IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent of: McFarland et al.
U.S. Patent No.: 5,959,633 Attorney Docket No.: 19473-0309IP1
Issue Date: Sep. 28, 1999
Appl. Serial No.: 08/726,091
Filing Date: Oct. 4, 1996
Title: METHOD AND SYSTEM FOR PRODUCING GRAPHICAL IMAGES

Mail Stop Patent Board
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

PETITION FOR *INTER PARTES* REVIEW OF UNITED STATES PATENT NO. 5,959,633 PURSUANT TO 35 U.S.C. §§ 311–319, 37 C.F.R. § 42

TABLE OF CONTENTS

EXHIBITS

GOOGLE1001	U.S. Patent No. 5,959,633 to McFarland et al. ("the '633 patent")
GOOGLE1002	Prosecution History of the '633 patent (Serial No. 08/726,091)
GOOGLE1003	Declaration of Dr. Anselmo Lastra
GOOGLE1004	U.S. Patent No. 5,883,639 to Walton et al. ("Walton")
GOOGLE1005	U.S. Patent No. 5,564,048 to Eick et al. ("Eick")
GOOGLE1006	Select portions of *Inside Visual C++, Second Edition: Version 1.5* by David J. Kruglinski, September 1, 1994 ("Kruglinski")
GOOGLE1007	Select portions of *The American Heritage Dictionary of the English Language* (3rd ed. 1992)
GOOGLE1008	*Micrografx, LLC, v. Google, Inc. and Motorola Mobility, LLC*, Civil Action No. 3:13-cv-03595-N, Plaintiff Micrografx, LLC's Preliminary Disclosure of Asserted Claims and Infringement Contentions dated January 6, 2014

I. INTRODUCTION

Google Inc., Samsung Telecommunications America, LLC, Samsung Electronics America, Inc., and Samsung Electronics Co., Ltd. ("Petitioners") petition for *Inter Partes* Review ("IPR") under 35 U.S.C. §§ 311–319 and 37 C.F.R. § 42 of claims 1-4, 6, 8-11, 13, and 15 ("the challenged claims") of U.S. Patent 5,959,633 ("the '633 patent"). Below, Petitioners demonstrate there is a reasonable likelihood of prevailing ("RLP") in their challenge of at least one claim identified as unpatentable in this Petition.

The '633 patent discloses a purported improvement to "computer graphics programs [that] provide tools within a computer program that allow a user to draw and edit a variety of shapes," but which were limited to drawing a predetermined set of shapes. GOOGLE1001 at 1:11-22. According to the '633 patent, that improvement was to "provid[e] a shape library external to the computer program" in which each shape had "external capabilities" that could produce "a graphical image of the shape." *Id.* at 1:41-50. As a result, according to the '633 patent, "[n]ew shapes may be easily added without rewriting the underlying computer program." *Id.* at 1:60-62.

But the claimed invention was not new. To the contrary, the '633 patent was improvidently granted without full consideration to the wide body of applicable prior art. For example, U.S. Patent No. 5,883,639 ("Walton") [GOOGLE1004] discloses the exact limitations that served as the basis for allowance, namely, "an external shape stored outside the computer program" that includes "external capabilities" capable of "production of a graphical im-

age of the external shape." *See* GOOGLE1002, Amendment filed Jan. 26, 1999, at 12. Specifically, Walton discloses accessing external shapes in the form of graphics objects, each object including an external shape ("graphic element") and external capabilities ("behavior element") that, among other things, can produce a graphical image of the shape. GOOGLE1004 at 13:13-17; GOOGLE1003, ¶¶ 36-45. Walton is not alone, as the other references cited herein likewise disclose this same functionality.

In sum, if the Office had been aware of Walton or the other cited references, the '633 patent never would have issued. Petitioners therefore request the Board to institute *inter partes* review of the challenged claims on the grounds set forth below.

II. MANDATORY NOTICES UNDER 37 C.F.R § 42.8

A. Real Parties-In-Interest Under 37 C.F.R. § 42.8(b)(1)

Google Inc., Samsung Telecommunications America, LLC, Samsung Electronics America, Inc., and Samsung Electronics Col., Ltd. are the real parties-in-interest.

B. Related Matters Under 37 C.F.R. § 42.8(b)(2)

Petitioners are not aware of any reexamination certificate or certificate of correction for the '633 patent. The Patent Owner (Micrografx, LLC) is a non-practicing entity that filed complaints alleging infringement of the '633 patent in lawsuits against Petitioners (*Micrografx, LLC, v. Google, Inc. and Motorola Mobility, LLC*, N.D. Tex., Case No. 3:13-cv-03595-N; *Micrografx, LLC, v. Samsung Telecommunications America, LLC, Samsung Electronics America, Inc., and Samsung Electronics Co., Ltd.,* N.D. Tex., Case No. 3:13-cv-03599-N). Both complaints were filed on September 9, 2013, and both actions remain pending. Peti-

tioners have also petitioned – on this same day – for *Inter Partes* Review of two other patents at issue in the above-noted litigation: U.S. Patent 6,057,854 (Davis, Jr. et al.) and U.S. Patent 6,552,732 (Davis, Jr. et al.).

C. Lead And Back-Up Counsel Under 37 C.F.R. § 42.8(b)(3)

Petitioners provide the following designation of counsel.

LEAD COUNSEL	BACK-UP COUNSEL
John C. Phillips, Reg. No. 35,322 12390 El Camino Real San Diego, CA 92130 Tel: 858-678-4304 / Fax 858-678-5099	Michael T. Hawkins, Reg. No. 57,867 3200 RBC Plaza, 60 South Sixth Street Minneapolis, MN 55402 Tel: 612-337-2569 / Fax 612-288-9696

D. Service Information

Please address all correspondence and service to the address of both counsel listed above. Petitioners also consent to electronic service by email at 19473-0309IP1@fr.com (referencing No. 19473-0309IP1 and cc'ing phillips@fr.com and hawkins@fr.com).

III. PAYMENT OF FEES – 37 C.F.R. § 42.103

Petitioners authorize the Patent and Trademark Office to charge Deposit Account No. 06-1050 for the petition fee set in 37 C.F.R. § 42.15(a) and for any other required fees.

IV. REQUIREMENTS FOR IPR UNDER 37 C.F.R. § 42.104

A. Grounds for Standing Under 37 C.F.R. § 42.104(a)

Petitioners certify that the '633 patent is eligible for IPR and that Petitioners are not barred or estopped from requesting IPR.

B. Challenge Under 37 C.F.R. § 42.104(b) and Relief Requested

Petitioners request IPR of claims 1-4, 6, 8-11, 13, and 15 of the '633 patent on the

grounds listed in the table below. In support, this Petition includes claim charts for each of these grounds and a supporting evidentiary declaration of Dr. Anselmo Lastra. (GOOGLE1003).

Ground	Claims	Basis for Rejection
Ground 1	**1-4, 6, 8-11, 13, and 15**	Anticipated under § 102 by U.S. Patent No. 5,883,639 to Walton et al. ("Walton")
Ground 2	**1-4, 6, 8-11, 13, and 15**	Obvious under § 103 by U.S. Patent No. 5,564,048 to Eick et al. ("Eick") in view of *Inside Visual C++, Second Edition: Version 1.5* by David J. Kruglinski (Kruglinski)

Walton and Eick are both prior art under at least 35 U.S.C. §102(e), having an effective filing date years before Oct. 4, 1996. Kruglinski is prior art under at least 35 U.S.C. §102(b), having a publication date before Oct. 4, 1995 (U.S. Copyright Reg. No. TX0004058221). None of these references were considered by the Examiner during prosecution of the '633 patent.

V. SUMMARY OF THE '633 PATENT

A. Brief Description

The '633 patent is directed to a computer system for producing graphical images for use in a computer graphics program. GOOGLE1001 at 1:5-59. The specification of the '633 patent contends that prior art systems "only enable a user to draw and edit a limited number of shapes. If additional shapes are desired, the computer program in the system must be modified to include the additional tools needed to draw and edit the desired shape."

Id. at 1:14-18. As described below, this characterization of the prior art was flawed, as there were numerous prior art references that overcome the supposed limitations. The '633 patent further explains that "one computer graphics system incorporates a limited component plug-in capability utilizing tables." *Id.* at 1:23-34. But this characterization of the prior art also was inconsistent with the state of the art at the time. The external shapes, according to the '633 patent, are computer programing objects libraries, or other data structures that are stored external to a computer program and that include data for use in producing graphical images. *Id.* at 3:17-29. External shapes include capabilities that "allow the generation of information required to produce a graphical image." *Id.* at 3:29-31. These capabilities can include "action methods" and "symbol methods." *Id.* "Action methods" define functions for receiving user interaction received by the application from an input device for creation or manipulation of graphical images defined by the external shapes. *Id.* at 6:19-29; 5:1-18. Examples of action methods include "the create action and edit actions." *Id.* at 5:12-13. "Symbol methods" define functions for creating, editing, rendering, modifying, reading, and/or writing a graphic image. *Id.* at 7:14-16; 5:19-25. One example of a symbol method includes a rotate function for rotating a graphic image. *Id.* at 5:25-30.

External shapes include external action data, which is data that is operated on by one or more action methods to define an external action. *Id.* at 6:4-6. External shapes further include external symbol data, which is data that is operated on by symbol methods to "create, edit, render, modify, read, or write a graphical object." *Id.* at 7:14-16.

Independent claim 1 is written nominally as a system claim, but in reality recites a generic computer, with method steps being recited in the context of a "computer program stored in [a] storage medium." Claim 8 and its dependent claims are so-called Beauregard claims, reciting the identical method steps to Claim 1 and its dependent claims (or in the case of claim 15, nearly identical method steps) in the context of a "computer program encoded on a computer-readable medium." Independent claims 1 and 8 recite broad elements of accessing an external shape object or library that includes capabilities for producing a graphical image. Such functionality was well known in the art at the time the application that matured into the '633 patent was filed. The dependent claims recite similarly broad features that were also well known in the art at the time of filing. Indeed, numerous software programming platforms implemented in the early to mid 1990's offered identical functionality of providing objects or libraries including capabilities for creating and manipulating graphical images that are stored external to applications. *See* GOOGLE1003 at ¶ 17.

B. Summary of the Original Prosecution

The '633 patent was filed on October 4, 1996. Prior to this filing date, several different systems and programming platforms described in patent applications and in other printed publications taught methods in which external objects and/or libraries containing code defining how to draw graphical images are accessed by one or more computer programs. None of these prior art references were before the Examiner during the prosecution of the '633 patent. Instead, the two independent claims 1 and 8 at issue in this IPR were

allowed in April 1999 after a brief examination involving prior art references that did not provide an accurate picture of the state of the art.

After all original 28 claims were rejected in the first office action, the applicant argued that claims 1-15 were patentable because the primary reference cited by the examiner did not explicitly suggest "an external shape stored outside the computer program" as recited in independent claims 1 and 8. GOOGLE1002 at pp. 101-104. In response to this argument, the examiner cited an additional reference (U.S. Patent No. 5,790,117 to Halviatti et al.) as teaching "an external shape stored outside the computer program." *Id.* at pp. 109-110.

The applicant countered that Halviatti does not teach an "external shape" and that the examiner did not provide a motivation to modify the primary reference (Visio) with the teachings of Halviatti. *Id.* at pp.124-126. However, the applicant did not address how the resulting combination of the systems described by Visio and Halviatti references would function or how such a resulting combination would not include an "external shape." *Id.* Rather than provide reasoning giving a motivation to modify the system taught by Visio with the teachings of Halviatti, the examiner allowed all claims with no express reasons for allowance other than the note in the Notice of Allowability that "this communication is responsive to the correspondence filed 1-26-99]." *Id.* at pp.129-130. Independent claims 1 and 8 were therefore allowed because the examiner accepted the applicant's argument that the cited references, in isolation, did not disclose:

> "a computer program further operable to: access an external shape

stored outside the computer program, the external shape comprising external capabilities; and delegate the production of a graphical image of the external shape to the external capabilities."

Id., pp. 114, response filed Jan. 26, 1999.

As described in more detail below, more pertinent prior art never considered by the examiner expressly disclosed an "external shape" and all other features of claims 1-4, 6, 8-11, 13, and 15.

VI. Claim Construction under 37 C.F.R. §§ 42.104(b)(3)

Claims are to be given their "broadest reasonable construction in light of the specification." 37 C.F.R. § 42.100(b). The constructions are intended to aid in this proceeding, and should not be understood as waiving any arguments concerning indefiniteness or claim breadth that may be raised in any litigation, which requires different construction standards.[1]

"external shape stored outside the computer program" (claims 1, 8) – an object, library component, or other data structure that contains source code, that is stored external to a computer program, and that includes data for use in producing a graphical image. GOOGLE1003 at ¶ 21. This interpretation is consistent with the broadest reasonable interpretation of these terms and with the specification of the '633 patent. *See* GOOGLE1001 at 2:66-3:7; 3:52-57; GOOGLE1003 at ¶ 21.

[1] The Patent Owner's preliminary infringement contentions are attached as Exhibit GOOGLE1008 for the Board's reference.

"external capabilities" (claims 1, 8, 15) – As defined in the '633 patent, "capabilities" are "action methods, symbol methods, or any other functions that allow the generation of information required to produce a graphical image." GOOGLE1001 at 3:29-31. "External capabilities," therefore, are such capabilities that are external to a computer program. GOOGLE1003 at ¶ 22. This interpretation is consistent with the broadest reasonable interpretation of these terms and with the specification of the '633 patent. *Id.* at ¶ 22.

"delegate" (claims 1,2, 8, 9) – the plain meaning of "delegate" is "to commit or entrust to another." GOOGLE1007 at p. 493. The '633 patent provides no express meaning for the term "delegate" that is different from this aforementioned definition. This interpretation is consistent with the broadest reasonable interpretation of these terms and with the specification of the '633 patent. *See* GOOGLE1001 at 3:17-31; GOOGLE1003 at ¶ 23.

"external action" (claims 2-4, 9-11) – functions and data that are external to the computer program and that are operable to receive user interaction. GOOGLE1003 at ¶ 24. This interpretation is consistent with the broadest reasonable interpretation of these terms and with the specification of the '633 patent. *See* GOOGLE1001 at 6:1-29.

"external symbol" (claims 2, 3, 6, 9, 10, 13) – functions and data that are external to the computer program and that are operable for creating, editing, rendering, modifying, reading, and/or writing a graphical image. GOOGLE1003 at ¶ 25. This interpretation is consistent with the broadest reasonable interpretation of these terms and with the specification of the '633 patent. *See* GOOGLE1001 at 7:11-16.

VII. THERE IS A REASONABLE LIKELIHOOD THAT AT LEAST ONE CLAIM OF THE `633 PATENT IS UNPATENTABLE

As detailed below, claims 1-4, 6, 8-11, 13, and 15 of the '663 patent are anticipated by at least one reference (Grounds 1) and rendered obvious by another combination of references (Ground 2). The two Grounds are not cumulative or redundant. Instead, they rely upon different primary references that individually assert unique benefits to the user and, additionally, they address the dependent claims in different ways, including using different statutory grounds of § 102 and § 103. *See* GOOGLE1003 at ¶¶ 26-164.

For example, the Walton patent, on which Ground 1 is based, describes a system for allowing users to create graphical user interfaces "without knowledge of programming languages and without having to learn a large set of complicated commands." GOOGLE1004 at 3:55-59. The system of Walton allows non-programming savvy lay-users to create a user interface and animations for graphical images by providing examples of how objects should respond to user input through a process of "animation by example." *Id.* at 3:59-67. The user can add "graphical objects" to the user interface and identify how the graphical objects are to respond to user input. *Id.* at 4:6-20.

By contrast, the Eick patent, on which Ground 2 is based, describes a C++ programming environment which provides graphical object classes, drawing classes, and function classes that allow a programmer to call the classes from applications written by the programmer. *See* GOOGLE1005 at Abstract; 4:19-42; 5:35-55. The system of Eick allows

a "programmer [to] use components from the [class] library in his program." (*Id.*at 1:51-57.) Specifically, the class libraries taught by Eick are intended for "use in object-oriented graphics programming." Thus the system of Eick is implemented for the benefit of computer programmers writing graphics programs in object-oriented languages, such as C++, while the system described by Walton is implemented for the benefit of non-programmers when creating user interfaces. The two Grounds of rejection differ in the way in which they address the claim elements of the '633 patent in numerous other ways, as detailed below.

A. Ground 1 sets forth a reasonable likelihood to prevail on at least one of Claims 1-4, 6, 8-11, 13, and 15

Referring to Ground 1 (charted below), Walton discloses accessing external graphical objects, each of which includes an external shape ("graphic element") and external capabilities ("behavior element") that, among other things, can produce a graphical image of the shape. GOOGLE1004 at 13:13-17; GOOGLE1003 at ¶¶ 36-45. Walton also discloses delegating the production of a graphical image to the external capabilities. GOOGLE 1004 at 11:8-9; 13:19-31; GOOGLE1003 at ¶¶ 46-49. Walton also discloses that the graphical objects are stored externally from a program ("user code"). GOOGLE1004 at 8:54-65; 21:10-16; 21:44-46.

More specifically, the system taught by Walton utilizes external graphical object libraries for storing graphical object data and input, drawing, and manipulation functions for various graphical objects that allow user applications to access the graphical objects and associated functionality stored within the external libraries. *See* GOOGLE1004 at 9:24-25;

9:46-47; 6:8-10; 6:17-20; 8:54-65; 21:10-16; GOOGLE1003 at ¶¶ 36-49. Annotated Fig. 3 of Walton below shows a modular arrangement for storing graphical object data and functionality in a "library of graphical objects 320" that is stored externally to an application, labeled as "user's source code 360:"

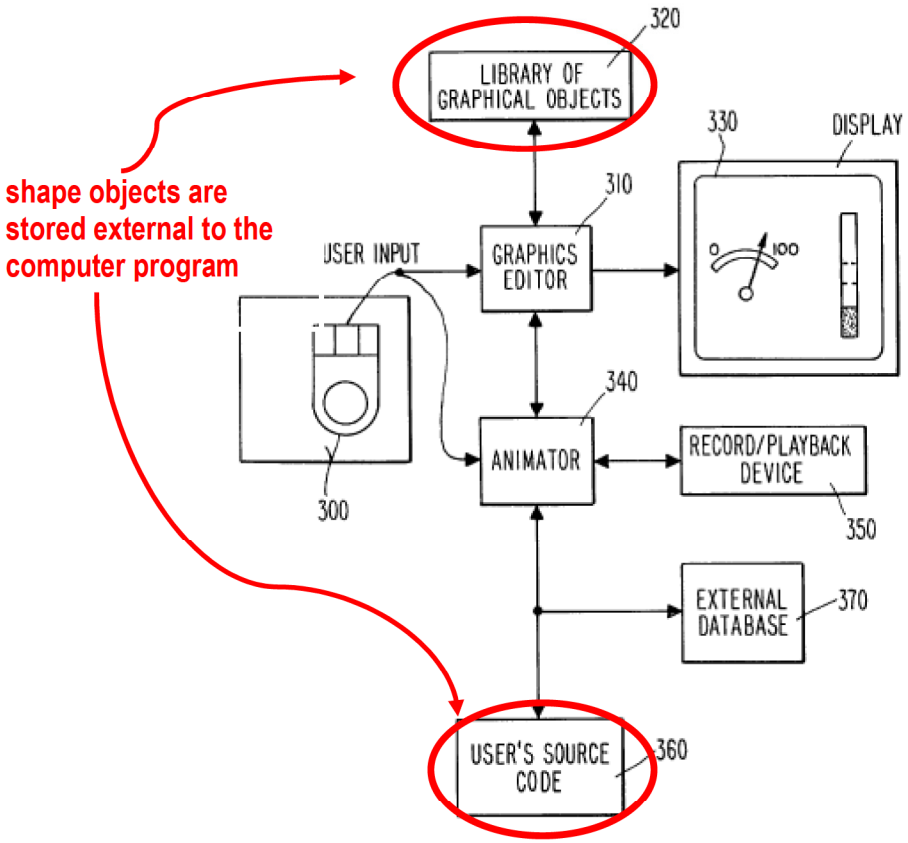


Fig. 3

Walton discloses that graphical objects are "stored as objects in an object-oriented database system and connected to other objects or user code" and that such techniques for storing the graphical objects external to the user code were "commonly used in object-oriented systems" at the time that the Walton patent was written. GOOGLE1004 at 8:54-59.

Walton further discloses that these external graphical objects can be "accessed by the user code 120" by connecting to "a client server via an interprocess communications mechanism of a type known to those skilled in the art." *Id.* at 8:58-62. Walton describes that storing the graphical objects externally to the user application is "particularly advantageous in that the user application code can read input and send outputs to the display screen using the created graphics objects without requiring the designer to write interface code." *Id.* at 8:24-27.

Walton further discloses delegating drawing and other functions relating to the external shapes to the externally stored graphical objects. For example, Walton indicates that "a graphical object in accordance with the invention **must be able to draw itself** if asked to do so." *Id.* at 13:19-21 (emphasis added). Walton additionally discloses that each "graphical object is responsible for controlling itself" and that a graphical object is responsible for "chang[ing] its graphical representation . . . on the display" in response to user input or other events. *Id.* at 11:8-9; 13:26-30.

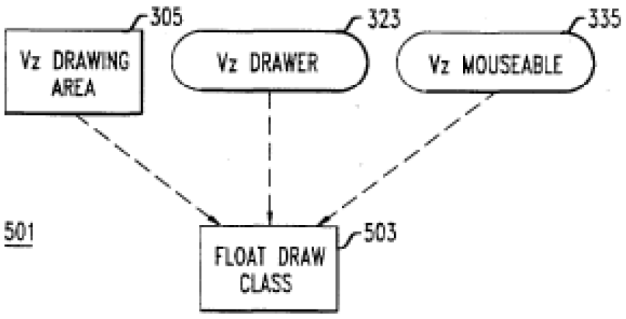
For at least these reasons, and the additional explanations set forth in the chart below, there is a reasonable likelihood that claims 1-4, 6, 8-11, 13, and 15 of the '633 patent are anticipated by Walton. *See, e.g.*, GOOGLE1003 at ¶¶ 26-49 (providing evidence that claim 1 is anticipated); *see also* ¶¶ 50-86. Moreover, to the extent the Patent Owner argues that any element of claims 1-4, 6, 8-11, 13, and 15 is not expressly disclosed in the Walton patent, such element would be inherently disclosed or rendered obvious in view of the disclosure of the Walton patent and in light of the knowledge of a POSITA, especially in light of

the fact that Walton provides a system (for storing, externally to a computer program, graphical elements having external capabilities) that is virtually the same as the preferred embodiment of the '633 patent. GOOGLE1003 at ¶ 88.

B. Ground 2 sets forth a reasonable likelihood to prevail on at least one of Claims 1-4, 6, 8-11, 13, and 15

Referring to Ground 2 (charted below), Eick is directed to a computer graphics system for drawing shapes in an object-oriented programming environment, such as C++. *See* GOOGLE1005 at 4:19-42. The system allows a program to call drawing classes from a class library and delegate various drawing functions to the called classes. *See Id.* at 1:51-57; 4:15-17; 5:35-55. Each class can inherit other classes to provide various functions and shape drawing capabilities to the inheriting class. *See Id.* at 6:6-9; 9:22-27; 13:1-8. For example, Eick discloses that any graphical object class "can inherit . . . any number of the functionality classes 203 defined by class library 211." *Id.* at 6:6-9. Inherited classes allow a graphical object class "to respond to drawing commands" and perform functions, such as "draw[ing] either the frame or a filled shape" for the graphical object. *Id.* at 53-56; 22:16-19. FIG. 5 shows library class FloatDraw inheriting the classes VzDrawingArea, VzDrawer, and VzMouseable:

FIG. 5



The inherited VzMouseable class allows FloatDraw to receive user input from a mouse, while the inherited VzDrawer class draws the actual object defined by a class that inherits FloatDraw. *Id.* at 5:23-34; 5:58-60; 6:54-61; 22:16-19. The drawing classes (such as VzBarLayout and FloatDraw) that are called from the class library are able to use functions and capabilities provided in inherited classes to draw and manipulate graphical objects on a display screen. *See Id.* at 8:40-47; 9:56-10:14; GOOGLE1003 at ¶¶ 89-92.

For example, the graphical object class "VzBarLayout" has capabilities for "drawing either rows or columns of bars and [allowing a user to use] the mouse to manipulate them." *Id.* at 8:40-44. Eick further discloses that the VzBarLayout class inherits the VzDrawingArea and VzMouseable classes and "is used to draw . . . either rows (horizontal) or columns (vertical) of bars" in response to tracked mouse movements. *Id.* at 13:1-3. The VzBarlayout class that draws bar shapes also inherits the VzDrawer class. *Id.* at 8:44-47.

Additionally, while Eick does not explicitly state that its C++ graphic object libraries are external libraries, the functionality described in Eick suggests implementation of the libraries of Eick as external libraries. (GOOGLE1003 at ¶ 105.) For example, Eick discloses

that “libraries of components of graphical user interface programs” can be implemented to allow a “programmer [to] use components from the library in his program and thus avoid having to write and debug them himself.” (GOOGLE1005 at 1:51-57.) Eick also describes the graphic object libraries not as being written as part of a graphics program, but as stand-alone objects that “may be used in writing a graphics program.” (GOOGLE1005 at 4:14-17.)

Additionally, persons of ordinary skill in the art prior to October 1996 commonly understood that a C++ library could be made into an external library, such as a .DLL or a shared library. GOOGLE1003 at ¶¶ 106-111. For example, Kruglinski, a text on a C++ programming language, describes that any class can be compiled into an external DLL library, describing such libraries as “an important part of Windows-based programming…. DLLs are Windows-based program modules that can be loaded and linked at run time. Many applications can benefit by being split into a series of main programs and DLLs.” *Id.*

Kruglinski describes an example project in which several classes are grouped into a DLL. *Id.* at pg. 645. The classes in Kruglinski’s example project "have mostly the same source code as their statically linked counterparts, except for some added code that demonstrate resource searching and runtime class access." *Id*. This use of external DLL files for storing libraries mirrors language in the ‘633 patent describing how external shapes are accessed by a user program. *See* the ‘633 patent at 3:52-67 (describing that the “shape collection modules 212 and 214 comprise a dynamic link library (DLL) that allows executa-

ble routines to be stored separately as files with DLL extensions and to be loaded only when needed by the program that calls them"); *see also* 4:27-32.

There are a number of reasons that would have prompted a person of ordinary skill in the art (POSITA) to combine this functionality disclosed in Kruglinski with the teachings of Eick. **First**, a POSITA would have been prompted to store the classes of Eick as DLL libraries as taught by Kruglinski because Eick describes that its library is written in C++, and a POSITA would understand, as taught in Kruglinski, that when programming in C++, "[m]any applications can benefit by being split into a series of main programs and DLLs." GOOGLE1006 at pg. 635; *see also* GOOGLE1003 at ¶¶ 108-111 One such benefit is suggested by Eick, which discloses that splitting an application into a series of main programs and DLLs would allow a programmer to write a graphics program without having to worry about writing and debugging the graphic object class libraries. GOOGLE1005 at 1:51-57.

Second, as a POSITA would have understood, storing libraries as DLL files allows for programs to be designed as a series of "modules" that can share various libraries, thereby allowing for quicker loading of programs, which would have prompted a POSITA to modify the system of Eick with the teachings of Kruglinski. Kruglinksi recognized this benefit: "You might have separate programs, or **modules**, for Payroll, Accounts Receivable, and so forth, but these programs have a lot of functionality in common. All modules might **share the same list management and database access classes**, for example. If you put the shared code in one or more DLLs, the individual modules will be smaller on disk and there-

fore **quicker to load**." GOOGLE1006 at pg. 635 (emphasis added). This same benefit was recognized by the '633 patent, touting that "the invention provides for modular production." GOOGLE1001 at 2:1-2.

Third, a POSITA would have been motivated to implement the libraries of Eick as external DLL files to achieve the benefit of allowing multiple applications to access the libraries, as taught by Kruglinski. *See* GOOGLE1003 at ¶ 110.

Fourth, a POSITA would have been prompted to modify Eick's system with this well-known feature of Kruglinski because doing so would be merely the use of a known technique (e.g., storing of libraries as external DLL files) to improve similar devices (e.g., object oriented programs implementing object libraries) in the same way. Indeed, "when a patent 'simply arranges old elements with each performing the same function it had been known to perform' and yields no more than one would expect from such an arrangement, the combination is obvious." *KSR Int'l Co. v. Teleflex Inc.*, 550 U.S. 398, 417 (2007). Here, both Eick and Kruglinski disclose the creation of class libraries using C++, with Kruglinski describing an improvement for the creation, storage, and access of such libraries. A POSITA would have readily applied the capability to store C++ class libraries as external DLL files (as taught by Kruglinski) to Eick's C++ class libraries, so as to provide a predictable result (e.g., a prior art method for storing class libraries). GOOGLE1003 at ¶ 111. Also, the resulting combination would continue to provide the original functionality taught by Eick (e.g., classes defining shapes having associated capabilities stored within libraries) while also providing

the additional benefits of external library storage described in Kruglinski. *Id.* at ¶111. Thus, the combination of Eick (which describes a C++ library) and Kruglinski (which is a text on the C++ programming language) is merely combining prior art elements according to known method to yield predictable results. *KSR*, 550 U.S. at 417.

Fifth, both Eick and Kruglinski are in the same field (object-oriented computer programming), and therefore a POSITA would have been motivated to utilize the object-oriented computer programming techniques taught by Kruglinski when implementing the computer graphics system of Eick.

For at least these reasons and the additional explanations described in the chart below and the accompanying declaration by Dr. Lastra, there is a reasonable likelihood that claims 1-4, 6, 8-11, 13, and 15 of the '633 patent are rendered obvious by Eick in view of Kruglinski. *See, e.g.*, GOOGLE1003 at ¶¶ 89-116 (providing evidence that claim 1 is rendered obvious); *see also* ¶¶ 117-162. Moreover, to the extent the Patent Owner argues that any element of claims 1-4, 6, 8-11, 13, and 15 is not expressly disclosed by the combination of Eick in view of Kruglinski, such element would be inherently disclosed or rendered obvious in view of the disclosures of the Eick and Kruglinski references and in light of the knowledge of a POSITA. *See Id.* at ¶ 164. Any such minor elements of the claims at issue that the Patent Owner argues are not expressly addressed by this petition would be known by a POSITA and could be easily and readily applied to the teachings of Eick and Kruglinski.

VIII. [GROUND 1 CLAIM CHART] – Anticipation of Claims 1-4, 6, 8-11, 13, and 15 under §102 by Walton

Claim Element	U.S. Pat. 5,883,639 to Walton et al.
[1.P] A computerized system comprising:	Walton discloses a computerized system. (GOOGLE1003 at ¶¶ 30-31.) For example, Walton describes "[a] visual software engineering system." (GOOGLE1004 at 7:44-47; *see also* Abstract ("A system for providing a simple, easy to learn and flexible means of creating user interfaces to products under development without the need of a programming language or the need to learn a large set of complicated commands."); 4:38-41 ("In accordance with a preferred embodiment of the invention, a **system is provided for creating and animating graphical objects** by directly manipulating the graphical objects on a display screen") (emphasis added).) Walton discloses that the system is "preferably implemented in **software on a computer** and is accessed via a window system 402." (emphasis added) (*Id*. at 9:52-54; *see also* 7:51-57 ("Since UNIX® work stations are in widespread use as developmental platforms in real-time projects, the preferred embodiment of the invention is described for use on UNIX® work stations. Also, since C and C++ computer languages are the primary implementation languages of interface developers, the source code listings attached as APPENDIX A have been prepared in these languages.").)
[1.1] a storage medium;	Walton discloses a storage medium. (GOOGLE1003 at ¶ 32.) For example, Walton discloses that a "user can also store the entire design of his or her user interface on disk" and that "objects [can] be saved and restored from disk files." (GOOGLE1004 at 4:29-33; 21:44-46.) Additionally, Walton describes that its system is "implemented in software on a computer and is accessed via a window system 402." (*Id.* at 9:52-54; *see also* 7:51-57 (describing implementation in C++ on UNIX workstations.) A POSITA would know that software requires a storage medium to execute on a com-

	puter. (GOOGLE1003 at ¶ 32.)
[1.2] a processor coupled to the storage medium;	Walton discloses a processor coupled to the storage medium. (GOOGLE1003 at ¶ 33.) Walton discloses that "[a] software interface is also easily connected to the actual code being constructed by the developer, where **such code may be running on the development host**, in an instruction set simulator on the host, **in the physical product's processor**, or on all three at once." (emphasis added) (GOOGLE1004 6:29-34.) Based on Walton's teaching and the broadest reasonable interpretation of "coupled" in this element, the processor would be coupled to a storage medium for the processor to be "running" the code. (GOOGLE1003 at ¶ 33.)
[1.3] a computer program stored in the storage medium, the computer program operable to run on the processor, the computer program further operable to:	Walton discloses a computer program stored in the storage medium that is operable to run on the processor. (GOOGLE1003 at ¶¶ 34-35.) For example, Walton describes a computer program in the form of user software. (GOOGLE1004 at 8:16-21 ("In particular, the **user software may be used to access the graphical object,** and by providing the behavior function name and the desired behavior state, the graphical object may be manipulated on the display screen directly from the user application code.") (emphasis added); 6:29-34 ("A software interface is also easily connected to the actual **code being constructed by the developer**, where such code may be running on the development host, in an instruction set simulator on the host, in the physical product's processor, or on all three at once.") (emphasis added); 10:17-21 ("A detailed description of each of the modules of FIGS. 4(a) and 4(b) will now be given a source code listing of the following preferred embodiment of the VSE system 400"); 9:35-40 ("In addition, the behavior states of the graphical objects created by the graphics editor 310 may also be **linked to user source code** 360 such that the source code 360 can manipulate the states of the graphical objects.") (emphasis added).) The user code / software described by Walton runs on

	a computer. (9:50-54 ("As will be more apparent from the following, the VSE system 400 is preferably implemented in **software on a computer** and is accessed via a window system 402.") (emphasis added).) Based on Walton's teaching and the broadest reasonable interpretation of this element, the computer program running on the processor of the computer would be stored in the storage medium. (GOOGLE1003 at ¶¶ 34-35.)
[1.4] access an external shape stored outside the computer program, the external shape comprising external capabilities; and	Walton discloses accessing an external shape stored outside the computer program, the external shape comprising external capabilities, in accordance with the broadest reasonable interpretation standards expressed above. (GOOGLE1003 at ¶¶ 36-45.) Walton describes accessing external shapes in the form of graphical objects, each object including an external shape ("graphic element") and external capabilities ("behavior element"). (GOOGLE1004 at 13:13-17 ("A graphical object is the key object for the VSE system 400 of the invention. Preferably, the VSE object consists of two major parts, the graphic element and the behavior element.").) Walton describes computer programs in the form of user source code. (*Id.* at 9:36-47; *see also* 18:40-46.) FIG. 3 shows the library of graphical objects 320 (i.e., "external shapes") being stored external to the computer program (user source code 360).

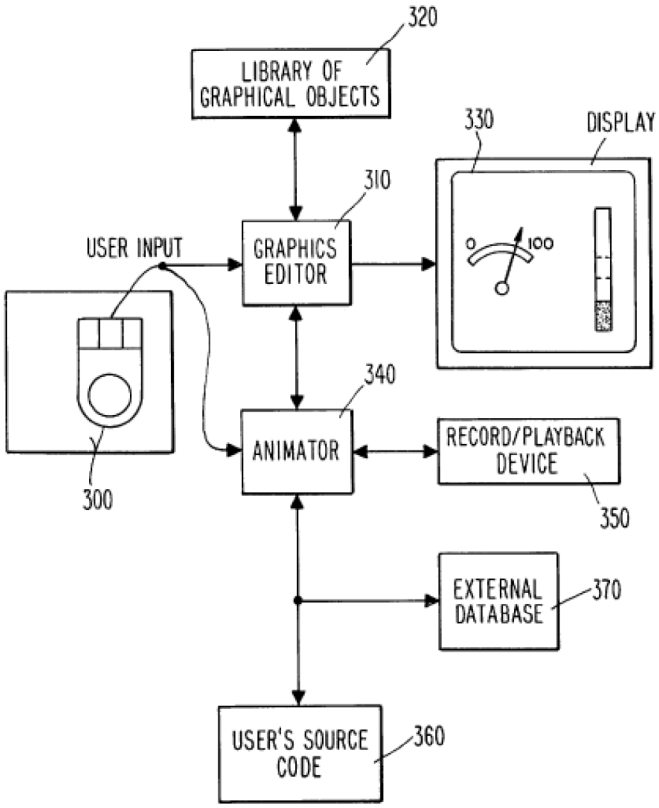


Fig. 3

Walton discloses that "the user software may be used to access the graphical object, and by providing the behavior function name and the desired behavior state, the graphical object may be manipulated on the display screen directly from the user application code." (*Id.* at 8:16-21; *see also* 9:4-8 ("[T]he user program preferably includes an interface 230 for accessing this animation information and for **calling the object and its associated behavior** and behavior value for the chosen animation.") (emphasis added).)

Walton discloses that shapes are stored as objects. (*Id.* at 6:38-40 ("The present invention is preferably designed using an object based technology where a drawn shape is a separately movable object."); *see also* 12:57-67.)

	Walton discloses that each shape object has external capabilities in the form of "input and output behavior states" and that these "behavior states can also be stored in external database 370." (*Id.* at 9:24-25; 9:46-47; *see also* 5:34-35 ("storing the defined output behavior with the corresponding graphical object for respective input states"); 6:8-10; 6:17-20 ("retrieving graphical objects with pre-defined input behavior states from a library.").) Walton discloses that the graphical objects are stored external to the computer program in that "[t]he **resulting objects are then stored as objects in an object-oriented database system** and connected to other objects or user code 120 in accordance with techniques commonly used in object-oriented systems. **The information so stored is accessed by the user code 120 by . . . communicating to a client server** via an interprocess communications mechanism of a type known to those skilled in the art." (emphasis added) (*Id.* at 8:54-62; *see also* 21:10-16 ("The client server 414 is the part of the VSE system 400 that connects user code to the VSE system 400 . . .[C]lient server 414 establishes the connection between a user program and the VSE system 400 and passes information between the two as required."); 14:19-21 (describing that user code can "receive graphical objects from other graphical editors outside of the VSE system 400.").) The graphics objects can also be saved to disk, external to the user code. For example, "Universe 420 is also provided as a place for storing all VSE graphical objects." (*Id.* at 10:1-2; *see also* 21:44-46 ("The universe 420 also allows objects to be saved and restored from disk files and views of the universe 420 to be added and deleted".); 11:23-30 ("The graphical file is created to store the graphical representations of objects for displaying or printing the appearance of those objects, while the design file contains the information the graphical file does along with descriptions of object

	relationships and object behavior functions. In other words, the graphical file stores pictures of objects, while the design file stores the appearance and behavior of objects.").) Walton further describes that the graphics objects have external capabilities in the form of "behavior states" that are stored external to user source code (*Id.* at 9:35-42 ("In addition, the **behavior states of the graphical objects created by the graphics editor 310 may also be linked to user source code** 360 such that the **source code 360 can manipulate the states of the graphical objects**. For this purpose, the user may access the stored behavior function and values for both input and output.") (emphasis added).) The graphic objects (i.e., "external shapes" in the parlance of the '633 patent) of Walton include external capabilities in the form of functions that allow the graphical objects to produce a graphical image. (GOOGLE1003 at ¶¶ 40-41, 44.) For example, "a graphical object in accordance with the invention **must be able to draw itself** if asked to do so and to indicate that it has been selected." (emphasis added) (GOOGLE1004 at 13:19-22; *see also* 11:8-9 ("the graphical element is responsible for controlling itself"); 25:62-26:4 ("the graphics for creating an interface to an object are created and then **input and output behavior are added** (as by clicks on mouse buttons). The object is then tested and saved. **The user then creates the desired application code and runs the application which calls upon particular objects**. The user application code is then connected to the VSE system 400 and the results displayed.") (emphasis added).) Walton describes the Visual Software Engineering (VSE) objects as "**stor[ing] behavior as well as graphics information**. This behavior information may represent any possible graphics transformation of a

	graphics object, such as change of color, move, rotate, scale, stretch, fill, map, unmap, raise and lower." (emphasis added) (*Id.* at 8:33-37; *see also* 9:40-42 ("the user may access the stored behavior function and values for both input and output.") Walton describes storing an object as a "graphical file stor[ing] pictures of objects" and a "design file stor[ing] the appearance and behavior of [the] objects." (*Id.* at 11:28-30; *see also* 13:15-17 ("the VSE object consists of two major parts, the graphic element and the behavior element."); 13:26-30 ("a VSE object of the invention tracks a behavior function (graphics manipulation) such that when a value change occurs (a behavior event), the VSE object can change its graphical representation and update itself on the display."); 18:63-64 ("When an object in universe 420 is saved to a design file or a palette, definitions for its associated behavior functions are saved as well.").)
[1.5] delegate the production of a graphical image of the external shape to the external capabilities.	Walton describes delegating the production of a graphical image of the external shape to the external capabilities, in accordance with the broadest reasonable interpretation standards expressed above. (GOOGLE1003 at ¶¶ 46-49.) For example, as discussed in claim element 1.4 above and incorporated here, Walton describes graphical objects stored outside of a computer program. (GOOGLE1004 at 8:54-65 ("[t]he resulting objects are then **stored as objects in an object-oriented database system and connected to other objects or user code** 120 in accordance with techniques commonly used in object-oriented systems. The information so stored is **accessed by the user code** 120 by . . . communicating to a client server via an interprocess communications mechanism of a type known to those skilled in the art.") (emphasis added); *see also* 5:34-35 ("storing the defined output behavior with the corresponding graphical object for respective input states"); 6:8-10; 6:17-20 ("retrieving graphical objects with pre-defined input behavior states from a library."); 9:46-47 (describing that objects and "behavior states can also be **stored in external database 370**.")

	(emphasis added).) Walton further discloses that the production of the graphical object is delegated to the external capabilities of an object in that "the graphical object is responsible for controlling itself." (*Id.* at 11:8-9; *see also* 13:19-21 ("a graphical object in accordance with the invention **must be able to draw itself** if asked to do so.") (emphasis added).) Specifically, for a graphical object to be displayed, "a VSE object of the invention tracks a behavior function (graphics manipulation) such that **when a value change occurs (a behavior event), the VSE object can change its graphical representation** and update itself on the display." (emphasis added) (*Id.* at 13:26-30.) Walton further describes delegation of production of a graphical object to external capabilities of the graphical object in the form of "input and output behavior states." (*Id.* at 9:24-25; *see also* 8:7-21.) Walton discloses that the "behavior information may represent any possible graphics transformation of a graphics object, such as change of color, move, rotate, scale, stretch, fill, map, unmap, raise and lower." (Id. at 8:33-37; *see also* 9:40-42 ("the user may access the stored behavior function and values for both input and output.") Walton describes storing an object as a "graphical file stor[ing] pictures of objects" and a "design file stor[ing] the appearance and behavior of [the] objects." (Id. at 11:28-30; *see also* 13:15-17 ("the VSE object consists of two major parts, the graphic element and the behavior element."); 18:63-64 ("When an object in universe 420 is saved to a design file or a palette, definitions for its associated behavior functions are saved as well.").)
[2.1] The computerized system of claim 1, wherein the computer program is further operable to: access an external shape stored outside the computer program,	As detailed above, Walton discloses all elements of claim 1. As demonstrated in claim element 1.4 above and incorporated here, Walton discloses accessing an external shape in the form of a graphic object. (GOOGLE1004 at 13:13-17 ("A graphical object is the key object for the VSE system 400 of the invention.

	Preferably, the VSE object consists of two major parts, the graphic element and the behavior element.").) Walton discloses that "the user software may be used to access the graphical object, and by providing the behavior function name and the desired behavior state, the graphical object may be manipulated on the display screen directly from the user application code." (Id. at 8:16-21; *see also* 9:4-8 ("[T]he user program preferably includes an interface 230 for accessing this animation information and for calling the object and its associated behavior and behavior value for the chosen animation.").) Walton also discloses that shapes are stored as objects. (Id. at 6:38-40 ("The present invention is preferably designed using an object based technology where a drawn shape is a separately movable object."); *see also* 12:57-67.) Moreover, Walton discloses that the graphical objects are stored external to the computer program in that "[t]he resulting objects are then stored as objects in an object-oriented database system and connected to other objects or user code 120 in accordance with techniques commonly used in object-oriented systems. **The information so stored is accessed by the user code 120 by . . . communicating to a client server** via an interprocess communications mechanism of a type known to those skilled in the art." (emphasis added) (Id. at 8:54-62; *see also* 9:46-47 (describing that objects and "behavior states can also be **stored in external database 370**.") (emphasis added); 21:10-16; 14:19-21; FIG. 3 (showing the "library of graphical objects 320" being stored outside the computer program ("user source code 360").).)
[2.2] the external shape comprising an external action and an external symbol; and	Walton discloses that the external shape objects have external actions in the form of functions and data that are external to the computer program and that are operable to receive user interaction for use in the creation, editing, rendering, modification, reading, and /or writing of graphical images. (GOOGLE1003 at ¶¶ 55-61.) For example, Walton discloses that each shape

	object has external actions in the form of "input and output behavior states" and that these "behavior states can also be stored in external database 370." (GOOGLE1004 at 9:24-25; 9:46-47.) Behavior states can define "user input operation[s]" that are associated with a shape object. (*Id.* at 4:52-55; *see also* 15:47-52.) Walton further discloses "retrieving graphical objects with pre-defined input behavior states from a library." (*Id.* at 6:17-20; *see also* 9:41-42 ("the stored behavior function [includes] input and output."); 6:8-10 ("storing the defined input behavior states with corresponding graphical objects."); 11:13-14 (describing that the system "asks the object itself what requests are legal.").) Additionally, Walton describes an object as defining input "behavior events" that lead to predefined outputs stored as part of the object: "a VSE object of the invention **tracks a behavior function** (graphics manipulation) such that **when a value change occurs (a behavior event), the VSE object can change its graphical representation and update itself on the display**." (emphasis added) (Id. at 13:26-30; *see also* 13:45-50 ("While receiving such user input, the graphical object registers with the behavior router 412 what events it wants next. When an event sequence has occurred, it can set the value of a behavior function and can also map events such as coordinate values and buttons to behavior function values.").) Walton further discloses "map[ping] a user defined input region . . . to a user input operation" for an object. (*Id.* at 15:49-50; *see also* 16:20-28 ("the details for a function of the behavior editor 408 includes the function name, the minimum and maximum numbers that establish the **range of values currently defined for the function's input parameter**, the selected value within the minimum to maximum range that should be passed to the function when the object is initially instantiated, **and an action list that textually describes the graphical operations that are performed by the**

	function and for what ranges of input values the operations are performed.") (emphasis added).) As demonstrated above in claim element 1.4 and incorporated here, Walton discloses storing an external symbol as part of a graphical object. (*Id.* at 11:28-30 (describing storing an object as a "graphical file stor[ing] pictures of objects" and a "design file stor[ing] the appearance and behavior of [the] objects."); 13:15-17 ("the VSE object consists of two major parts, the graphic element and the behavior element."); 11:23-30 ("The graphical file is created to store the graphical representations of objects for displaying or printing the appearance of those objects, while the design file contains the information the graphical file does along with descriptions of object relationships and object behavior functions. In other words, the graphical file stores pictures of objects, while the design file stores the appearance and behavior of objects.".) The graphics elements are manipulated by "behaviors [including] change color, move, rotate, scale, stretch, fill, map, unmap, raise, lower and the like." (*Id.* at 13:37-38.) Walton further describes that a VSE external graphical object "keeps the graphical state of the object as well as all the operations that can be performed on it. Graphical state information includes, for example, current graphical transformation, type of object, colors, line styles, and the like." (*Id.* at 21:65-22:4.)
[2.3] delegate the production of graphical image of the external shape to the external action and the external symbol.	Walton discloses delegating the production of graphical image of the external shape to the external action and the external symbol. (GOOGLE1003 at ¶¶ 62-66.) Walton describes that the production of the graphical object is delegated to its external action and external symbol in that "a graphical object in accordance with the invention **must be able to draw itself** if asked to do so and to indicate that it has been selected." (emphasis added) (GOOGLE1004 at 13:19-22; *see also* 11:8-9 ("the graphical element is responsible for con-

	trolling itself"); 25:62-26:4 ("the graphics for creating an interface to an object are created and then input and output behavior are added (as by clicks on mouse buttons). The object is then tested and saved. **The user then creates the desired application code and runs the application which calls upon particular objects**. The user application code is then connected to the VSE system 400 and the results displayed.") (emphasis added).) Specifically, for a graphical object to be displayed, "a VSE object of the invention tracks a behavior function (graphics manipulation) such that **when a value change occurs (a behavior event), the VSE object can change its graphical representation and update itself on the display**." (emphasis added) (*Id.* at 13:26-30.) For example, "for an object that accepts input from the user, a mouse or keyboard event may cause the object to execute one of its behavior functions." (18:19-21; *see also* 8:7-21("In accordance with the invention, an animation window is created which allows the user to assign a behavior function name to a graphical object and then manipulate the graphical object using a graphics editor to illustrate by example what the designer wants that particular graphical object to do. The present behavior of the manipulated graphical object is then stored as a behavior state with the current manipulation (display state) of the graphical object. In this manner, the designer creates "frames" of animation which may be later called upon to create the desired sequence of actions. In particular, the user software may be used to access the graphical object, and by providing the behavior function name and the desired behavior state, the graphical object may be manipulated on the display screen directly from the user application code."); 10:6-7 ("VSE primitive objects 424 are also provided for representing an object that the user has drawn on the display.").) Walton further describes delegation of production of a

	graphical object to "input and output behavior states." (*Id.* at 9:24-25.) Walton discloses that the "behavior information may represent any possible graphics transformation of a graphics object, such as change of color, move, rotate, scale, stretch, fill, map, unmap, raise and lower." (Id. at 8:33-37.) Walton further describes client code "initiat[ing] a behavior event" by "ask[ing] objects . . . to execute their respective F(integer value) behavior functions" for a specified parameter value. (Id. at 18:41-44.) Stored object code defines "graphical changes an object goes through" when a "behavior function is called with different values or ranges of values." (Id. at 18:55-59; *see also* 18:10-12 ("Each object, however, has its own definition for the action or actions performed by the behavior function."); 18:3-5 (describing that upon receiving a user input, the system "contacts each object and requests the object to execute its version of the behavior function with the desired value").)
[3.1] The computerized system of claim 2, wherein the computer program is further operable to receive user input in a manner defined by the external action; and	As detailed above, Walton discloses all elements of claims 1 and 2. Walton also discloses receiving user input in a manner defined by the external action. (GOOGLE1003 at ¶¶ 67-68.) Walton describes that a graphical object can define acceptable types of input. (GOOGLE1004 at 9:22-25 ("The invention of FIG. 3 is particularly characterized by an animator 340 which allows the user to define by example what input and output behavior states a graphical object should have.").) Walton discloses that the external actions stored as part of each graphic object define functions for receiving user interaction received by the computer program from an input device for manipulation of shapes that are not contained within the computer program. (GOOGLE1003 at ¶¶ 67-68.) For example, Walton discloses that each shape object has external capabilities in the form of "input and output behavior states" and that these "behavior states can also be stored in

	external database 370." (GOOGLE1004 at 9:24-25; 9:46-47.) Walton further discloses "retrieving graphical objects with **pre-defined input behavior states** from a library." (emphasis added) (Id. at 6:17-20; *see also* 9:41-42 ("the stored behavior function [includes] input and output."); 6:8-10 ("storing the **defined input behavior states with corresponding graphical objects**.") (emphasis added); 11:13-14 (describing that the system "asks the object itself what request are legal.").) Walton discloses how a user can identify which user inputs to associate with a particular graphical object. (*Id.* at 5:2-17("Preferably, the manipulating means comprises means for pressing and releasing one or more buttons on the example mouse pad under control of the hardware mouse and associated hardware mouse pad, whereby a button to be depressed on the example mouse pad is selected by the user by using the hardware mouse and the button on the example mouse pad is depressed by the user by depressing a corresponding button on the hardware mouse pad. The manipulating means preferably also comprises means for moving the example mouse on its associated example mouse pad under control of the hardware mouse and associated hardware mouse pad. As a result, the example mouse is selected by the user by using the hardware mouse and is moved by moving the hardware mouse such that the example mouse is moved a corresponding amount on its associated example mouse pad on the display screen relative to the movement of the hardware mouse.") Walton additionally discloses defining the input states as particular types of external actions (received user input). (13:42-50 ("the graphical object can also be set up to track sequences of user input such as a mouse button depression followed by a mouse drag, followed by mouse button release. **While receiving such user**

	input, the graphical object registers with the behavior router 412 what events it wants next. When an event sequence has occurred, it can set the value of a behavior function and can also map events such as coordinate values and buttons to behavior function values.") (emphasis added).)
[3.2] manipulate the graphical image in response to the user input in a manner defined by the external symbol.	Walton describes that a graphics object can be manipulated in response to behaviors executed in response to user input. (GOOGLE1003 at ¶¶ 69-72; GOOGLE1004 at 18:19-21 ("For an object that accepts input from the user, **a mouse or keyboard event may cause the object to execute one of its behavior functions**,") (emphasis added); GOOGLE1004 at 5:2-17("Preferably, the manipulating means comprises means for pressing and releasing one or more buttons on the example mouse pad under control of the hardware mouse and associated hardware mouse pad, whereby a button to be depressed on the example mouse pad is selected by the user by using the hardware mouse and the button on the example mouse pad is depressed by the user by depressing a corresponding button on the hardware mouse pad. The manipulating means preferably also comprises means for moving the example mouse on its associated example mouse pad under control of the hardware mouse and associated hardware mouse pad. As a result, the example mouse is selected by the user by using the hardware mouse and is moved by moving the hardware mouse such that the example mouse is moved a corresponding amount on its associated example mouse pad on the display screen relative to the movement of the hardware mouse.").) The graphics object uses its defined behaviors to manipulate a graphics image. (*Id.* at 13:26-28 ("Preferably, a VSE object of the invention tracks a behavior function (graphics manipulation) such that **when a value change occurs (a behavior event), the VSE object can change its graphical representation and update itself on the display**.") (emphasis added); *see also*

	13:37-38 (“behaviors [can include] change color, move, rotate, scale, stretch, fill, map, unmap, raise, lower and the like.”); 6:43-47 ("Mouse events that occur over these objects will cause the objects to execute their internal behavior. For example, a mouse click over a "button" object may cause the button to invert its border colors to simulate the button being pressed."); 16:43-56.) Walton further describes client code “initiat[ing] a behavior event” by “ask[ing] objects . . . to execute their respective F(integer value) behavior functions” for a specified parameter value. (Id. at 18:41-44.) Stored object code defines “graphical changes an object goes through” when a “behavior function is called with different values or ranges of values.” (Id. at 18:55-59; *see also* 18:10-12 (“Each object, however, has its own definition for the action or actions performed by the behavior function.”); 18:3-5 (describing that upon receiving a user input, the system “contacts each object and requests the object to execute its version of the behavior function with the desired value”).)
[4] The computerized system of claim 2 wherein the external action comprises a plurality of external methods and external data.	As detailed above, Walton discloses all elements of claims 1 and 2. Walton further discloses that the external action comprises a plurality of external methods and external data. (GOOGLE1003 at ¶¶ 73-74.) For example, Walton describes multiple behavior states (having multiple external methods and external data) associated with each object. (GOOGLE1004 at 5:62-64 (“The invention also preferably comprises a method of **defining user input to graphical objects as respective behavior states of the graphical objects**.”) (emphasis added); *see also* 5:34-35 (“storing the defined output behavior with the corresponding graphical object for respective input states”); 6:8-10; 6:17-20 (“retrieving graphical objects **with pre-defined input behavior states** from a library.”) (emphasis added); 17:24-30 (“An input object (i.e., an object whose appearance changes in response to a keyboard or mouse event) is preferably just a superset of an output object.

	The input object's necessary changes in appearance are mapped to states of a behavior function so that it is only necessary to map keyboard or mouse events to the range of values accepted by the behavior function.") (emphasis added); 9:40-42 ("For this purpose, the user may access the stored behavior function and values for both input and output.").) Walton further describes an external action comprising a plurality of external methods as well as external data. (GOOGLE1003 at ¶ 74.) For example, Walton discloses client code "initiat[ing] a behavior event" by "ask[ing] objects . . . to execute their respective F(integer value) behavior functions" for a specified parameter value. (GOOGLE1004 at 18:41-44.) Stored object code defines "graphical changes an object goes through" when a "behavior function is called **with different values or ranges of values**." (Id. at 18:55-59; *see also* 18:10-12 ("Each object, however, has its own definition for the action or actions performed by the behavior function."); 18:3-5 (describing that upon receiving a user input, the system "contacts each object and requests the object to execute its version of the behavior function **with the desired value**").) (emphasis added).
[6] The computerized system of claim 2 wherein the external symbol comprises a plurality of external methods and external data.	As detailed above, Walton discloses all elements of claims 1 and 2. Walton also discloses that the external symbol comprises a plurality of external methods and external data. (GOOGLE1003 at ¶¶ 75-77.) For example, Walton discloses that graphical objects are saved as multiple files, each file including information for rendering the object, including graphical files (external data) and design files (external methods). (GOOGLE 1004 at 11:23-30 ("The graphical file is created to **store the graphical representations of objects** for displaying or printing the appearance of those objects, while the design file contains the information the graphical file does along with **descriptions of object relationships and object behavior functions**. In other words, the graphical file stores pictures of objects, while the design file stores the appearance and behav-

	ior of objects.") (emphasis added).) Behaviors stored as part of graphic objects include external methods for manipulating the underlying graphics image and keeping track of external data in the form of behavior information. (8:34-38 "VSE objects in accordance with the invention thus store behavior as well as graphics information. **This behavior information may represent any possible graphics transformation of a graphics object**, such as change of color, move, rotate, scale, stretch, fill, map, unmap, raise and lower.") (emphasis added); *see also* 21:65-22:4 ("A VSE primitive object 424 is the part of the VSE system 400 that represents an object that the user has drawn on the screen. **It keeps the graphical state of the object as well as all the operations that can be performed on it**. Graphical state information includes, for example, current graphical transformation, type of object, colors, line styles, and the like.") (emphasis added.)
[8.P] A computer program encoded on a computer-readable medium, the computer program operable to:	Walton discloses a computer program encoded on a computer-readable medium. (GOOGLE1003 at ¶¶ 79-80.) For example, Walton describes a computer program in the form of user software. (GOOGLE1004 at 8:16-21 ("In particular, the user software may be used to access the graphical object, and by providing the behavior function name and the desired behavior state, the graphical object may be manipulated on the display screen directly from the user application code."); 6:29-34; 10:17-21.) The software runs on a computer. (*Id.* at 9:50-54 ("As will be more apparent from the following, the VSE system 400 is preferably implemented in software on a computer and is accessed via a window system 402."); *see also* 7:44-59.) Based on Walton's teaching and the broadest reasonable interpretation of this element, the computer program running on the processor of the computer would be encoded on a computer-readable medium of the computer. (GOOGLE1003 at ¶¶ 79-80.)
[8.1] access an external shape	*See* corresponding explanations for claim 1

stored outside the computer program, the external shape comprising external capabilities; and delegate the production of a graphical image of the external shape to the external capabilities.	
Claim 9	Claim 2 corresponds to claim 2. *See* explanations for claim 2.
Claim 10	Claim 10 corresponds to claim 3. *See* explanations for claim 3.
Claim 11	Claim 11 corresponds to claim 4. *See* explanations for claim 4.
Claim 13	Claim 13 corresponds to claim 6. *See* explanations for claim 6.
[15] The computer program of claim 8 wherein the external capabilities comprise a plurality of external methods and external data.	As detailed above, Walton discloses all elements of claim 8 (*See explanations for claim* 1). Walton also discloses external capabilities in the form of external methods of the external action and external methods of the external symbol. (GOOGLE1003 at ¶¶ 83-86.) For example, Walton describes an external action comprising a plurality of external methods as well as external data. (GOOGLE1003 at ¶¶ 83-86.) Walton discloses client code "initiat[ing] a behavior event" by "ask[ing] objects . . . to execute their respective F(integer value) behavior functions" for a specified parameter value. (GOOGLE1004 at 18:41-44.) Stored object code defines "graphical changes an object goes through" when a "behavior function is called with different values or ranges of values." (Id. at 18:55-59; *see also* 18:10-12 ("Each object, however, has its own definition for the action or actions performed by the behavior function."); 18:3-5 (describing that upon receiving a user input, the system "contacts each object and requests the object to execute its version of the behavior function with the desired value").) (emphasis added) Walton discloses external data in the form of external data of the external action and external data of the external symbol. (GOOGLE1003 at ¶ 84.) For example,

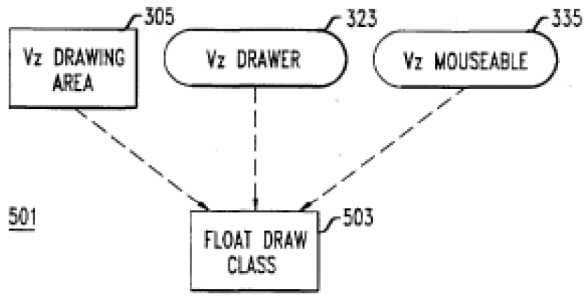
	Walton discloses that graphical objects are saved as multiple files, each file including information for rendering the object, including graphical files (external data) and design files (external methods). (GOOGLE 1004 at 11:23-30 ("The graphical file is created to store the graphical representations of objects for displaying or printing the appearance of those objects, while the design file contains the information the graphical file does along with descriptions of object relationships and object behavior functions. In other words, the graphical file stores pictures of objects, while the design file stores the appearance and behavior of objects.").)

IX. [GROUND 2 CLAIM CHART] – Obviousness of Claim 1-4, 6, 8-11, 13, and 15 under §103 by Eick in view of Kruglinski

Claim Element	Eick in view of Kruglinski
[1.P] A computerized system comprising:	Eick discloses a computerized system. (GOOGLE1003 at ¶¶ 93-94.) For example, Eick discloses "an object-oriented programming system 101 [] used to produce code for a graphical user interface. Object-oriented programming system 101 includes a compiler 107 for an object-oriented programming language which produces object code 109 for CPU 111. When CPU 111 executes code 109, the execution results in outputs 123 to display 113…. A user of CPU 111 may employ keyboard 117 and/or mouse 119 to provide inputs to window 115 … [that] affect the execution of code 109, and consequently may result in changes to window 115." (GOOGLE1005 at 4:20-31.)
[1.1] a storage medium;	Eick discloses "a storage medium." (GOOGLE1003 at ¶ 95.) For example, Eick describes using an Iris Indigo computer with 64 megabytes of memory. (GOOGLE1005 at 4:36-40 ("In a preferred embodiment, the object-oriented programming language is C++, compiler 107 is the Unix Software Laboratories (USL) C++ compiler, running on an Iris Indigo (Silicon Graphics Inc.) with 64 MBytes memory.").)
[1.2] a processor coupled to the storage medium;	Eick discloses a processor coupled to the storage medium. (GOOGLE1003 at ¶ 96.) For example, Eick describes us-

	ing a CPU to execute software code. (GOOGLE1005 at 4:25-26 ("When CPU 111 executes code 109, the execution results in outputs 123 to display 113.").) Based on Eick's teaching and the broadest reasonable interpretation of "coupled" in this element, the software code would be stored on a storage medium of Eick's computer for the CPU to access and execute it. (GOOGLE1003 at ¶ 96.)
[1.3] a computer program stored in the storage medium, the computer program operable to run on the processor, the computer program further operable to:	Eick discloses a computer program stored in the storage medium, the computer program operable to run on the processor. (GOOGLE1003 at ¶ 97.) For example, Eick describes the VZ library, which is "[a] library of C++ classes for use in writing data visualization programs." (GOOGLE1005 at Abstract.) Eick discloses that "Object-oriented programming system 101 includes a compiler 107 for an object-oriented programming language which produces object code 109 for CPU 111. When CPU 111 executes code 109, the execution results in outputs 123 to display 113. In response to these outputs, display 113 produces window 115." (*Id.* at 4:22-26.) Eick discloses that accessing of libraries and other functions recited within Eick "may be used in writing a graphics program." (*Id.* at 4:14-17.) Eick further discloses "An Example Graphics Program" that is implemented using the below detailed functionality. (*Id.* at 9:1-5)
[1.4] access an external shape stored outside the computer program, the external shape comprising external capabilities; and	Eick in view of Kruglinski discloses accessing an external shape stored outside the computer program, the external shape comprising external capabilities, in accordance with the broadest reasonable interpretation standards expressed above. (GOOGLE1003 at ¶¶ 89-92; 98-111.) Eick provides a C++ drawing class (VzDrawer) for creating shapes. (GOOGLE1005 at 6:54-61 ("VzDrawer 323 [is] a functionality class which **gives an area object the ability to respond to drawing commands**. Added operations include outputting the area as a Postscript file, doing double buffering, convening between coordinate systems, specifying fonts and measuring the size of strings, and **doing drawing operations** such as drawing points, lines, rectangles, ovals, polygons, filling polygons, and the like.") (emphasis added); *see also* 22:16-19 ("The remaining

	methods [of the VzDrawer class] draw into the current object. **They draw** either the frame or **a filled shape** and use either the pixel-based or floating valued coordinate systems. There are therefore ususally four routines for each drawing command.") (emphasis added); 20:36-59, (providing a list of drawing routines, including DrawPoint(), DrawLine(); FrameRect(); FillRect(); FrameOval(); FillOval(); FramePoly(); FillPoly() for drawing shapes.).) Eick further describes a C++ class "VzBarlayout" that is capable of "drawing either rows or columns of bars and using the mouse to manipulate them." (*Id.* at 8:40-44; *see also* 13:1-3 ("A VzBarLayout is--a VzDrawingArea and a VzMouseable that is used to draw and track the mouse for either rows (horizontal) or columns (vertical) of bars.").) The VzBarlayout class that draws bar shapes inherits the VzDrawer class. (*Id.* at 8:44-47.) Eick also describes the insertion of other shapes such as buttons, sliders, windows, and menus. (*Id.* at 7:27-31 ("The area classes define operations on areas such as windows, sliders, buttons, menus, and so on in display 113. In many graphical user interface systems, these areas are represented by native objects provided by the graphical user interface system."); *see also* 9:56-59 ("Objects of class FloatDraw 503 have three operations: reading a file to produce the information from which window 401 may be produced, **drawing [the] window 401**, and responding to mouse selections of line representations 405.") (emphasis added).) Eick's shapes are defined by object specifications that define the capabilities of an object. (*Id.* at 5:1-4 ("an object's class determines what operations can be performed on the object. The operations themselves are specified by means of function specifications in the class specifications 103.").) For example, the object may perform graphics operations and respond to input: "the program may perform graphics operations on the window represented by an object of class FloatDraw 503 and will also respond to mouse inputs on the window." (*Id.* at 9:27-30; *see also* 5:23-34 ("An example of the above is provided by the Vz library's class

	VzMouseable, which includes a virtual function DoMouse. **The virtual function interface defines how information concerning the behavior of the mouse is passed to code 109**, but does not define what action is taken in response to the information.") (emphasis added); *see also* 5:58-60 ("the functionality class VzMouseable provides the functionality of responding to a mouse.").) Eick provides "a library of C++ classes called Vz" (*Id.* at 2:64; *see also* 6:6-9 ("an object class specification 209 which inherits from any of the entity classes 201 defined by class library 211 can inherit any and any number of the functionality classes 203 defined by class library 211."); 9:22-27 ("FIG. 5 shows how classes of Vz library 301 are used in specifying FloatDraw 503. FloatDraw 403 inherits the library area class VzDrawingArea 305 and the library functionality classes VzDrawer 323 and VzMouseable 335.").) FIG. 5 shows library class FloatDraw inheriting VzDrawingArea, VzDrawer, and VzMouseable: *FIG. 5*  While Eick does not explicitly state that its C++ graphic object libraries are external libraries, the desired functionality described in Eick suggests implementation of the libraries of Eick as external libraries. (GOOGLE1003 at ¶ 105.) For example, Eick discloses that "libraries of components of graphical user interface programs" can be implemented to allow a "programmer [to] **use components from the library in his program and thus avoid having to write and debug them himself**." (emphasis added)

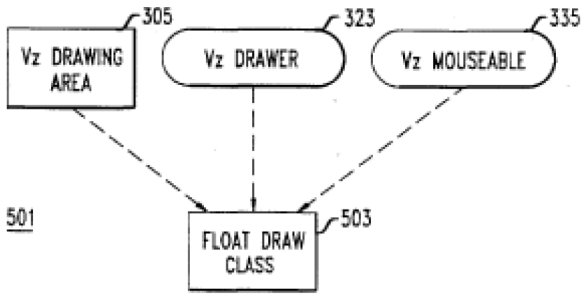
	(GOOGLE1005 at 1:51-57.) Such a teaching from Eick indicates that the libraries are external to the programmer's program and are accessed by the program, rather than included as a part of the program. (GOOGLE1003 at ¶ 105.) Eick also describes the graphic object libraries not as being written as part of a graphics program, but as standalone objects that "may be used in writing a graphics program." (GOOGLE1005 at 4:14-17.) Additionally, to the extent that Eick does not expressly disclose that its libraries are implemented as external libraries, it was well known to persons of ordinary skill in the art that any C++ library could be made into an external library such as a .DLL or a shared library. (GOOGLE1003 at ¶ 106-111.) For example, Kruglinski, a text on a C++ programming language, describes that any class can be compiled into an external DLL library. (GOOGLE1006 at pg. 635 ("The dynamic link library has always been an important part of Windows-based programming…. DLLs are Windows-based program modules that can be loaded and linked at run time. Many applications can benefit by being split into a series of main programs and DLLs.").) Kruglinski describes an example project "that combines the CStudent class …, CPersistentFrame class …, and the CRowView class … into a single dynamic link library." (*Id.* at pg. 645.) The classes in the example project for DLL "have mostly the same source code as their statically linked counterparts, except for some added code that demonstrate resource searching and runtime class access." (*Id.*) This use of external DLL files for storing libraries mirrors language in the '633 patent describing how external shapes are accessed by a user program. (*See* GOOGLE1001 at 3:52-67 ("shape collection modules 212 and 214 comprise a dynamic link library (DLL) that allows executable routines to be stored separately as files with DLL extensions and to be loaded only when needed by the program that calls them."; 4:27-32 ("The shared library 130 may also be a DLL. However, the present invention contemplates any suitable software architecture using dynamic

	link libraries, plug-ins, extensions, initialization files, or other modular arrangement that allows utility functions to be stored externally to computer graphics application 122.").) For the reasons described above in Section VII.B. and incorporated here, a person of ordinary skill in the art would have been prompted to modify Eick's system to include storing the Eick's graphical object libraries as external DLL files (as taught by Kruglinski).
[1.5] delegate the production of a graphical image of the external shape to the external capabilities.	The resulting combination of Eick in view of Kruglinski discloses the delegation of the production of a graphical image of the external shape to the external capabilities, in accordance with the broadest reasonable interpretation standards expressed above. (GOOGLE1003 at ¶¶ 112-116.) For example, Eick describes objects including functionality for drawing shapes of the objects. (GOOGLE1005 9:56-59 ("Objects of class FloatDraw 503 have three operations: reading a file to produce the information from which window 401 may be produced, **drawing [the] window 401**, and responding to mouse selections of line representations 405.") (emphasis added).) Eick additionally discloses that the VzBarLayout class "is used to draw . . . either rows (horizontal) or columns(vertical) of bars" based on tracked mouse movements. (*Id.* at 13:1-3.) As another example, Eick describes that "an object of class FloatDraw" can inherit "functionalities" from several classes that allow "the program [to] perform graphics operations" in response to mouse inputs. (*Id.* at 9:26-30.) Eick further describes yet another external capability in the form of a DoExpose function that is responsible for drawing the graphics images. (*Id.* at 9:56-63 ("Objects of class FloatDraw 503 have three operations: reading a file to produce the information from which window 401 may be produced, drawing window 401, and responding to mouse selections of line representations 405. **All of these operations are specified as virtual functions** 102, the read operation as the ReadFile virtual function at 617, **the drawing operation as the DoExpose function** at 619, and the response to the mouse selections at DoMouse 621.") (emphasis added).) For example, the DoExpose

	function can draw lines. (*Id.* 10:6-13 ("The display of columns 403 and line representations 405 is produced when window 401 is exposed by the implementation of DoExpose at 625 in FIG. 6B. The function determines the current size of the window and then uses the arrays produced by ReadFile to produce the display. A **line representation 405 is drawn for each line**; if there is a match in the line, the line representation 405 is highlighted.") (emphasis added).) The DoExpose function is called any time a shape needs to be redrawn. (*Id.* at 23:15-19 (disclosing that DoExpose is a "pure virtual member-function. It is called when a region of the VzDrawingArea needs to be redrawn. The rectangle passed frames the region.").) The VzDrawer class also has external capabilities for producing graphical images. (*Id.* at 22:16-19 ("The remaining methods [of VzDrawer class] draw into the current object. **They draw either the frame or a filled shape** and use either the pixel-based or floating valued coordinate systems. There are therefore usually four routines for each drawing command.") (emphasis added); *see also* 20:36-59, (providing a list of drawing routines, including DrawPoint(), DrawLine(); FrameRect(); FillRect(); FrameOval(); FillOval(); FramePoly(); FillPoly() for drawing shapes.).) While Eick does not explicity disclose that its C++ graphical objects (i.e., external shapes) are "external," the desired functionality described in Eick suggests implementation of the graphical objects of Eick as external libraries. (GOOGLE1003 at ¶¶ 105; 116.) For example, Eick discloses that "libraries of components of graphical user interface programs" can be implemented to allow a "programmer [to] use components from the library in his program and thus avoid having to write and debug them himself." (GOOGLE1005 at 1:51-57.) Additionally, the resulting combination of Eick in view of Kruglinski (explained above with respect to element 1.4, the explanation of which is incorporated here within) would provide Eick's graphical objects, and thus their associated functionalities (i.e., the "ex-

	ternal capabilities") implemented as external DLL files as expressly suggested by Kruglinski. (GOOGLE1003 at ¶ 106-111; 116;GOOGLE1006 at pp. 635; 645) For the reasons described above and incorporated here, a person of ordinary skill in the art would have been prompted to modify Eick's system to include storing the Eick's graphical object libraries as external DLL files (as taught by Kruglinski).
[2.1] The computerized system of claim 1, wherein the computer program is further operable to: access an external shape stored outside the computer program,	As detailed above, the resulting combination of Eick in view of Kruglinski provides all elements of claim 1. The resulting combination of Eick in view of Kruglinski also provides accessing an external shape stored outside the computer program. (GOOGLE1003 at ¶¶ 117-119.) For example, Eick provides a C++ drawing class (VzDrawer) for creating shapes. (GOOGLE1005 6:54-61 ("VzDrawer 323, a functionality class which gives an area object the ability to respond to drawing commands. Added operations include outputting the area as a Postscript file, doing double buffering, convening between coordinate systems, specifying fonts and measuring the size of strings, and doing drawing operations such as drawing points, lines, rectangles, ovals, polygons, filling polygons, and the like.") *see also* 22:16-19 ("The remaining methods [of the VzDrawer class] draw into the current object. They draw either the frame or a filled shape and use either the pixel-based or floating valued coordinate systems. There are therefore ususally four routines for each drawing command."); 20:36-59, (providing a list of drawing routines, including DrawPoint(), DrawLine(); FrameRect(); FillRect(); FrameOval(); FillOval(); FramePoly(); FillPoly() for drawing shapes.).) While Eick does not explicity disclose that its C++ graphical objects (i.e., external shapes) are "external," the desired functionality described in Eick suggests implementation of the graphical objects of Eick as external libraries. (GOOGLE1003 at ¶¶ 105; 119.) For example, Eick discloses that "libraries of components of graphical user interface programs" can be implemented to allow a "programmer [to] use components from the library in his program and thus avoid having to write and debug them himself."

	(GOOGLE1005 at 1:51-57.) Additionally, the resulting combination of Eick in view of Kruglinski (explained above with respect to element s1.4 and 1.5, the explanation of which is incorporated here within) would provide Eick's graphical objects (i.e., the "external shapes"), and thus their associated functionalities (i.e., "external capabilities") implemented as external DLL files as taught by Kruglinski. (GOOGLE1003 at ¶¶ 106-111; 119; GOOGLE1006 at pp. 635; 645)
[2.2] the external shape comprising an external action and an external symbol; and	The resulting combination of Eick in view of Kruglinski provides the external shape comprising an external action and an external symbol. (GOOGLE1003 at ¶¶ 120-128.) As detailed with respect to element 2.1 above, Eick provides a C++ drawing class (VzDrawer) for creating shapes. (GOOGLE1005 6:54-61; *see also* 22:16-19; 20:36-59.) Eick further describes a C++ class "VzBarlayout" that is capable of "drawing either rows or columns of bars and using the mouse to manipulate them." (*Id.* at 8:40-44.) The class VzBarlayout includes external actions in the form of the inherited class "VzMouseable" that is used by the VzBarlayout object to "track the mouse" and further includes an external symbol in the form of information for drawing "either rows (horizontal) or columns (vertical) of bars." (*Id.* at 13:1-3.) The VzBarlayout class that draws bar shapes inherits the VzDrawer class. (*Id.* at 44-47.) Eick further describes "[o]bjects of the class FloatDraw" as including external symbols in the form of "information from which window 401 may be produced" and external actions in the form of information for "responding to mouse selections of line representations 405." (*Id.* at 9:56-59.) FloatDraw uses this information for "drawing [the] window 401." (*Id.*) Eick further describes that FloatDraw includes external actions in the form of code that allows the class to "respond to mouse inputs on the window" to "perform graphics operations on the window represented by an object of class FloatDraw 503." (*Id.* at 9:27-30.) FIG. 5 shows library class FloatDraw inheriting external symbols (VzDrawingArea, VzDrawer), and an external action

	(VzMouseable): *FIG. 5*  Eick also describes the insertion of other shapes such as buttons, sliders, windows, and menus. (*Id.* at 7:27-28) ("The area classes define operations on areas such as windows, sliders, buttons, menus, and so on in display 113. In many graphical user interface systems, these areas are represented by native objects provided by the graphical user interface system."); *see also* 22:62-64;) Eick's shapes are defined by object specifications that define the external symbols and actions for an object. (*Id.* at 5:1-4 ("an object's class determines what operations can be performed on the object. The operations themselves are specified by means of function specifications in the class specifications 103.") Eick further describes action classes, including "VzMouseable" and "VzKeyable" that can be inherited by the above-described shape classes that define how input is received by the shape classes. (*Id.* at 5:23-34 ("An example of the above is provided by the Vz library's class VzMouseable, which includes a virtual function DoMouse. The virtual function interface **defines how information concerning the behavior of the mouse is passed to code 109**, but does not define what action is taken in response to the information. A class which inherits VzMouseable must provide an ordinary function 104 for DoMouse.") (emphasis added); 7:1-2 ("VzMouseable 335 is a class which provides

	an area object with the ability to respond to input from a mouse."); 5:58-60 ("the functionality class VzMouseable provides the functionality of responding to a mouse."); 7:9-15 ("VzKeyable 339 is a class which provides an area object with the ability to respond to input from the keyboard.").) Eick discloses additional examples of external symbol information in the form of VZ color management classes containing external symbol information for manipulating and creating shapes. (*Id.* at 17:1-6 ("A VzColorManager is a VzFunctionality which adds the ability to define colors and color ranges and to **create and use overlay layers**. There are no virtual functions in this class which the user needs to define. The intended purpose is for other classes to inherit from this class and gain access to the color management routines.") (emphasis added).) As described above, Eick provides "a library of C++ classes called VZ" (*Id.* at 2:64; *see also* 6:6-9; 9:22-27.) While Eick describes that its library is written in C++, Eick does not explicitly state that the library is an "external" library. However, it was well-known to persons of ordinary skill in the art that any C++ library could be made into an external library such as a .DLL or a shared library. (GOOGLE1003 at ¶¶ 105-107; 128.) For example, Kruglinski, a text on a C++ programming language, describes that any class can be compiled into an external DLL library. (GOOGLE1006 at pg. 635 ("The dynamic link library has always been an important part of Windows-based programming…. DLLs are Windows-based program modules that can be loaded and linked at run time. Many applications can benefit by being split into a series of main programs and DLLs.").) For the reasons described above and incorporated here, a person of ordinary skill in the art would have been prompted to modify Eick's system to store the C++ class libraries of Eick as external DLL files (as taught by Kruglinski). (GOOGLE1003 at ¶¶ 106-111; 128.)
[2.3] delegate the production of graphical image of the ex-	The resulting combination of Eick in view of Kruglinski provides for delegating the production of graphical image of

ternal shape to the external action and the external symbol.	the external shape to the external action and the external symbol. (GOOGLE1003 at ¶¶ 129-136.) For example, Eick describes objects including functionality for drawing shapes of the objects. (GOOGLE1005 at 9:26-30 ("By virtue of the functionalities inherited from classes 323 and 335, the program may perform graphics operations on the window represented by an object of class FloatDraw 503 and will also respond to mouse inputs on the window.").) Eick additionally discloses that the VzBarLayout class "is used to draw . . . either rows (horizontal) or columns(vertical) of bars" based on tracked mouse movements. (*Id.* at 13:1-3.) The FloatDraw shape class inherits the DoExpose function (external symbol information defining manipulation of a shape) and DoMouse (an external action). (*Id.* at 9:56-63 ("Objects of class FloatDraw 503 have three operations: reading a file to produce the information from which window 401 may be produced, **drawing [the] window 401**, and responding to mouse selections of line representations 405. All of these operations are specified as virtual functions 102, the read operation as the ReadFile virtual function at 617, **the drawing operation as the DoExpose function at 619**, and the response to the mouse selections at DoMouse 621.") (emphasis added).) For example, the DoExpose function can create lines. (*Id.* at 10:6-13 "The display of columns 403 and line representations 405 is produced when window 401 is exposed by the implementation of DoExpose at 625 in FIG. 6B. The function determines the current size of the window and then uses the arrays produced by ReadFile to produce the display. **A line representation 405 is drawn for each line**; if there is a match in the line, the line representation 405 is highlighted.") (emphasis added).) Indeed, the DoExpose function is called any time a shape needs to be redrawn. (*Id.* at 23:15-19 ("virtual void DoExpose(int left, int top, int width, int height)=0 A pure virtual member-function. It is called when a region of the VzDrawingArea needs to be redrawn. The rectangle passed frames the region.").)

	The VzDrawer class also has other external symbols for producing graphical images. (*Id.* at ("The remaining methods [of VzDrawer class] draw into the current object. They draw either the frame or a filled shape and use either the pixel-based or floating valued coordinate systems. There are therefore ususally four routines for each drawing command.") Furthermore, the C++ class "VzBarlayout" is capable of producing graphical images by "drawing either rows or columns of bars and using the mouse to manipulate them." (*Id.* at 8:40-44.) The class VzBarlayout includes external actions in the form of the inherited class "VzMouseable" that defines inputs used by the VzBarlayout object and further includes an external symbol in the form of information for producing graphical images of "either rows (horizontal) or columns (vertical) of bars." (*Id*. at 13:1-3.) The VzBarlayout class that draws bar shapes inherits the VzDrawer class. (*Id*. at 44-47.) The "VzColorManager 321…class specifies the functionality needed to control the use of colors in an area of a display 113. Operations added by this class include allocating colors, mapping colors to a scale of values, overlaying colors, turning colors off and on, and establishing foreground and background color." (*Id. at* 6:48-53.) The VZDrawer class (and therefore the VzBarlayout class) has the VzColorManager capabilities. (*Id.* at 6:54-56 ("VzColorManager 321 is in turn inherited by VzDrawer 323, a functionality class which **gives an area object the ability to respond to drawing commands**.") (emphasis added).) As described above with respect to claim element 2.2, Eick further describes external action classes, including "VzMouseable" and "VzKeyable," that can be inherited by the above-described shape classes and that define how input is received by the shape classes. (*Id.* 7:1-2 ("VzMouseable 335 is a class which provides an area ob-

	ject with the ability to respond to input from a mouse."); 7:9-15 ("VzKeyable 339 is a class which provides an area object with the ability to respond to input from the keyboard."); 5:23-34; 5:58-60).) While Eick does not explicity disclose that its C++ graphical objects (i.e., the '633 patent's "external shapes") are "external," the desired functionality described in Eick suggests implementation of the graphical objects of Eick as external libraries. (GOOGLE1003 at ¶¶ 105; 136.) For example, Eick discloses that "libraries of components of graphical user interface programs" can be implemented to allow a "programmer [to] use components from the library in his program and thus avoid having to write and debug them himself." (GOOGLE1005 at 1:51-57.) Additionally, the resulting combination of Eick in view of Kruglinski (explained above with respect to element s1.4 and 1.5, the explanation of which is incorporated here within) would provide Eick's graphical objects, and thus their associated functionalities (i.e., "external capabilities") implemented as external DLL files as taught by Kruginski. (GOOGLE1003 at ¶¶ 106-111; 136; GOOGLE1006 at pp. 635; 645)
[3.1] The computerized system of claim 2, wherein the computer program is further operable to: receive user input in a manner defined by the external action; and	As detailed above, the resulting combination of Eick in view of Kruglinski provides all elements of claim 2. This combination also provides receiving user input in a manner defined by the external action. (GOOGLE1003 at ¶¶ 137-141.) For example, Eick's classes "VzMouseable" and "VzKeyable" can be inherited by shape classes and define how input is received by the shape classes. (GOOGLE1005 at 5:23-34 ("An example of the above is provided by the Vz library's class VzMouseable, which includes a virtual function DoMouse. The virtual function interface **defines how information concerning the behavior of the mouse is passed to code 109**, but does not define what action is taken in response to the information. A class which inherits VzMouseable must provide an ordinary function 104 for DoMouse. For example, the ordinary function 104 might provide that if the mouse is used to select a portion of the text on display in a window, the selected portion is highlighted in the display and saved for use in

	a later operation such as moving or deleting the text.") (emphasis added); 7:1-2 ("VzMouseable 335 is a class which provides an area object with the ability to respond to input from a mouse."); 5:58-60 ("the functionality class VzMouseable provides the functionality of responding to a mouse."); 7:9-15 ("VzKeyable 339 is a class which provides an area object with the ability to respond to input from the keyboard.").) The FloatDraw shape class inherits the DoExpose function (external symbol information defining manipulation of a shape) and DoMouse (an external action defining what input can be received). (*Id*. at 9:56-63 ("Objects of class FloatDraw 503 have three operations: reading a file to produce the information from which window 401 may be produced, drawing window 401, and **responding to mouse selections of line representations** 405. All of these operations are specified as virtual functions 102, the read operation as the ReadFile virtual function at 617, the drawing operation as the DoExpose function at 619, and **the response to the mouse selections at DoMouse 621**.") (emphasis added); *see also* 9:27-30 ("the program may perform graphics operations on the window represented by an object of class FloatDraw 503 and will also respond to mouse inputs on the window.").) Additionally, the VzBarlayout class, which is capable of producing graphical images by "drawing either rows or columns of bars and using the mouse to manipulate them," includes external actions in the form of the inherited class "VzMouseable" that defines the manner in which user input can be received. (*Id.* at 8:40-44; 13:1-3.)
[3.2] manipulate the graphical image in response to the user input in a manner defined by the external symbol.	The resulting system of Eick in view of Kruglinski also discloses the manipulation of the graphical image in response to the user input in a manner defined by the external symbol. (GOOGLE1003 at ¶¶ 142-145.) For example, Eick plainly discloses "[t]he ordinary function which is executed in response to a selection of a highlighted line by responds to selection by the mouse is DoMouse at 627 in FIG. 6C. This function obtains the current mouse position in the

	window, erases an area in memory used for overlaying on a window, determines from the mouse position what line representations 405 were selected, checks whether any of them contains a match for the search text, and if it does, the text is **drawn at the proper position in the overlay**, thus causing it to be displayed as shown at 407 in FIG. 4." (emphasis added) (GOOGLE1005 at 10:14-24.) The VzDrawer class also has other external symbols for producing graphical images. (*Id.* at ("The remaining methods [of VzDrawer class] draw into the current object. **They draw either the frame or a filled shape** and use either the pixel-based or floating valued coordinate systems. There are therefore usually four routines for each drawing command.") (emphasis added).) Furthermore, the class "VzBarlayout" is capable of producing graphical images by "drawing either rows or columns of bars and using the mouse to manipulate them." (*Id.* at 8:40-44.) The class VzBarlayout includes external actions in the form of the inherited class "VzMouseable" that defines inputs used by the VzBarlayout object and further includes an external symbol in the form of information for producing graphical images of "either rows (horizontal) or columns (vertical) of bars." (*Id.* at 13:1-3.) The VzBarlayout class that draws bar shapes inherits the VzDrawer class. (*Id.* at 44-47.) The FloatDraw shape class inherits the DoExpose function (external symbol information defining manipulation of a shape). (*Id.* at 9:56-63 ("Objects of class FloatDraw 503 have three operations: reading a file to produce the information from which window 401 may be produced, **drawing [the] window 401**, and responding to mouse selections of line representations 405. All of these operations are specified as virtual functions 102, the read operation as the ReadFile virtual function at 617, **the drawing operation as the DoExpose function at 619**, and the response to the mouse selections at DoMouse 621.") (emphasis added); *see also* 9:27-30 ("the program may perform graphics op-

	erations on the window represented by an object of class FloatDraw 503 and will also respond to mouse inputs on the window.").)
[4] The computerized system of claim 2 wherein the external action comprises a plurality of external methods and external data.	As detailed above, the resulting combination of Eick in view of Kruglinski provides all elements of claim 2. This resulting combination would also provide that the external action comprises a plurality of external methods and external data. (GOOGLE1003 at ¶¶ 146-151.) For example, Eick's VzMouseable and VzKeyable classes define how input is received by shape classes and include external methods and external data. (GOOGLE1005 at 5:23-34 ("An example of the above is provided by the Vz library's class VzMouseable, which includes a virtual function DoMouse. The virtual function interface **defines how information concerning the behavior of the mouse is passed to code 109**, but does not define what action is taken in response to the information. A class which inherits VzMouseable must provide an ordinary function 104 for DoMouse.") (emphasis added); 5:58-60 ("the functionality class VzMouseable provides the functionality of responding to a mouse."); 7:1-2 ("VzMouseable 335 is a class which provides an area object with the ability to respond to input from a mouse."); 7:9-15 ("VzKeyable 339 is a class which provides an area object with the ability to respond to input from the keyboard."); 35:3-10 ("The functionality classes provide the objects defined by the area classes with the **capability of responding to particular inputs**. Functionality classes of particular interest in the Vz library include the VzMouseable class for making an area of the screen responsive to inputs from the mouse, the VzKeyable class for making the area responsive to inputs from the keyboard.") (emphasis added).) Eick discloses "any object which inherits any of the entity classes may inherit any of the functionality classes." (*Id.* at 34:52-54.) For example, Eick, FIG. 6A, block 613 shows class FloatDraw inherits from VzMousable. Based upon Eick's explicit teaching described herein, Eick indicates that FloatDraw could have likewise inherited from VzKeyable. (GOOGLE1003 at ¶¶ 148; 151.)

	The VzMouseable class provides a method for handling an external action, such as mouse input. (GOOGLE1005 at 7:1-8 ("The constructor **establishes which types of mouse input will cause the methods of this class to be called**, such as mouse clicking only, or clicking and dragging or all mouse motion. There are operations for getting and setting which types of mouse input will trigger a response, and there is a virtual call back function for specifying how the area object is to respond to input from the mouse.") (emphasis added).) The VzMouseable class includes a class called DoMouse that provides external data in the form of identifying which button was clicked on a mouse. (*Id.* at 29:17-26 ("virtual void DoMouse(VzMouseActions, int button, VzNativeEvent const*) Called when any one of the selected VzMouseActions occurs. The default action is to do nothing. The actual VzMouseAction and button are passed. If the VzAction does not involve a button, the button argument is zero. Additionally, a pointer to the triggering VzNativeEvent is also passed. In at least the X-Windows/OSF-Motif implementation of the Vz library, this is needed to pop-up a menu with button 3.").) The VzKeyable class additionally provides a method for handling an external action, such as keyboard input. (*Id.* at 7:9-15. ("The constructor establishes a data structure for storing keyboard state, there are added operations for getting and setting the keyboard state, and there is a virtual call back function for specifying how the area object is to respond to input from the keyboard.").) The VzKeyable class also includes external data in the form of specifying which key to watch for input. (*Id.* at 27:50-64 (describing storing "the last (not current) key that was pressed or released.").)
[6] The computerized system of claim 2 wherein the external symbol comprises a plurality of external methods and ex-	As detailed above, the resulting combination of Eick in view of Kruglinski provides all elements of claim 2. This combination also discloses that the external symbol comprises a plurality of external methods and external data.

ternal data.	(GOOGLE1003 at ¶¶ 152-157.) For example, Eick's class FloatDraw, which as addressed above includes external actions and external symbols, includes multiple methods used by the external symbol of FloatDraw to create and manipulate the graphic images created by FloatDraw. (GOOGLE1005 at 9:56-63 ("Objects of class FloatDraw 503 have three operations: reading **a file to produce the information from which window 401 may be produced, drawing [the] window 401**, and responding to mouse selections of line representations 405. All of these operations are specified as virtual functions 102, the read operation as the ReadFile virtual function at 617, the drawing operation as the DoExpose function at 619, and the response to the mouse selections at DoMouse 621.") (emphasis added).) The DoExpose function is an example of an external symbol function. For example, the DoExpose function can draw lines. (*Id.* at 10:6-13 ("The display of columns 403 and line representations 405 is produced when window 401 is exposed by the implementation of DoExpose at 625 in FIG. 6B. The function determines the current size of the window and then **uses the arrays produced by ReadFile to produce the display. A line representation 405 is drawn for each line**; if there is a match in the line, the line representation 405 is highlighted.") (emphasis added).) The DoExpose function is called any time a shape needs to be redrawn. (*Id.* at 23:15-19 ("virtual void DoExpose(int left, int top, int width, int height)=0 A pure virtual member-function. It is called when a region of the VzDrawingArea needs to be redrawn. The rectangle passed frames the region.").) The VzDrawer class also has other external symbols and associated methods for producing graphical images. (*Id.* at ("The remaining methods [of VzDrawer class] draw into the current object. They draw either the frame or a filled shape and use either the pixel-based or floating valued coordinate systems. There are therefore usually four routines for each drawing command.")

	Furthermore, the class “VzBarlayout” includes external symbols that include methods for drawing rows of bars, and methods for drawing columns of bars (*Id*. at 8:40-44 (“drawing either rows or columns of bars and using the mouse to manipulate them.”).) Other methods inherited by VzBarlayout include VzFloating (for performing bar size calculations), VzContinuousHighlighting, VzDrag and VzLeave. (*Id.* at 13:25-50.) The VzBarlayout class that draws bar shapes inherits the VzDrawer class. (*Id.* at 44-47.) The VzBarlayout class includes external data, including threshold values such as a threshold height value stored for rows of bars and a threshold width value stored for the height of bars. (*Id.* at 13-5-8.) VzBarlayout also includes external data for indicating the size of gaps between bars (VzGaps). (*Id.* at 13:22-24.) The "VzColorManager 321…class specifies the functionality needed to control the use of colors in an area of a display 113. Operations added by this class include allocating colors, mapping colors to a scale of values, overlaying colors, turning colors off and on, and establishing foreground and background color." (*Id. at* 6:48-53.) The VZDrawer class (and therefore the VzBarlayout class) has the VzColorManager capabilities. (*Id.* at 6:54-56 ("VzColorManager 321 is in turn inherited by VzDrawer 323, a functionality class which gives an area object the ability to respond to drawing commands.").)
[8.P] A computer program encoded on a computer-readable medium, the computer program operable to:	Eick discloses a computer program encoded on a computer-readable medium. (GOOGLE1003 at ¶ 159.) For example, Eick describes the VZ library, which is “[a] library of C++ classes for use in writing data visualization programs.” (GOOGLE1005 at Abstract.) More specifically, Eick explains that "Object-oriented programming system 101 includes a compiler 107 for an object-oriented programming language which produces object code 109 for CPU 111. When CPU 111 executes code 109, the execution results in outputs 123 to display 113. In response to these outputs, display 113 produces window 115." (*Id.* at 4:22-26.) A POSITA would know that the code executed by the CPU

	would be stored on a computer-readable medium. (GOOGLE1003 at ¶ 158.)
[8.1] access an external shape stored outside the computer program, the external shape comprising external capabilities; and delegate the production of a graphical image of the external shape to the external capabilities.	*See* corresponding explanations for claim 1
Claim 9	Claim 2 corresponds to claim 2. *See* explanations for claim 2.
Claim 10	Claim 10 corresponds to claim 3. *See* explanations for claim 3.
Claim 11	Claim 11 corresponds to claim 4. *See* explanations for claim 4.
Claim 13	Claim 13 corresponds to claim 6. *See* explanations for claim 6.
[15] The computer program of claim 8 wherein the external capabilities comprise a plurality of external methods and external data.	The resulting combination of Eick in view of Kruglinski provides external capabilities in the form of external methods of the external action (*see* explanation for claim 4) and external methods of the external symbol (*see* explanation for claim 6). (GOOGLE1003 at ¶¶ 146-157; 162.) Eick discloses external data in the form of external data of the external action (*see* explanation for claim 4) and external data of the external symbol (*see* explanation for claim 6). (GOOGLE1003 at ¶¶ 146-157; 162.)

X. CONCLUSION

Claims 1-4, 6, 8-11, 13, and 15 of the '633 patent are invalid over the prior art pursuant to Grounds 1-2 set forth above. Accordingly, Petitioners requests *inter partes* review of claims 1-4, 6, 8-11, 13, and 15.

Respectfully submitted,

Dated: March 24, 2014

/John C. Phillips/

John C. Phillips, Reg. No. 35,322
Michael T. Hawkins, Reg. No. 57,867
Fish & Richardson, P.C.
3200 RBC Plaza
60 South Sixth Street
Minneapolis, MN 55402
T: 858-678-4304
F: 877-769-7945

(Trial No. IPR2014-____)

Attorneys for Petitioner

CERTIFICATE OF SERVICE

Pursuant to 37 CFR §§ 42.6(e)(4)(i) *et seq.* and 42.105(b), the undersigned certifies that on March 24, 2014, a complete and entire copy of this Petition for *Inter Partes* Review and all supporting exhibits were provided by Federal Express, cost prepaid, to the Patent Owner by serving the correspondence address of record as follows:

Maschoff Brennan
1389 Center Drive, Suite 300
Park City UT 84098

/Diana Bradley/

Diana Bradley
Fish & Richardson P.C.
60 South Sixth Street, Suite 3200
Minneapolis, MN 55402
(858) 678-5667