and may enforce this Agreement as it relates to their intellectual property.

LICENSEE further agrees that, if Boston Dynamics terminates this Agreement for default, LICENSEE will, within ten (10) days after any such termination, deliver to Boston Dynamics or render unusable all SOFTWARE originally provided hereunder and any copies thereof embodied in any medium.

8. Governing Law: This Agreement shall be governed by and interpreted in accordance with the laws of the Commonwealth of Massachusetts, excluding its choice of law rules.

9. U.S. GOVERNMENT PURCHASES: If SOFTWARE is acquired for or on behalf of the U.S. Government, then:

a. It is recognized and agreed that the SOFTWARE: (i) was developed at private expense; (ii) was not required to be originated or developed under a Government contract; (iii) was not generated as a necessary part of performing a Government contract; and (iv) was not accomplished during and necessary for the performance of a Government contract.

b. It is recognized and agreed that SOFTWARE is commercial computer software, as that term is used in FAR Parts 12 and 27.4 (and any applicable agency supplements thereto), and DFARS Parts 211.70, 227.4 (OCT 1988), 227.71 (JUN 1995) and 227.72 (JUN 1995).

c. If this purchase is subject to FAR Part 27, and FAR 52.227-19 has been incorporated into the terms of this purchase, the Government's rights in the SOFTWARE are no greater than RESTRICTED RIGHTS as specified in FAR 52.227-19.

d. If this purchase is subject to DFARS Part 227 (OCT 1988), the Government's rights in the SOFTWARE are no greater than RESTRICTED RIGHTS as specified in DFARS 252.227-7013(c)(1)(i).

e. The Government's rights in the SOFTWARE are no greater than the rights expressly stated in the body of this Standard Licensing Agreement, if this purchase is subject to (i) FAR Part 12; (ii) FAR Part 27, and FAR 52.227-19 has not been incorporated into the terms of this purchase; (iii) DFARS Part 211.70; (iv) DFARS Parts 227.71 and 227.72 (JUN 1995); or (v) any other regulation or clause permitting the contractor to deliver commercial computer software under the contractor's standard commercial license.

f. Any related technical data shall be delivered to the Government with no more than limited rights.

10. DISCLAIMER OF SOFTWARE WARRANTY: BOSTON DYNAMICS PROVIDES THE SOFTWARE TO LICENSEE "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR WITH RESPECT TO INTELLECTUAL PROPERTY OWNERSHIP, NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY ANY BOSTON DYNAMICS EMPLOYEE, REPRESENTATIVE OR DISTRIBUTOR WILL CREATE A WARRANTY FOR THE SOFTWARE, AND LICENSEE MAY NOT RELY ON ANY SUCH INFORMATION OR ADVICE.

11. Limited Warranty on Media: Boston Dynamics warrants the media on which SOFTWARE is recorded and provided to LICENSEE under this Agreement to be free from defects in materials and workmanship under normal use for a period of ninety (90) days after the date of the original delivery of SOFTWARE to LICENSEE. Such warranty is solely for LICENSEE's benefit and LICENSEE has no authority to assign, pass through or transfer this warranty to any other person or entity. If LICENSEE returns any defective media to Boston Dynamics or an authorized Boston Dynamics representative during the warranty period with proof of purchase, Boston Dynamics will, at its sole option, either replace such defective media or refund the purchase price for such media. This warranty will not apply to any media that has been damaged by abuse, accident or misuse. The foregoing warranty sets forth Boston Dynamics' entire liability and LICENSEE's exclusive remedy for any defects in any media and is in lieu of, and Boston Dynamics disclaims, all other warranties, express, implied, or otherwise, including without limitation any warranty of merchantability or fitness for a particular purpose.

12. LIMITATION OF LIABILITY: IN NO EVENT SHALL BOSTON DYNAMICS OR ITS LICENSORS BE LIABLE TO LICENSEE FOR ANY SPECIAL, CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES OF ANY KIND (INCLUDING WITHOUT LIMITATION THE COST OF COVER, DAMAGES ARISING FROM LOSS OF DATA, USE, PROFITS OR GOODWILL, OR PROPERTY DAMAGE), WHETHER OR NOT BOSTON DYNAMICS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSS, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY ARISING OUT OF THIS AGREEMENT. THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING THE FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY. BOSTON DYNAMICS' LIABILITY ARISING OUT OF THIS SOFTWARE LICENSE AGREEMENT AND/OR LICENSEE'S USE OR POSSESSION OF THE SOFTWARE, INCLUDING WITHOUT LIMITATION ANY AND ALL CLAIMS COMBINED, WILL NOT EXCEED THE AMOUNT OF THE LICENSE FEE LICENSEE PAID FOR THE SOFTWARE PROVIDED UNDER THIS AGREEMENT.

13. Laws Governing Warranties and Liability: The law(s) of a jurisdiction may define the scope of warranty to be provided for products or the manner in which a supplier's liability may be limited, and such law(s) shall govern this Agreement only to the extent a party protected by such law(s) cannot waive the protection thereof by contract. In the U.S. and other countries, some states, territories or other principalities do not allow the limitation or exclusion of liability for incidental or consequential damages, or allow the exclusion of implied warranties, so the limitation and exclusion above may not apply to LICENSEE, and LICENSEE may have other rights that vary from state, territory or principality to state, territory or principality.

DI-Guy User Guide - Table of Contents

INTRODUCTION **8**

1. OVERVIEW **8**

2. DI-GUY AUTHOR IN ACTION **9**

WHAT THE END USER SEES 9

WHAT THE END USER DOESN'T SEE 10

3. IG INTEGRATION **13**

PROGRAMMING REFERENCE 13

Introduction

Welcome to *DI-Guy Author*™! DI-Guy Author puts the power of creating and editing DI-Guy scenarios directly into your application.

1. Overview

Because DI-Guy Author is part of the DI-Guy SDK, for use by programmers, and not an end product like DI-Guy Scenario, this document will be somewhat technical in nature.

DI-Guy Author is a combination of the DI-Guy Scenario application and the DI-Guy Graphics API. Like DI-Guy Scenario, it can create and edit scenarios, except the 3D interface is implemented in your IG (image generator), not the stand-alone DI-Guy Scenario application. Like the DI-Guy Graphics API, the 3D parts of DI-Guy Author are "3D library agnostic", meaning they can be implemented using almost any 3D library.

The needed 3D visual editing objects – e.g., waypoints, path splines, region meshes – are to be subclassed by you in a way similar to the way the DI-Guy Graphics API implements the 3D objects needed to render characters in an IG. C++ base classes are provided that provide all of the data needed to create and render editing objects. Your subclasses will take that information and do The Right Thing for your IG.

An additional class, again to be subclassed, provides an interface by which you give DI-Guy Author input information from you IG such as mouse movement and keystrokes, and by which DI-Guy Author can ask for needed 3D database information such as line to terrain intersections. With these data DI-Guy Author can internally modify the scenario under construction. DI-Guy Author will then tell you when and where to create the 3D visual objects to show the updated state of the scenario.

DI-Guy Author will do all of the calculations internally of how mouse input should affect waypoint positions and orientations and therefore path splines, which parts of the terrain should be covered with region polygons painted by the region paintbrush, and other authoring details.

The number of classes needed to be extended, and the amount of information needed to be supplied to DI-Guy Author, is quite small.

Scenario Editing in 2D

In addition to extensive editing possible in a 3D environment, DI-Guy Author provides a 2D user interface for fine-tuning edits made in 3D, and making edits to scenario objects that don't have a 3D component to them such as DI-Guy AI mind scripts.

2. DI-Guy Author in Action

Your application will herein be referred to as the Host IG, or sometimes just the IG. For some example purposes we'll use a fictitious product called the FarScene IG which has DI-Guy characters and DI-Guy Author capabilities.

What the End User Sees

This section describes how an end user interacts with the FarScene IG. The example provides only one possible implementation; most details can be tailored to fit your IG's needs.

Application Start

An end user of the FarScene IG starts the program normally. In "non-author" mode the IG runs just as it would if DI-Guy Author were not present: all visual geometry is rendered by the IG, and all inputs do whatever they would normally do. No DI-Guy Author elements are visible in either the 3D or 2D user interface (UI).

DI-Guy Author Activation

The FarScene IG user decides to edit a scenario using DI-Guy Author. He selects a menu item enabling Author, and a small window appears to show Author is active and allows the Input Mode of Author to be selected. The Input Mode starts out as "DI-Guy Editing Disabled", meaning all mouse and keyboard input will do whatever the IG would normally do with that input, such as move the camera, select entities, etc.

In addition to the Input Mode window, a new menu item called "Author" appears in the IG menu that contains menu items for doing various DI-Guy Author-specific actions such as loading and saving scenarios.

Opening a Scenario

The user selects the menu item "Author|Open Scenario", and finds and loads a DI-Guy scenario file that he previously worked on. The characters and other DI-Guy objects the user previously added to the scenario are loaded and show up in the 3D window. The user navigates the camera to the scenario location.

PeopleBlitzer

The user changes the DI-Guy Input Mode to PeopleBlitz. In PeopleBlitz mode, mouse clicks and drags will create new characters in the 3D window.

The user selects the character type “soldier_07” in the Input Mode window, selects a desired appearance for the soldier, and does a mouse click and drag in the IG’s 3D window. A new character appears in the scenario. Geometry representing the character’s path and waypoints also appears in the 3D window, showing where the character will move and providing handles that the user can use to edit the path.

The user now wants to see the characters in action, so he selects a menu item to display the Author Time Control window. The Time Control window contains a set of VCR-type buttons, including Play, Stop, Fast Forward, and Reset. The user clicks play, watches the characters in the scenario move around, and clicks Reset to put them back in their starting positions with time stopped.

The newly created character needs to make various turns and do various actions in the environment, so the user changes to Path Edit input mode, adds and modifies new path waypoints by clicking and dragging waypoint visuals in the 3D window, then changes to Action Edit input mode, and edits and creates new action beads on the path that tell the character to do different things on different parts of the path. The user plays the scenario again to see how his edits look.

CrowdBlitz

The user now wants a small crowd of people in his scenario, so he changes to Crowd Blitz mode so that he can paint in a large number of characters with a few mouse movements.

The user selects a crowd profile in the Input Mode window that will determine the types and appearances of characters in the crowd. In the 3D window the mouse cursor has changed to a translucent sphere, the Author 3D “paintbrush”, used for painting areas in the 3D window. The user clicks and drags the paintbrush in the 3D window to lay out the region in which the crowd should appear, and on mouse button release a set of new characters appears in the painted region.

The user plays the scenario again, resets it, and makes edits to the region just painted using the Crowd Edit input mode. Satisfied with the results the user selects the “Author|Save Scenario” menu item.

Closing Author

The user exits Author mode, at which point all Author windows and editing visuals in the 3D window disappear. The IG UI returns to exactly what it looked like before Author was enabled. The characters remain, and move through the IG terrain to act out the edited scenario.

What the End User Doesn’t See

This section describes what happens “behind the scenes” in the above example.

Application Start

As stated, the user of the FarScene IG starts it normally. Beyond the fact that DI-Guy Author has been integrated into the IG by a programmer in the same way that the DI-Guy Graphics API has been, nothing is different.

DI-Guy Author Activation

When the user activates DI-Guy Author via a DI-Guy SDK function call, DI-Guy initializes some internal state to support Author editing. The IG modifies a native UI control, probably a checkbox, toggle button, or menu item, to show that Author is active. Until the user changes DI-Guy's Input Mode, however, the IG still acts as it would normally.

3D vs. 2D User Interface

There are two distinct user interface components required by DI-Guy Author: 3D "editing visuals" in the IG 3D window such as waypoints, regions, arrows, etc.; and "controls" in the IG 2D user interface such as buttons, checkboxes, text edit boxes, dialog boxes, etc. Integrating these two components into the IG require very different steps.

Native 2D Controls vs. DI-Guy Author 2D Controls

Further notes about terminology for some upcoming discussion: as mentioned above the term "controls" refers to 2D UI elements such as buttons, menu items, checkboxes, etc. "Native controls" refer to controls in the IG 2D user interface that are implemented natively in the IG. While they may query state from DI-Guy and change state appropriately, DI-Guy Author itself does not render them or change their state. "Author controls" refer to 2D controls created by and managed by DI-Guy Author. Author controls will show up in subwindows and dialog boxes implemented and controlled by DI-Guy Author.

Note that many DI-Guy operations can be accomplished by calling DI-Guy SDK functions based on native UI events. For example, the Input Mode, Playback Mode, and many other things can be set using DI-Guy SDK function calls. This means that if only a small set of DI-Guy operations need to be available these could be implemented using native controls. If more advanced editing operations are required, the full DI-Guy Author UI can be enabled.

DI-Guy Author UI

While DI-Guy Author can be used with strictly native controls, there is a lot more user interface available if the Author UI is enabled. A majority of the 2D UI in the DI-Guy Scenario application is available in the Author UI. Anyone already familiar with DI-Guy Scenario will feel at home with DI-Guy Author.

Showing Author UI Windows

Once the Author UI is running, there are DI-Guy function calls that can be made in the IG that will show and hide 2D user interface elements of the Author UI. In the above example the Time Control window of the Author UI is shown by a function call made in the IG.

Opening a Scenario

The "Open Scenario" menu item uses DI-Guy function calls in the IG to load a previously created scenario. Said scenario can have been created previously by the Author-enhanced IG, or by using the DI-Guy Scenario program.

PeopleBlitzer

When the user changes the Input Mode to PeopleBlitzer, the IG is now responsible for sending most user input in the 3D window to DI-Guy Author. This input will mostly include mouse clicks and drags, keyboard presses, as well as the state of the Shift, Control, and Alt keys. When the IG sends the click and drag information to DI-Guy Author while in PeopleBlitzer Input Mode, DI-Guy Author will internally decide how to handle that information. It looks at its current PeopleBlitz settings, which include DI-Guy character type, appearance, action, and mind. It then creates the specified character and a path that begins at the clicked point, and ends at the mouse release point. The current character data is shown in the Author UI so the user can see and edit details of the new character.

3D Editing Visuals

When Author editing is enabled in the 3D window, specialized 3D polygonal objects to represent DI-Guy editing visuals will need to be shown. These visuals are created by DI-Guy Author calling virtual functions that were defined when DI-Guy Author was integrated into the IG. In the PeopleBlitz case, IG-native geometry for waypoints and paths are requested by DI-Guy Author. The IG creates these based on recommended geometry and material settings provided by DI-Guy Author, and positions and orients the visuals from data provided by DI-Guy Author. When DI-Guy Author determines that any of these visuals should be repositioned, re-colored, or destroyed, it will notify the IG of this.

CrowdBlitz

When the user selects the CrowdBlitz Input Mode, mouse inputs will have different effects. When the user is painting regions in the IG 3D window DI-Guy Author will need to make multiple queries against the IG's native terrain in order to set the points that comprise the region. These queries are made by DI-Guy Author calling virtual functions defined when DI-Guy Author was integrated into the IG.

When the user is done and saves the scenario, the current scenario state as it exists in the IG is saved to a .dss file.

Closing Author

When the IG user turns off DI-Guy Author, the Author UI is shut down, and most DI-Guy Author related native controls and menu items are hidden. The IG should once again run as if DI-Guy Author were not present.

3. IG Integration

If your IG already uses the DI-Guy Graphics API for rendering DI-Guy characters, the integration of DI-Guy Author should be fairly straight-forward.

If your IG does not yet use the DI-Guy Graphics API, that integration must happen first, before you add DI-Guy Author. A DI-Guy Author integration always has to be done in conjunction with a DI-Guy Graphics API integration.

Some of the scenario authoring concepts here will be clearer if you are familiar with the DI-Guy Scenario product. In short, DI-Guy Scenario is a stand-alone application that can create and edit DI-Guy scenarios interactively with a combination 3D interface and 2D interface. See the DI-Guy Scenario documentation for more information.

Programming Reference

In addition to the information here, there is a substantial amount of documentation in the DI-Guy Author source code. Most functions in the C++ header files have documentation comments with them. While doing the integration you will likely be referring to the source code documentation often.

In addition, there is an Open Scene Graph (OSG) based example IG implementation, with source code, to provide concrete examples of how the various functions should be called and classes written. Most likely you will copy the provided example classes and replace the OSG code with code specific to your 3D environment.

The classes provided by DI-Guy Author can be broken into two groups: classes for interchanging information between your IG and Author, and classes for rendering 3D editing visuals.

DI-Guy Author Interface Class

The class diguyAuthorInterface is the primary way you send and receive information to and from DI-Guy Author. Its functions can be broken down into the following groups:

Mouse Functions: These functions are called by you from the IG when Author editing is enabled. They provide DI-Guy Author with the 3D window input such as mouse moves and mouse button clicks it needs for editing the scenario.

Keyboard Function: Similar to mouse functions, but for keystrokes.

Selected Object Functions: These functions are called by you from the IG when Author editing is enabled. They tell DI-Guy Author which object was selected, usually by mouse click, in the 3D window.

Terrain and Intersection Virtual Functions: These functions will be called by DI-Guy Author when it needs information from the IG about the environment. This includes altitude at a point, converting screen to world coordinates and back, requests to update selected objects, and functions DI-Guy Author needs for generating its navigation mesh.

Author UI Functions: These functions can be called by you from the IG. They provide control for when the Author UI is launched and request that specific Author UI windows are shown and hidden.

Author UI Status Update Virtual Functions: These functions will be called by DI-Guy Author when the status of various parts of the Author UI application change. This gives the IG a chance to update its state on what is happening with the Author UI, such as enabling or disabling various native user interface controls when a window is opened or closed.

3D Visual Classes

There are a number of classes that provide the IG with the information and infrastructure needed to render 3D editing visuals. These will need to be subclassed by you so you can do things appropriate to your IG. Like for the DI-Guy Graphics API, you need to register functions for creating instances of your subclasses when DI-Guy Author needs an object pointer to the base class.

Note that these are *suggested* visual representations of editing visuals. If, for example, you would prefer different colors for editing visuals, you are free to ignore the colors specified by the visual's material. If you would prefer different geometry altogether for waypoints – e.g., a textured disc on the ground – you are free to ignore the polygons suggested in the reference implementation.

Most of the geometry visual classes follow the same template, one similar the DI-Guy Graphics API classes. Each of them has most of the following functions:

- constructor

Called when a new instance of the visual is needed. For example, if a new path is added to the scenario, a new multiline visual will be created. Note that actual geometry should not be generated until build() is called.

- build()

  Called when the actual geometry for the visual should be generated.

  It's common for build() and update() to use a shared utility function for creating polygons, etc. In the reference implementation this function is called update_geometry().

- unbuild()

  Called geometry created by build() should be freed. This is usually, but no always, when the visual is being destroyed.

- update()

  Called when the geometry for the visual needs to be updated in some way. This can be because then overall lines or polygons need to be (re)created, the position of the visual has changed, or the material has changed.

  As mentioned above, it's common for build() and update() to use a shared utility function for creating polygons, etc.

- draw()

  Called when the visual should be drawn in the scene. This is usally not necessary for scene-graph style renderers, as they handle drawing themselves.

- show()

  Called when the visual should be shown, typically by attaching it to a scene graph or updating a scene graph switch node.

- hide()

  Opposite of show().

…

Visual Materials

Author visual materials are used to specify basic color and other associated information suggested for rendering Author editing visuals. They specify pen color (used for drawing lines), brush color (used for filling polygons), line width, etc.

Author visuals in general require less information than full DI-Guy Graphics API materials as stored in diguyGraphicsMaterial objects. Because of this a "light-weight" class is used that contains only basic information. All Author visuals provide a pointer to an Author visual material.

The class diguyAuthorMaterial is used for accessing material information.

Multiline Visuals

Author "multiline" visuals are used to render path splines. They are a collection of line segments connected end to end. They should be rendered using the visual material's pen color. As paths become selected and unselected the color of the pen will change.

The class diguyAuthorVisualMultiline is used for accessing multiline visual information.

Waypoint Visuals

Author "waypoint" visuals are used to render path waypoints. Each waypoint is a collection of polygonal solids that show the waypoint's position and orientation. They should be rendered using the visual material's brush color. As waypoints become selected and unselected the color of the brush will change.

The class diguyAuthorVisualWaypoint is used for accessing waypoint visual information.

Object Handle Visuals

Author "object handle" visuals are used to render various simple objects that need only a position and orientation. These include path beads, character selection discs, and terrain point pickers. The object handles specify the basic shape the handle should have – disc, sphere, cone, etc. – and the radii, height and other measurements for these shapes. The colors of the object handles are generally dependent on their role.

The class diguyAuthorVisualObjectHandle is used for accessing object handle visual information.

Region Mesh Visuals

Author "region mesh" visuals are used to render the meshes needed for showing the bounds and contents of crowd regions and general regions. They contain the line segments that outline the region boundaries, and the polygons that make up the region interior. In general the materials for regions are translucent so that stacked regions can be represented.

The class diguyAuthorVisualRegionMesh is used for accessing region mesh visual information.

Region Paintbrush Visuals

The Author "paintbrush" represents the spherical volume that shows where region painting will occur in certain Author input modes. The material for the paintbrush is translucent so that the terrain inside the brush can be seen.

The class diguyAuthorVisualRegionPaintbrush is used for accessing paintbrush visual information.