(54) **IMAGE FILE AS CONTAINER FOR WIDGETS IN GUI AUTHORING TOOL**

BILDDATEI ALS BEHÄLTER FÜR WIDGETS EINES WERKZEUGES MIT GRAFISCHER BENUTZEROBERFLÄCHE (GUI).

FICHIER IMAGE COMME CONTENEUR DE COMPOSANTS D'INTERFACE GRAPHIQUE

(72) Inventors:
• **KLASK, Kenneth J.
San Jose
California 95118 (US)**
• **KASHYAP, Manohar
Fremont
California 94539 (US)**

(74) Representative: **Exell, Jonathan Mark
Williams Powell
Staple Court
11 Staple Inn Buildings
London
WC1V 7QH (GB)**

(56) References cited:
**US-A1- 2006 005 207   US-A1- 2006 015 818
US-A1- 2008 235 602**

• **None**

EP 2 539 810 B1

Description

FIELD OF INVENTION

[0001]   This invention relates to graphical user interface authoring tools.

DESCRIPTION OF RELATED ART

[0002]   A What-You-See-Is-What-You-Get (WYSI-WYG) authoring tool for graphical user interface (GUI) offers a palette of GUI widgets that the designer can select from, visually arrange, and manipulate to assemble a GUI. The palette can either be fixed or extendable. Often, palette extensions are done by the designer using a conventional programming language such as C or Java. The palette extensions are usually intended for use by an individual customer and are not easily shared with other users of the authoring tool. More advanced authoring environments have a plug-in architecture that allows developers to build reusable GUI widgets that can be distributed to other users so that they can add the GUI widgets to their palettes and use them just like the standard palette. GUI widgets are often a collection of source code and graphics (vector, bitmap, or both). GUI widgets are either distributed as a single file in a proprietary format or as a collection of files. When distributed as a single file, the proprietary nature of the file may prevent an operating system (OS) file explorer from providing a thumbnail of the GUI widget and a standard graphical editing tool from modifying the graphics in the file. When distributed as a collection of files, the GUI widget is difficult to manage. US2006005207 discloses a widget authoring and editing environment.

STATEMENT OF INVENTION

[0003]   The invention is defined by independent method claim 1; dependent claims define preferred embodiments.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004]   In the drawings:

Fig. 1 illustrates a hardware-software environment for implementing a graphical user interface (GUI) for a host device;

Fig. 2 illustrates a single image file of a standard format containing a widget in one or more embodiments of the present disclosure;

Fig. 3 illustrates an implementation of a portable network graphics (PNG) file containing a widget in one or more embodiments of the present disclosure;

Fig. 4 illustrates XML (Extensible Markup Language)

properties of a radio button widget in a tEXt chunk of the PNG file of Fig. 3 in one or more embodiments of the present disclosure;

Fig. 5 illustrates a What-You-See-Is-What-You-Get (WYSIWYG) authoring tool for GUI in one or more embodiments of the present disclosure;

Fig. 6 illustrates a user interface for a widget builder of Fig. 5 in one or more embodiments of the present disclosure;

Fig. 7 is a flowchart of a method for a processor to implement the widget builder of Fig. 5 in one or more embodiments of the present disclosure;

Fig. 8 illustrates a user interface for a GUI builder of Fig. 5 in one or more embodiments of the present disclosure;

Fig. 9 is a flowchart of a method for a processor to implement the GUI builder of Fig. 5 in one or more embodiments of the present disclosure;

Figs. 10A and 10B illustrate an XML project file created from user input by the GUI builder of Fig. 5 in one or more embodiments of the present disclosure;

Fig. 11 illustrates state images in a customized widget file specified in the XML project file of Figs. 10A and 10B in one or more embodiments of the present disclosure;

Fig. 12 is a flowchart of a method for the GUI builder of Fig. 5 to program a GUI into a controller chip of Fig. 1 or run the GUI in a simulator of the controller chip in one or more embodiments of the present disclosure;

Fig. 13 illustrates XML properties of a slider widget in one or more embodiments of the present disclosure; and

Fig. 14 illustrates XML properties of a list widget in one or more embodiments of the present disclosure.

[0005]   Use of the same reference numbers in different figures indicates similar or identical elements.

DETAILED DESCRIPTION

[0006]   Fig. 1 illustrates a hardware-software environment 100 for implementing a graphical user interface (GUI) for a host device 102. A user 104 uses a hypertext markup language (HTML) authoring tool 106 to design the GUI like a website by laying out text, images, animations, and hypertext links on one or more HTML pages. User 104 may add non-HTML GUI widgets to the HTML

pages. The widgets are entered into the HTML pages using Java applet tags so a drag-and-drop HTML authoring tool 106 will accept widget properties nested as elements between the applet tags but the widgets are not written in Java.

[0007] The widgets in the HTML pages may be created from template widgets 108. Each template widget 108 includes an applet file 108-1 and optionally includes any number of bitmap image files 108-2 that form part of the widget. Applet file 108-1 includes the widget properties nested between applet tags. The widget properties specify a widget type, widget parameters for the widget, and one or more function calls to jump to another HTML page or for the widget to communicate data with host device 102, other widgets, or an input device 110 upon an event. The widget properties optionally include any number of properties for vector graphics that form part of the widget.

[0008] Once the GUI is constructed, user 104 runs a compiler 112 to compile the HTML pages and the linked images into μHTML code 114 to reduce size and execution requirements. User 104 loads μHTML code 114 into a nonvolatile memory 116 (e.g., a flash memory). Nonvolatile memory 116 also stores a μHTML operating system (OS) 118 and a widget library 120. A controller chip 122 includes a processor that runs μHTML OS 118, which executes μHTML code 114 to render the current GUI screen to a frame buffer in a volatile memory 124 (e.g., a random access memory). In particular, μHTML OS 118 accesses widget library 120 for the drawing methods and the behaviors of the widgets. Controller chip 122 also includes a display controller that raster scans the GUI screen in the frame buffer to a display 126. Controller chip 122 communicates user input with input device 110, such as a touch screen. Controller chip 122 also communicates data and user input with an embedded processor in host device 102.

[0009] For more details, see U.S. Patent No. 7,100,118 and U.S. Patent Application Serial No. 11/502,071 . Controller chip 122 may be the Amulet Graphical OS Chip, and μHTML OS 118 and widget library 120 may be the Amulet OS, both available from Amulet Technologies of Campbell, California. The syntax of the HTML and the widget properties are found in the Programmer's Guide for the Amulet Graphical OS Chip, which is available at the website of Amulet Technologies, LLC of Campbell, California.

[0010] In accordance with the present disclosure, a single widget file is used as the container for all the elements of a widget. The widget file conforms to a standard image file format that allows commenting (text that is ignored by compilers and interpreters). Widget properties are stored as comments in the widget file as specified by the image file format. When vector graphics are part of the widget, their properties are stored as part of the widget properties in the comments. When bitmap images are part of the widget, they are stored as images in the widget file as specified by the image file format. The standard image format is supported by conventional operating systems so an OS file explorer can provide a thumbnail of the widget file and a standard editing tool can modify the bitmaps in the widget file. As a single file, the widget is easy to distribute between GUI developers.

[0011] Fig. 2 illustrates a file 200 for a widget in one or more embodiments of the present disclosure. The widget is either a template widget or a customized widget created from a template widget. Widget file 200 conforms to a standard image file format. Widget properties are stored as comments 202 in widget file 200 as specified by the image file format. The widget properties may specify a widget type, widget parameters for the widget, and one or more function calls. Based on an event, a function call may cause a jump to another GUI page or the widget to communicate data with host device 102, other widgets, or input device 110. A widget may include any number of vector graphics and bitmap images. When vector graphics are present, their properties are stored as part of the widget properties in comments 202. When bitmap images 204-1 to 204-i are present, they are stored as images in widget file 200 as specified by the image file format. When a widget does not include any bitmap images, a representative thumbnail image may be generated and saved as an image in widget file 200 as specified by the image file format.

[0012] Fig. 3 illustrates an implementation of a widget file 300 conforming to the portable network graphics (PNG) file format in one or more embodiments of the present disclosure. Widget file 300 includes a PNG signature chunk 302 containing a file header. PNG signature chunk 302 is followed by one or more groups of chunks. A first group 304-1 includes an IHDR chunk 306 containing an image header, a tEXt chunk 308 containing comments, an IDAT chunk 310 containing image data, and an IEND chunk 312 marking the end of the image. Widget properties are stored as comments in tEXt chunk 308. When one or more vector graphics are part of the widget, their properties are stored as part of the widget properties in tEXt chunk 308. When one or more bitmap images are part of the widget, the first bitmap image is stored in IDAT chunk 310 of first group 304-1.

[0013] First group 304-1 is optionally followed by groups 304-2 to 304-i, which each includes an IHDR chunk 306, an IDAT chunk 310, and an IEND chunk 312. When additional bitmap images are present, they are stored in IDAT chunks 310 of groups 304-2 to 304-i. When a widget does not include any bitmap image, a representative thumbnail image may be generated and saved in IDAT chunk 310 in first group 304-1. When one or more non-PNG images are present, they may be stored in one or more fiLe chunks 314, which is a new type of ancillary, private, and safe to copy chunk. The fiLe chunks 314 may also be used to store PNG images if desired. An fiLe chunk 314 has a length field, a chunk type (i.e., fiLe), the data field, and a cyclic redundancy check (CRC) calculated on the preceding bytes in the chunk but not including the length field. The data field includes a byte count of an image format description, the

image format descriptor (e.g., JPEG), and the non-PNG image data.

[0014] Fig. 4 illustrates widget properties 400 implemented in XML (Extensible Markup Language) in one or more embodiments of the present disclosure. Widget properties 400 for a template radio button widget are provided as an example. Widget properties for all widget types follow the same syntax but contains information specific to the widgets. Widget properties such as widget types, widget parameters for the various widget types, and function calls for the various widget types are explained in the Programmer's Guide for the Amulet Graphical OS Chip, which is available at the website of Amulet Technologies, LLC of Campbell, California. For illustrative purposes, widget properties of a slider widget and a list widget are provided in the Appendix of the present disclosure. Additional widget types include a checkbox widget, a button widget, a spinner widget, and a menu widget. This is a non-exhaustive list of widget types and future widget types may be developed.

[0015] Widget properties 400 begin with a text declaration at line 402. Line 404 includes a start tag for a template element. When present, the template element identifies the widget as a template widget for creating customized widgets. Line 406 includes a start tag for a widget type element nested within the template element. The widget type is "RadioButton," which identifies the widget as a radio button widget. Nested in the widget type element are widget parameter elements for the widget.

[0016] The widget parameter elements each include start and end tags, a widget parameter name and one or more attributes in the start tag, and one or more values for the widget parameter between the tags. When a start tag includes an attribute of "Editable="True"," the widget parameter may be modified by the user when the user creates a widget from the template widget and vice versa. When a start tag includes an attribute of "Optional="False"," the widget parameter is a parameter that is automatically included in a properties editor 602 (described later with Fig. 6) in a widget builder 502 (described later with Fig. 5) when the user creates a customized widget from a template widget. When a start tag includes an attribute of "Optional="True"," the widget parameter is listed as one of the parameters in a popup menu 608 (described later with Fig. 6) used to add a widget parameter to properties editor 602. When a start tag includes an attribute of "Ignore="True"," the widget is created without a widget parameter in a GUI builder 606 (described later with Fig. 6).

[0017] Line 408 includes a "Name" element having a value that specifies a name (e.g., "RBI") for the radio button widget. Line 410 includes a "label" element having a value that specifies a label (e.g., "Button Label") appearing with the radio button widget. Lines 412 and 414 include an "initialCondition" element having values that specify an initial condition for the radio button widget when the UI is initially loaded and choices for the initial condition. For example, the values for the initialCondition

element are "Off On Off," where the first "Off specifies the set initial condition, the "On Off" thereafter specifies the choices for the initial conditions, and the values are separated by a tab delimiter (not visible in the XML text). Line 416 includes a "groupName" element having a value that specifies a group (e.g., "rbgl") that this radio button widget is part of. Line 418 includes a "href element having a value that specifies a function (e.g., a placeholder function "Amule:Nop") to be invoked when the radio button widget is set.

[0018] Line 420 includes a "Width" element having a value that sets a width (e.g., 164 pixels) for the radio button widget. Line 422 includes a "Height" element having a value that sets a height (e.g., 34 pixels) for the radio button widget. Lines 424 and 426 include a "fontStyle" element having one or more values that specify one or more font styles (e.g., plain, bold and italic) for the label. Line 428 includes a "font" element having values that specify the font and the font size (e.g., Amulet Sans Serif, 12 pt) of the label.

[0019] Lines 430, 432, and 434 include elements that identify images to be used for the widget in various states (also referred to as "state images"). Line 430 includes an "emptyImage" element having one or more values that specify one image or multiple images (also referred to as "image components") that make up the empty state image of the radio button widget when the widget is not set. When image components are specified for a state image, the image components are layered over each other to create the state image. Line 432 includes a "fullImage" element having one or more values that specify one image or multiple image components that make up the full state image of the radio button widget when the widget is set. Line 434 includes a "TrackingImage" element having one or more values that specify one image or multiple image components that make up the tracking state image of the radio button widget when the widget is being touched in a touch screen or when a cursor is placed over the widget. Line 436 is the end tag for the widget type "RadioButton", which has a corresponding start tag on line 406. Line 438 is the end tag for the template element "Template," which has a corresponding start tag on line 404.

[0020] Each start tag for a widget parameter element includes an attribute of "Type=" that specifies the data type of the widget parameter. A popup data type 10 is a popup menu in which the user can select only one option for a widget parameter. For example, popup data type 10 is used for the initialCondition parameter in line 412. As described above, the values for the initialCondition element are "Off On Off," where the first "Off" specifies the set initial condition, the "On Off" thereafter specifies the choices for the initial conditions, and the values are separated by tab delimiters (a nonprinting character).

[0021] A color data type 11 specifies that a widget parameter value is a color. For example, color data type 11 is used to describe widget parameters such as text color.

[0022] A string data type 12 specifies that a widget pa-

rameter value is a string value. For example, string data type 12 is used to describe widget parameters such as name and label.

[0023] A number data type 13 specifies a widget parameter value as a number. For example, number data type 13 is used to describe widget parameters such as width, height, minimum values, and maximum values.

[0024] A font data type 14 specifies a widget parameter value as a font and font size. For example, font data type 14 is used to describe widget parameters such as font type.

[0025] An IWCMethod data type 15 specifies widget parameter values as one or more functions used in a widget and one or more events that trigger the function call. Some of the functions include jumping to another GUI page, sending values out, setting values, halting current widget activity, and saving current status to memory. For example, a href element in the radio button widget may use the IWCMethod data type to toggle the polarity of a display between positive and negative.

[0026] A file data type 16 specifies a widget parameter value as a bitmap file external of the widget filet. For example, the Empty Image parameter of the radio button widget uses file data type 16 to point to a file which contains the empty state image. When the radio button widget is not set, this image will show on the GUI.

[0027] A Boolean data type 17 specifies a widget parameter value that has two choices. For example, in a line graph widget, a column clear parameter may be classified as Boolean because it has two choices: TRUE or FALSE.

[0028] A MultiSelectionPopup data type 18 has a pop-up menu that shows different options the user may choose. This is different than popup data type 10 because the user may select more than one option simultaneously. For this data type, the widget parameter values include the selected choices separated by "|" characters, the selected choices are separated by a tab delimiter (a non-printing character) from the choices for the widget parameter element, and the choices are separated by tab delimiters. For example, a font element having MultiSelectionPopup data type 18 allows the user to select font size, font color, and font style at the same time.

[0029] An EmbeddedImage data type 19 specifies one or more widget parameter values as one PNG image or multiple PNG image components saved in the widget file. When multiple PNG image components are specified, the image components are grouped to form a single image. For example, image A, B, and C can be grouped as a down button image and image B, D and E can be grouped as an up button.

[0030] An UnsignedFloatingPoint type 20 specifies one or more values as unsigned floating point numbers. For example, an Update Rate widget parameter specifies how often a widget calls a href function. The first unsigned floating point number specifies the href function call frequency (specified in seconds, with a single floating-point number). The range is 0.00 - 655.35 where a value of

0.00 means update never. The second number specifies the delay time from when a GUI page is loaded until the first href function call (specified in seconds, with a single floating-point number). The range is 0.01 - 655.35. If the second number is not specified, then the delay time defaults to the first number (frequency).

[0031] An ImageArray type 21 specifies an array of images used in the widget. Images may be non-PNG. The images can be saved in the widget file or a separate file in itself.

[0032] Fig. 5 illustrates a What-You-See-Is-What-You-Get (WYSIWYG) authoring tool 500 for GUI in one or more embodiments of the present disclosure. Authoring tool 500 is executed by a computer 501 and may be stored on a computer readable storage medium in the computer.

[0033] Authoring tool 500 includes a widget builder 502 for building widgets 504, a GUI builder 506 for building a GUI using the template widgets, a runtime rendering tool 508 for drawing GUI components, including widgets, as they would appear in a native environment with controller chip 122 (Fig. 1), compiler 112 for compiling the GUI for programming the controller chip or running on a simulator 509 of the controller chip. Widgets 504 may be template widgets or customized widgets created from the template widgets. GUI builder 506 saves the GUI in an XML project file. The XML project file includes one or more XML pages having GUI components such as texts, widgets, images, or background images. Widget builder 502 and GUI builder 506 may be part of the same software running on a non-native host environment, such as a Windows operating system executed by an x86 processor on computer 501. Widget builder 502 may also be a plug-in for a conventional graphics editor.

[0034] Runtime rendering tool 508 includes a translator 510 and a library wrapper 512 around the runtime code of widget library 120. Library wrapper 512 allows widget library 120 to run as a library on a non-native platform, such as a Windows operating system, and provides an application programming interface (API) to the widget library. Library wrapper 512 may be implemented as a dynamic link library in the Windows operating system.

[0035] To generate a widget as it would appear in a native environment with controller chip 122 (Fig. 1), GUI builder 506 outputs XML widget properties of the widget to translator 510. Translator 510 translates the XML widget properties from GUI builder 506 into data structures that can be passed to the runtime code of widget library 120 via the API defined by library wrapper 512. The runtime code of widget library 120 renders the graphic representation of each widget as described by values contained in the data structures passed by library wrapper 512. The rendered image of the widget is returned to GUI builder 506. Widget builder 502 may operate similarly to provide a view of the widget as it would appear in the native environment.

[0036] To compile the GUI for programming controller chip 122 (Fig. 1) or running on simulator 509 of the con-

troller chip, GUI builder 506 outputs the XML project file to translator 510. Translator 510 translates the project file into an HTML document, including the applet codes for the widgets, which compiler 112 uses to generate μHTML document that can be executed by controller chip 122. The μHTML document may be programmed into controller chip 122 or ran in simulator 509 of the controller chip. Simulator 509 includes an application wrapper 516 around the runtime code of widget library 120. Application wrapper 512 allows widget library 120 to run as an application on a non-native platform, such as the Windows operating system.

[0037] Fig. 6 illustrates an exemplary user interface 600 of widget builder 502 (Fig. 5) in one or more embodiments of the present disclosure. Fig. 7 is a flowchart of a method 700 for a processor to implement widget builder 502 with user interface 600 (Fig. 6) in one or more embodiments of the present disclosure. Method 700 may include one or more operations, functions or actions as illustrated by one or more of blocks 702, 704, 706, 708, and710. Although the blocks are illustrated in a sequential order, these blocks may also be performed in parallel, and/or in a different order than those described herein. Also, the various blocks may be combined into fewer blocks, divided into additional blocks, and/or eliminated based upon the desired implementation.

[0038] In block 702, widget builder 502 (Fig. 5) provides user interface 600 (Fig. 6) for the user to create a customized widget from a template widget or create a new template widget. Template widgets are used in GUI builder 506 (Fig. 5) to create widgets in a GUI.

[0039] In block 704, widget builder 502 (Fig. 5) determines if the user wishes to create a customized widget from a template widget or create a new template widget. The user wishes to create a customized widget from a template widget when the user opens an existing template widget file by double-clicking the template widget file, by dragging-and-dropping the widget file into user interface 600 (Fig. 6) or a user interface 800 (Fig. 8) for a GUI builder 506 (Fig. 5), by selecting the widget file through a file menu in user interface 600 (not shown) or user interface 800. If so, block 704 is followed by block 706. The user wishes to create a new template widget when the user selects to create a new template widget through the file menu. If so, block 704 is followed by block 712.

[0040] In block 706, widget builder 502 (Fig. 5) populates a properties editor 602 (Fig. 6) with widget properties from the existing template widget file. Referring to Fig. 6, properties editor 602 includes a property column for the widget parameter names, a value column for the widget parameter values, and a "static" column for making widget parameter values user-editable or not in GUI builder 506 (Fig. 5). An image preview window 604 may provide a list of images that may be previewed and a preview panel of a selected image in the list. To add an image to the preview list, the user selects an add button 612 to select the image. To view an image in the preview

list, the user selects a corresponding check box next the image in the preview list and the image is displayed in the preview panel. To delete an image from the preview list, the user selects a delete button 614. Alternatively image preview window 604 provides a preview of a default state image or a selected state image from properties editor 602. Block 706 is followed by block 708.

[0041] Referring to Fig. 7, in block 708, widget builder 502 (Fig. 5) receives user input for the widget properties. Referring to Fig. 6, to add an additional widget parameter, the user selects an add button 606 to bring up a popup menu 608 of appropriate widget parameters for the widget type. As described before, widget parameter automatically appears in properties editor 602 when it is not optional and it appears in popup menu 608 when it is optional. To edit a widget parameter, the user selects the widget parameter and enters one or more value or selects one or more values from a popup menu. The user may make a widget parameter non-editable in GUI builder 506 (Fig. 5) by checking the static box for the widget parameter. To delete a widget parameter, the user selects the widget parameter and then a delete button 610. The user provides default values for non-editable widget parameters and optionally provides default values for user-editable widget parameters. Any widget parameter that is not provided by the user receives default values predetermined for the widget type.

[0042] To select a state image, the user selects the "+" box in the value column of the state image to bring up a file explorer and selects an image or multiple image components as the state image. When multiple image components are selected, they are superimposed over each other to form the state image. Alternatively the user may drag-and-drop one image or multiple image components from the preview list in image preview window 604 into the + box to form the state image. Block 708 is followed by block 710.

[0043] Referring to Fig. 7, in block 710, widget builder 502 (Fig. 5) saves the widget properties and any embedded images in a customized widget file as described above regarding Figs. 2 and 3. The widget properties of the customized widget follow the XML syntax of the template widget as described above regarding Fig. 4 but without the template element. Depending on the widget type, widget builder 502 may include properties for vector graphics in the widget properties. When the template widget does not include any bitmap image, widget builder 502 may generate a representative thumbnail image and save it in the widget file. The customized widget file and any external image files may be saved with other customized widget files in a widget folder known to GUI builder 506 (Fig. 5).

[0044] In block 712, widget builder 502 (Fig. 5) receives user input of the widget type for the new template widget. Widget builder 502 may generate a popup menu from which the user selects a widget type. Block 712 is followed by block 714.

[0045] In block 714, widget builder 502 (Fig. 5) receives

user input for the widget properties. Block 714 is similar to block 708. Depending on the widget type, some widget parameters are optional and some are not optional. Block 714 is followed by block 716.

[0046] In block 716, widget builder 502 (Fig. 5) saves the widget properties in a new template widget file as described above regarding Figs. 2 and 3. The widget properties of the customized widget follow the XML syntax of the template widget as described above regarding Fig. 4. The above described blocks of method 700 may be repeated to create additional customized and template widgets.

[0047] Fig. 8 illustrates an exemplary screenshot of a user interface 800 of GUI builder 506 (Fig. 5) in one or more embodiments of the present disclosure. Fig. 9 is a flowchart of a method 900 for a processor to implement GUI builder 506 with interface 800 in one or more embodiments of the present disclosure. Method 900 may include one or more operations, functions or actions as illustrated by one or more of blocks 902, 904, 906, 908, 910, 912, 914, 916, 918, 918, 920, 922, 924, 926, 928, 930, and 932. Although the blocks are illustrated in a sequential order, these blocks may also be performed in parallel, and/or in a different order than those described herein. Also, the various blocks may be combined into fewer blocks, divided into additional blocks, and/or eliminated based upon the desired implementation.

[0048] In step 902, GUI builder 506 (Fig. 5) provides user interface 800 (Fig. 8) for the user to build a GUI. Referring to Fig. 8, user interface 800 allows the user start a new GUI page or to add/delete GUI components including text, widget, image, and background image to/from a layout canvas 802 of the current GUI page. In response to the user selecting an add button 804, GUI builder 506 provides a popup menu 806 for the user to add a GUI page, a text, a widget (listed as "controls"), an image, or a background image. The GUI components are listed in a properties editor 808 with their property names and values. User deletes a GUI component by selecting the GUI component from properties editor 808 and then a delete button 810. Once the user adds a GUI component, the user may arrange it in layout canvas 802 and edit its properties in properties editor 808. GUI builder 506 saves the GUI component as part of the GUI in an XML project file 1000 (Figs. 10A and 10B). XML project file 1000 includes one or more XML pages with XML code for the GUI components. The widgets generally follow the XML syntax of the customized widgets. Referring to Fig. 9, step 902 is followed by step 904.

[0049] In step 904, GUI builder 506 (Fig. 5) receives user input to add a widget, which is an instance of a customized widget. The user may create a widget by selecting "controls" in popup menu 806 (Fig. 8), which brings up a file explorer for selecting a customized widget file. The user may also drag-and-drop a customized widget file into layout canvas 802 (Fig. 8) or right-click another widget on the layout canvas and selecting a duplicate option. When a template widget file is opened in GUI builder 506, widget builder 502 is started to create a customized widget from the template widget. Step 904 is followed by step 906.

[0050] In step 906, GUI builder 506 (Fig. 5) parses the selected customized widget file for the comments to extract the widget properties. Step 906 is followed by step 908.

[0051] In step 908, GUI builder 506 (Fig. 5) populates properties editor 806 (Fig. 8) with the user editable parameter names and their values of the selected customized widget. Properties editor 806 includes a property column for the widget parameter names and a value column for the widget parameter values in an expandable file structure. Step 908 is followed by step 910.

[0052] In step 910, GUI builder 506 (Fig. 5) converts the images of the customized widgets, which may be in different image formats, into objects of a common class and creates pointers to the image objects. Step 910 is followed by step 912.

[0053] In step 912, GUI builder 506 (Fig. 5) creates an instance of the customized widget in memory with the widget type, the widget parameters, and the pointers to the image objects in memory. Step 912 is followed by step 914.

[0054] In step 914, GUI builder 506 (Fig. 5) calls runtime rendering tool 508 (Fig. 5) to draw and return a rendered image of the widget. GUI builder 506 provides runtime rendering tool 508 with the XML widget properties and pointers to the image objects.

[0055] Referring to Fig. 5, runtime rendering tool 508 includes translator 510 and a library wrapper 512 around the runtime code of widget library 120. Using the pointers to the image objects, translator 510 determines the size of the state image of the widget in its initial condition (default state image), and then translates the XML widget properties and the size of the default state image into data structures that can be passed to the runtime code of widget library 120 via the API defined by library wrapper 512. The runtime code of widget library 120 renders the graphic representation of the widget as described by the values contained in the data structures passed by the library wrapper 512. The rendered image of the widget is returned to GUI builder 506. Note that the runtime code of widget library 120 may not need to render a bitmap default state image since it is a bitmap. Instead, the runtime code of widget library 120 returns the appropriate size and the location of the default state image relative to the other graphical elements rendered by the widget library, and GUI builder 506 constructs the final image of the widget accordingly. Alternatively, the runtime code of widget library 120 may read the default state image and construct the final image of the widget in its entirety. Step 914 is followed by step 916.

[0056] Referring back to Fig. 9, in step 916, GUI builder 506 (Fig. 5) displays the final image of the widget on layout canvas 802 (Fig. 8). GUI builder 506 tracks the layering of the various GUI components so that it knows if the widget is displayed in front or behind other GUI

components. The user may adjust the layering of the GUI components using buttons 812. The user may also adjust the alignment of the GUI components using buttons 813. Step 916 is followed by step 918.

[0057] In step 918, GUI builder 506 (Fig. 5) determines if the user is repositioning the widget on layout canvas 802 (Fig. 8) or resizing the widget. If so, step 918 is followed by step 920. Otherwise step 918 is followed by step 922.

[0058] In step 920, GUI builder 506 (Fig. 5) displays the rendered image of the widget at a new location on layout canvas 904 or at a new size according to user input. Step 920 loops back to step 918.

[0059] In step 922, GUI builder 506 (Fig.5) determines if the user is changing an editable widget property. To edit a widget property, the user selects the widget property and enters one or more value or selects one or more values from a popup menu. When the user is changing an editable widget property, step 922 is followed by step 924. Otherwise step 922 is followed by step 926.

[0060] In step 924, GUI builder 506 (Fig. 5) updates the instance of the widget template in memory with changes made by the user. Step 924 is followed by step 914, where GUI builder 506 calls runtime rendering tool 508 (Fig.5) to redraw and return a new rendered image of the widget based on the changes made by the user. Note that the rendered image of the widget is updated in real-time as the user makes changes (e.g., as the user changes a property).

[0061] In step 926, GUI builder 506 (Fig. 5) determines if the user is selecting another widget in layout canvas 802 (Fig. 8). If so, step 926 is followed by step 918. Otherwise step 926 is followed by step 928.

[0062] In step 928, GUI builder 506 (Fig. 5) determines if the user wishes to add another widget. If so, step 928 is followed by step 904. Otherwise step 928 is followed by step 930.

[0063] In step 930, GUI builder 506 (Fig.5) determines if the user wishes to compile or run the GUI. The user may compile or run the GUI by selecting a program button 814 or a run button 816 (Fig. 8). If so, step 930 is followed by step 932. Otherwise step 930 is followed by step 918 where GUI builder 506 continues to monitor for user input modifying the GUI. Steps 904 to 928 may be repeated for additional GUI pages. The process for adding texts, images, and background images are not described as they are similar to authoring an HTML page.

[0064] In step 932, GUI builder 506 (Fig. 5) outputs the XML project file of the GUI to translator 510 (Fig. 5), which outputs an HTML document to compiler 112, which generate the μHTML code for programming controller chip 122 (Fig. 2) or running in simulator 509 (Fig. 5) of the controller chip

[0065] Figs. 10A and 10B illustrate an XML project file 1000 created with GUI builder 506 (Fig. 5) in one or more embodiments of the present disclosure. XML project file 1000 begins with a text declaration at line 1001. Line 1002 includes a start tag for a project element having

attributes that identify GUI builder and compiler versions. Nested in the project file element are GUI page elements, and nested in the GUI page elements are GUI component elements. Although not shown, hardware elements may be included to specify the hardware parameters for the GUI (e.g., type and size of display).

[0066] Line 1003 includes a start tag for a GUI page element having a "NAME" attribute of "Page_1." Lines 1004 to 1009 include overall properties for the GUI page, such as transition effects for the GUI page. Lines 1010 to 1021 include properties for a background image. Specifically, line 1014 shows the path to a background image file "fringe.png," and line 1015 shows that the image is not to be cached in volatile memory (e.g., RAM 124 in Fig. 1) but read from nonvolatile memory (e.g., flash 116 in Fig. 1).

[0067] Lines 1022 to 1051 include properties for the body of the GUI page. Lines 1023 to 1033 include properties for a radio button widget. The start tag of the radio button widget includes an "imageSource" attribute with the path to the customized widget file. State images are not listed in these lines because they are specified in the customized widget file. Fig. 11 shows state images 1102, 1104 and 1006 in the customized widget file in one or more embodiments of the present disclosure.

[0068] Referring back to Figs. 10A and 10B, lines 1034 to 1050 include properties for a function button widget. The start tag of the function button does not include an "imageSource" attribute because it is a standard widget having its entire graphics, either bitmap or vector, provided in widget library 120. Line 1040 includes a function call to jump to a GUI page named "Page_2" when the function button is set.

[0069] Line 1051 includes an end tag of the GUI page "Page_1." Line 1052 is the start tag of GUI page "Page_2." Lines 1053 to 1073 include the GUI components on the GUI page. Line 1074 includes an end tag of the GUI page followed by line 1075 with an end tag of the project.

[0070] Fig. 12 is a flowchart of a method 1200 for GUI editor 506 (Fig.5) to output the XML project file of the GUI in step 932 (Fig. 9) in one or more embodiments of the present disclosure. Method 1200 may include one or more operations, functions or actions as illustrated by one or more of blocks 1202, 1204, 1206, 1208, 1210, 1212, 1214, and 1216. Although the blocks are illustrated in a sequential order, these blocks may also be performed in parallel, and/or in a different order than those described herein. Also, the various blocks may be combined into fewer blocks, divided into additional blocks, and/or eliminated based upon the desired implementation.

[0071] In step 1202, GUI editor 506 (Fig.5) selects one of the one or more GUI XML pages from the project file. Step 1202 is followed by step 1204.

[0072] In step 1204, GUI editor 506 (Fig.5) selects one of the one or more widgets on the GUI XML page. Step 1204 is followed by 1206.

[0073] In step 1206, GUI editor 506 (Fig.5) parses

widget properties of the widget from the XML page and loads them into memory. Step 1206 is followed by step 1208.

[0074] In step 1208, GUI editor 506 (Fig.5) extracts any embedded images and retrieves any external images of the widget and saves them as individual image files in a HTML temporary folder. Step 1208 is followed by step 1210.

[0075] In step 1210, GUI editor 506 (Fig.5) calls translator 510 (Fig.5) to translate the widget properties into the HTML syntax understood by compiler 112 (Fig. 1). Step 1210 is followed by step 1211.

[0076] In step 1211, GUI editor 506 (Fig. 5) calls compiler 112 (Fig. 5) to compile the HTML code into μHTML code. Block 1211 is followed by block 1212.

[0077] In step 1212, GUI editor 506 (Fig. 5) determines if there is any remaining widget on the current GUI screen it has not processed. If so, step 1212 is followed by step 1204. Otherwise step 1212 is followed by step 1214.

[0078] In step 1214, GUI editor 506 (Fig. 5) determines if there is any remaining GUI page it has not processed. If so, step 1214 is followed by step 1202. Otherwise step 1214 is followed by step 1216.

[0079] In block 1216, GUI editor 506 (Fig.5) waits for compiler 112 (Fig. 5) to finish compiling. Block 1216 is followed by block 1218.

[0080] In block 1218, GUI editor 506 (Fig. 5) programs controller chip 122 (Fig. 1) with the μHTML or calls simulator 509 of the controller chip to run the μHTML code.

[0081] Various other adaptations and combinations of features of the embodiments disclosed are within the scope of the invention. Instead of a PNG file, a graphics interchange format (GIF) file may be used to container for the widget. In the GIF file, the image components are saved in graphic rendering blocks and the widget source code is saved as comments in a comment extension block. Numerous embodiments are encompassed by the following claims.

APPENDIX

Slider Parameter Types

[0082] Fig. 13 lists the parameters that describe the Slide widget in one or more embodiments of the present disclosure.

Name

[0083] VALUE="text" — Specifies the name Slider widget.

Width

[0084] VALUE="number" — Specifies the width of the Slider widget.

Height

[0085] VALUE="number" — Specifies the height of the Slider widget.

channelColor

[0086] VALUE= "color" — Specifies the color of the channel the handle slides along If no channel color is specified, the default color is black.

handleColor

[0087] VALUE="color" — Specifies the color inside of the handle. If no handle color is specified, the default color is white.

handleFrameColor

[0088] VALUE="color" — Specifies the color of the handle frame. If no handle frame color is specified, the default color is black.

handleTrackingColor

[0089] VALUE="color" — Specifies the color of the handle when being touched. If no handle frame color is specified, the default color is black.

Href

[0090] VALUE="function(s)"- The function (or multiple/sequenced functions) invoked upon the event specified in hrefEvent.

hrefEvent

[0091] VALUE="onChange"|"onPenUp"- The event which triggers the launching of the href function. If set to onChange, the slider will launch the function call whenever the handle is moved. If set to onPenUp, the slider will only launch the function call upon the releasing of the slider handle. If hrefEvent is not specified, the default is onChange.

initialCondition

[0092] VALUE="number"|"FromInitHre '- Specifies handle position when the page is loaded. The position value becomes the argument given to the href function(s). If FromInitHref is selected, the function specified by the InitHref attribute is called. The returned value determines the handle position. The range is 0 - 65535 (0x00 - OxFFFF).

max

[0093] VALUE="number"- The maximum value used

as the argument given to the function(s) specified in href. Maximum value is achieved when handle is full-right on a horizontal slider, or full-top on a vertical slider. The range is 1 - 65535 (0x01 - OxFFFF).

min

[0094] VALUE= "number"- The minimum value used as the argument given to the function(s) specified in href. Minimum value is achieved when handle is full-left on a horizontal slider, or full-bottom on a vertical slider. The range is 0 - 65534 (0x00 - OxFFFE).

minAt

[0095] VALUE="LEFT"|"RIGHT" |"TOP"|"BOTTOM"- Determines where the minimum value of the slider is located. As the handle sweeps from the minAt location, the value increases until the maximum value is reached at the opposite extreme of the minAt location. Default values are LEFT for horizontal sliders and BOTTOM for vertical sliders. The options are:

LEFT- The slider value increases from left to right, (horizontal slider only)

RIGHT- The slider value increases from right to left, (horizontal slider only)

TOP- The slider value increases from top to bottom, (vertical slider only)

BOTTOM- The slider value increases from bottom to top. (vertical slider only)

tickColor

[0096] VALUE="color"- Specifies the color of the ticks drawn on the channel. If no tick color is specified, the default color is black.

tickCount

[0097] VALUE="number"|"NONE"|"AUTO"- The total number of tick marks along the slider channel. If NONE, no tick marks are visible. If AUTO, the number of tick marks is derived from the min and max values. If tickCount is not given, the default is NONE. The range is 0 - 255 (0x00 - OxFF).

handleOffset

[0098] VALUE= "number"- Specifies the number of pixels from the center of the channel the handle is located. If a vertical slider, positive numbers shift the handle to the right and negative numbers shift it to the left. If a horizontal slider, positive numbers shift the handle to the bottom and negative numbers shift it to the top. The range

is -100 through 100

handleThickness

[0099] VALUE="number"- The width (in pixels) of the handle if a horizontal slider, or the height of the handle if a vertical slider. If handleThickness is not given, the default is 11 pixels. The range is 4 - 255 (0x04 - OxFF).

hrefOnHitOnly

[0100] VALUE="TRUE"1"FALSE"- Specifies if the slider href function(s) is launched only upon a "hit" or not. By default, hrefOnHitOnly is false, which means the slider will launch its href upon loading the page. The href is also launched after a forceUpdate() call. By setting hrefOnHitOnly to true, the href will only be launched upon the slider physically being selected or upon a forceHit() call.

initHref

[0101] VALUE="function"- Only valid if FromInitHref is used as the initialCondition. Specifies the function called when the page is loaded. The value returned from this function call will be used as the initial condition of the Slider handle.

Orientation

[0102] VALUE="HORIZONTAL"|"VERTICAL"- Specifies if the handle is to travel horizontally or vertically. The orientation parameter will override the orientation determined by the height and width dimensions.

tickLength

[0103] VALUE="number"- The length (size) of each tick mark, in pixels. The range is 1 - 255 (0x01 - OxFF). If tickLength is not given, the default is 9 pixels.

tickPosition

[0104] VALUE="CENTER"|"TOP"|"BOTTOM"|"LEFT" |"RIGHT"- The position of the tick marks in relation to the channel. The range for a horizontal slider is CENTER, TOP or BOTTOM. The range for a vertical slider is CENTER, LEFT or RIGHT. If tickPosition is not given, the default is CENTER (inside the channel).

waitForInit

[0105] VALUE="TRUE" |"FALSE" — Only valid if FromInitHref is used as the initialCondition. Specifies if the Slider handle will wait for valid data before being displayed on the channel. If TRUE, the Slider handle will not display until the data from the initHref function is received. If FALSE, or the attribute is not present, the Slider handle momentarily starts at the minAt location until the

initHref function receives its data.

List Parameter Types

[0106] Fig. 14 lists the parameters that describe the List widget in one or more embodiments of the present disclosure.

Name

[0107] VALUE="text" — Specifies the name List widget.

Width

[0108] VALUE="number" — Specifies the width of the List widget.

Height

[0109] VALUE="number" — Specifies the height of the List widget.

fillColor

[0110] VALUE="color" — Specifies the desired list fill color. If no fill color is specified, the default color is the current background color.

Font

[0111] VALUE="font" — Specifies the font used for the option names within the list box. The corresponding .auf file must be included in the Amulet/Color/Color/Configuration/Fonts folder, the root folder, or the root/Fonts folder. Default is Bitstream Vera Sans.

fontColor

[0112] VALUE="color" — Specifies the desired font color. If no font color is specified, the default color is black.

fontSize

[0113] VALUE="number" — Specifies the font size for the option names within the list box. It can be one of the HTML sizes 1-7 or an actual point size (8pt-99pt), such as "9" or "9pt". Only one value is allowed per list box; you cannot mix font sizes. The assigned HTML values are:

    1 = 8-point font

    2 = 10-point font

    3 = 12-point font

    4 = 14-point font

    5 = 18-point font

    6 = 24-point font

    7 = 36-point font

fontStyle

[0114]
VALUE="PLAIN"|"BOLD"|"ITALIC"|"UNDERLINE"|"STRIKE" — Specifies the style associated with the list box font. To create a custom look, styles can be combined by using the "logical or" (|). However, PLAIN overrides any other style. The available font styles are:

    PLAIN — The option text uses the standard font.
    BOLD- The option text is bold. (i.e. **text**)
    ITALIC- The option text is italicized, (i.e. *text*)
    UNDERLINE- The option text is underlined, (i.e. text)
    STRIKE- The option text is stricken through, (i.e.)

Href

[0115] VALUE="function(s)"- The function(or multiple/sequenced functions) invoked when a list item is hit.

initialCondition

[0116] VALUE="string"|"FromInitHre '- Specifies which option string is highlighted when the page is loaded. The value associated with the highlighted option string is the argument given to the href function(s). If "FromInitHre ' is selected, the function specified by the InitHref attribute is called. Whichever option has the same intrinsic value as the returned value from the initHref function will be initially highlighted.

Options

[0117] VALUE="stringl,string2,string3(etc.)"- Specifies the list strings. All strings are comma delimited. Note: Any spaces before or after the commas are included in the text string. To specify an intrinsic value, append "= xxx" to the individual titles; (where xxx is a number from 0 - 65535 (0x00 - OxFFFF) or a STRING. The first option displays at the top of the list and each subsequent option displays directly below the previous.

selectionColor

[0118] VALUE="color"- Specifies the color of the highlighted entry in the list box. If no selection color is specified, the default color is black.

selectionFontColor

[0119] VALUE="color"- Specifies the color of the font within the highlighted entry in the list box. If no selection

color is specified, the default color is white.

downArrow

[0120] VALUE="image" — Specifies the "page down arrow" image to use when the list has more items than can be viewed. If this attribute is not present, then a default image, downArrow.gif, located in Amulet/Color/Configuration/Widgets/List/, is used. (Images can be either .GIF, .JPG, or .PNG.)

downArrowAlt

[0121] VALUE="image" — Specifies the "page down arrow" image to use when the down arrow is selected. If this attribute is not present, then a default image, downArrowAlt.gif, located in Amulet/Color/Configuration/Widgets/List/, is used. (Images can be either .GIF, .JPG, or .PNG.)

initHref

[0122] VALUE="function" — Specifies the function called when the page is loaded. Use this attribute whenever FromInitHref is used as the initialCondition. The value returned from this function call will determine which option string starts out highlighted. The value should match the intrinsic value of one of the options strings.

scrollRate

[0123] VALUE="seconds" — Specifies the rate at which the list scrolls when selecting the area directly below or above the list box. The default scroll rate is .45 seconds. The range is 0.01 - 655.35.

upArrow

[0124] VALUE="image" — Specifies the "page up arrow" image to use when the list has more items than can be viewed. If this attribute is not present, then a default image, upArrow.gif, located in Amulet/Color/Configuration/Widgets/List/, is used. (Images can be either .GIF, .JPG, or .PNG.)

upArrowAlt

[0125] VALUE="image" — Specifies the "page up arrow" image to use when the up arrow is selected. If this attribute is not present, then a default image, upArrowAlt.gif, located in Amulet/Color/Configuration/Widgets/List/, is used. (Images can be either .GIF, .JPG, or .PNG.)

Claims

1. A method for authoring a graphical user interface, GUI, comprising:

> providing (702), using a processor of a computer (501), a widget builder (502) user interface (600) for creating a widget;
> receiving (708; 714) user input, at the processor, of widget properties for the widget, the widget properties including a widget type, one or more widget parameters, and one or more images; and **characterised by**:
> saving (710; 716), using the processor, the widget properties and the one or more images into a widget file (200) conforming to a standard image file format in a computer readable storage medium of the computer.

2. The method of claim 1, wherein the widget is a customized widget created from a template widget or the widget is a template widget for creating customized widgets.

3. The method of claim 1, wherein the widget file is a portable network graphics, PNG, file, the widget type and the one or more widget parameters are saved as comments in a text chunk, and the one or more images are saved in one or more image data chunks.

4. The method of claim 3, wherein the widget type and the one or more widget parameters are saved as XML elements.

5. The method of claim 3, wherein each widget parameter comprises a start tag with a widget parameter name and one or more attributes, an end tag, and one or more widget parameter values between the start and the end tags.

6. The method of claim 5, further comprising:

> receiving user input of settings for the one or more widget parameters as editable or not editable; and
> saving the settings as attributes in start tags.

7. The method of claim 1, further comprising populating a properties editor (602) in the widget builder user interface with widget parameters and widget parameter values from a template widget file.

8. The method of claim 1, where the widget type is selected from a radio button widget, a slider widget, a list widget, a checkbox widget, a button widget, a spinner widget, and a menu widget.

9. The method of claim 1, wherein the one or more widget parameters include a state image comprising one image or multiple image components.

10. The method of claim 1, further comprising:

> providing (902) a GUI builder (506) user interface (800) for building the GUI;
> receiving (904) user input selecting the widget file to create a widget in the GUI;
> parsing (906) the widget file for the widget properties;
> saving the widget properties as part of a project file;
> using the widget properties, calling (914) a rendering tool (508) to render the widget as it would appear on a native platform; and
> displaying (916) the rendered widget in the GUI builder user interface.

11. The method of claim 10, further comprising:

> receiving (928) user input selecting an other widget file to create an other widget in the GUI;
> parsing the other widget file for other widget properties;
> saving the other widget properties as part of the project file;
> using the other widget properties, calling the rendering tool (508) to render the other widget as it would appear on the native platform; and
> displaying the rendered other widget in the GUI builder user interface.

12. The method of claim 10, wherein the rendering tool comprises a translator (510) and a (512) library (120), library wrapper around runtime code of a widget the and calling rendering tool comprises:

> providing the translator with the widget type and the widget parameters, wherein the translator translates the widget type and the widget parameters into data structures for the library wrapper; and
> using the data structures, the library wrapper rendering the widget and providing the rendered widget to the GUI builder user interface

13. The method of claim 12, further comprising:

> extracting the one or more images from the widget file and converting them into one or more objects in a common class;
> creating one or more pointers to the one or more objects; and
> providing the one or more pointers along with the widget type and the widget parameters to the library wrapper via the translator, wherein the runtime code of the widget library uses a pointer to determine an original size of an image, a rendered size of the image, and a position of the image relative to one or more graphical

elements rendered by the rendering tool.

14. The method of claim 12, further comprising:

> receiving user input updating one or more widget parameter values;
> in response to the updated one or more widget parameter values, calling the rendering tool to re-render the widget; and
> displaying the re-rendered widget in the GUI builder user interface in real-time.

15. The method of claim 10, wherein the rendering tool comprises a translator and a library wrapper around runtime code of a widget library for rendering widgets, the method further comprising:

> extracting the one or more images from the widget file into one or more individual image files;
> providing the translator with the project file, wherein the translator translates the project file into HTML code; and
> providing the HTML code and the one or more image files to a compiler, the compiler compiling the HTML code and the one or more image files into μHTML code.

16. The method of claim 15, further comprising:

> programming a controller chip (122) with the μHTML code or running the μHTML code (509) the on a simulator of controller chip, wherein the simulator comprises an application wrapper around runtime code of a widget library for rendering widgets.

Patentansprüche

1. Verfahren zum Erstellen einer grafischen Benutzeroberfläche, GUI, aufweisend:

> Bereitstellen (702), unter Verwendung eines Prozessors eines Computers (501), einer Widget-Erstellungs-(502)-Schnittstelle (600) zum Erstellen eines Widgets;
> Empfangen (708; 714) einer Benutzereingabe, an dem Prozessor, von Widget-Eigenschaften für das Widget, wobei die Widget-Eigenschaften einen Widget-Typ, einen oder mehrere Widget-Parameter und ein oder mehrere Bilder umfassen; und
> **gekennzeichnet durch**:
> Speichern (710; 716), unter Verwendung des Prozessors, der Widget-Eigenschaften und des einen oder der mehreren Bilder in einer Widget-Datei (200), die einem Standardbilddateiformat

entspricht, in einem Computer lesbaren Speichermedium des Computers.

2. Verfahren nach Anspruch 1, wobei das Widget ein angepasstes Widget ist, das aus einem Vorlagen-Widget erstellt wurde, oder das Widget ein Vorlagen-Widget zum Erstellen angepasster Widgets ist.

3. Verfahren nach Anspruch 1, wobei die Widget-Datei eine tragbare Netzwerkgrafikdatei, PNG, ist, der Widget-Typ und der eine oder die mehreren Widget-Parameter als Blöcke in einem Textabschnitt gespeichert werden, und das eine oder die mehreren Bilder in einem oder mehreren Bilddatenblöcken gespeichert werden.

4. Verfahren nach Anspruch 3, wobei der Widget-Typ und der eine oder die mehreren Widget-Parameter als XML-Elemente gespeichert werden.

5. Verfahren nach Anspruch 3, wobei jeder Widget-Parameter einen Start-Tag mit einem Widget-Parameternamen und ein oder mehrere Attribute, einen End-Tag und einen oder mehrere Widget-Parameterwerte zwischen den Start- und End-Tags aufweist.

6. Verfahren nach Anspruch 5, weiterhin aufweisend:

    Empfangen von Benutzereingaben von Einstellungen für den einen oder die mehreren Widget-Parameter als bearbeitbar oder nicht bearbeitbar; und
    Speichern der Einstellungen als Attribute in Start-Tags.

7. Verfahren nach Anspruch 1, weiterhin aufweisend das Einpflegen eines Eigenschafteneditors (602) in der Widget-Erstellungs-Schnittstelle mit Widget-Parametern und Widget-Parameterwerten aus einer Vorlagen-Widget-Datei.

8. Verfahren Anspruch 1, wobei der Widget-Typ ausgewählt wird aus einem Optionsfeld-Widget, einem Schieberegler-Widget, einem Listen-Widget, einem Kontrollkästchen-Widget, einem Schaltflächen-Widget, einem Spinner-Widget und einem Menü-Widget.

9. Verfahren nach Anspruch 1, wobei der eine oder die mehreren Widget-Parameter ein Zustandsbild umfassen, das ein Bild oder mehrere Bildkomponenten aufweist.

10. Verfahren nach Anspruch 1, weiterhin aufweisend:

    Bereitstellen (902) einer GUI-Erstellungs-(506)-Nutzerschnittstelle (800) zum Erstellen der GUI;

    Empfangen (904) einer Benutzereingabe, die die Widget-Datei auswählt, um ein Widget in der GUI zu erzeugen;
    Analysieren (906) der Widget-Datei auf die Widget-Eigenschaften;
    Speichern der Widget-Eigenschaften als Teil einer Projektdatei;
    Unter Verwendung der Widget-Eigenschaften, Aufrufen (914) eines Rendering-Tools (508), um das Widget so zu rendern, wie es auf einer nativen Plattform erscheinen würde; und
    Anzeigen (916) des gerenderten Widgets in der GUI-Erstellungs-Nutzerschnittstelle.

11. Verfahren nach Anspruch 10, weiterhin aufweisend:

    Empfangen (928) einer Benutzereingabe, die eine andere Widget-Datei auswählt, um ein anderes Widget in der GUI zu erzeugen;
    Analysieren der anderen Widget-Datei auf andere Widget-Eigenschaften;
    Speichern der anderen Widget-Eigenschaften als Teil der Projektdatei;
    Unter Verwendung der anderen Widget-Eigenschaften, Aufrufen (914) des Rendering-Tools (508), um das andere Widget so zu rendern, wie es auf einer nativen Plattform erscheinen würde; und
    Anzeigen des gerenderten anderen Widgets in der GUI-Erstellungs-Nutzerschnittstelle.

12. Verfahren nach Anspruch 10, wobei das Rendering-Tool einen Übersetzer (510) und einen Bibliotheks-Wrapper (512) um Laufzeitcode einer Widget-Bibliothek (120) aufweist und das Aufrufen eines Rendering-Tools aufweist:

    Bereitstellen des Widget-Typs und der Widget-Parameter an den Übersetzer, wobei der Übersetzer den Widget-Typ und die Widget-Parameter in Datenstrukturen für den Bibliotheks-Wrapper übersetzt; und
    Verwenden der Datenstrukturen, den Bibliothekswrapper, der das Widget rendert und
    Bereitstellen des gerenderten Widgets an die GUI-Erstellungs-Nutzerschnittstelle.

13. Verfahren nach Anspruch 12, weiterhin aufweisend:

    Extrahieren des einen oder der mehreren Bilder aus der Widget-Datei und
    Konvertieren derselben in ein oder mehrere Objekte in einer gemeinsamen Klasse;
    Erzeugen eines oder mehrerer Zeiger auf das eine oder die mehreren Objekte; und
    Bereitstellen des einen oder der mehreren Zeiger zusammen mit dem Widget-Typ und den Widget-Parametern an den Bibliotheks-Wrap-

per über den Übersetzer, wobei der Laufzeit-code der Widget-Bibliothek einen Zeiger verwendet, um eine Originalgröße eines Bildes, eine gerenderte Größe des Bildes, und eine Position des Bildes relativ zu einem oder mehreren grafischen Elementen, die durch das Rendering-Tool gerendert werden, zu bestimmen.

14. Verfahren nach Anspruch 12, weiterhin aufweisend:

Empfangen von Benutzereingaben, die einen oder mehrere Widget-Parameterwerte aktualisieren;
als Reaktion auf den aktualisierten einen oder die mehreren Widget-Parameterwerte,
Aufrufen des Rendering-Tools, um das Widget erneut zu rendern; und
Anzeigen des neu gerenderten Widgets in der GUI-Erstellungs-Nutzerschnittstelle in Echtzeit.

15. Verfahren nach Anspruch 10, wobei das Rendering-Tool einen Übersetzer und einen Bibliotheks-Wrapper um Laufzeitcode einer Widget-Bibliothek zum Rendering von Widgets aufweist, wobei das Verfahren weiterhin aufweist:

Extrahieren des einen oder der mehreren Bilder aus der Widget-Datei in eine oder mehrere einzelne Bilddateien;
Bereitstellen des Übersetzers mit der Projektdatei, wobei der Übersetzer die Projektdatei in HTML-Code übersetzt; und
Bereitstellen des HTML-Codes und der einen oder mehreren Bilddateien an einen Compiler, wobei der Compiler den HTML-Code und die eine oder mehreren Bilddateien in pHTML-Code kompiliert.

16. Verfahren nach Anspruch 15, weiterhin aufweisend: Programmieren eines Controller-Chips (122) mit dem μHTML-Code oder Ausführen des μHTML-Codeε auf einem Simulator (509) des Controller-Chips, wobei der Simulator einen Anwendungs-Wrapper um Laufzeitcode einer Widget-Bibliothek zum Rendern von Widgets aufweist.

Revendications

1. Procédé de création d'une interface utilisateur graphique, GUI, comprenant :

fournir (702), en utilisant un processeur d'un ordinateur (501), une interface utilisateur (600) de constructeur de widget (502) pour créer un widget ;
recevoir (708 ; 714) une entrée d'utilisateur, au niveau du processeur, de propriétés de widget pour le widget, les propriétés de widget incluant un type de widget, un ou plusieurs paramètres de widget et une ou plusieurs images ; et **caractérisé par** :
enregistrer (710 ; 716), à l'aide du processeur, les propriétés de widget et la ou les images dans un fichier de widget (200) conforme à un format de fichier d'image standard dans un support de stockage lisible par ordinateur de l'ordinateur.

2. Procédé selon la revendication 1, dans lequel le widget est un widget personnalisé créé à partir d'un widget de modèle ou le widget est un widget de modèle pour créer des widgets personnalisés.

3. Procédé selon la revendication 1, dans lequel le fichier de widget est un fichier graphique de réseau portable, PNG, le type de widget et le ou les paramètres de widget sont enregistrés sous forme de commentaires dans un fragment de texte, et la ou les images sont enregistrées dans un ou plusieurs fragments de données d'image.

4. Procédé selon la revendication 3, dans lequel le type de widget et le ou les paramètres de widget sont enregistrés sous forme d'éléments XML.

5. Procédé selon la revendication 3, dans lequel chaque paramètre de widget comprend une étiquette de début avec un nom de paramètre de widget et un ou plusieurs attributs, une étiquette de fin et une ou plusieurs valeurs de paramètre de widget entre les étiquettes de début et de fin.

6. Procédé selon la revendication 5, comprenant en outre :

recevoir une entrée d'utilisateur de réglages pour le ou les paramètres de widget comme étant modifiables ou non modifiables ; et
enregistrer les réglages en tant qu'attributs dans des étiquettes de début.

7. Procédé selon la revendication 1, comprenant en outre le remplissage d'un éditeur de propriétés (602) dans l'interface utilisateur de constructeur de widget avec des paramètres de widget et des valeurs de paramètre de widget provenant d'un fichier de widget de modèle.

8. Procédé selon la revendication 1, dans lequel le type de widget est sélectionné parmi un widget de bouton radio, un widget de curseur, un widget de liste, un widget de case à cocher, un widget de bouton, un widget de pointeur toupie et un widget de menu.

9. Procédé selon la revendication 1, dans lequel le ou les paramètres de widget comprennent une image

d'état comprenant une image ou de multiples composants d'image.

10. Procédé selon la revendication 1, comprenant en outre :

> fournir (902) une interface utilisateur (800) de constructeur de GUI (506) pour construire la GUI ;
> recevoir (904) une entrée d'utilisateur sélectionnant le fichier de widget pour créer un widget dans la GUI ;
> analyser (906) le fichier de widget pour les propriétés de widget ;
> enregistrer les propriétés de widget en tant que partie d'un fichier de projet ;
> en utilisant les propriétés de widget, appeler (914) un outil de rendu (508) pour rendre le widget tel qu'il apparaîtrait sur une plate-forme native ; et
> afficher (916) le widget rendu dans l'interface utilisateur de constructeur de GUI.

11. Procédé selon la revendication 10, comprenant en outre :

> recevoir (928) une entrée d'utilisateur sélectionnant un autre fichier de widget pour créer un autre widget dans la GUI ;
> analyser l'autre fichier de widget pour d'autres propriétés de widget ;
> enregistrer les autres propriétés de widget en tant que partie du fichier de projet ;
> en utilisant les autres propriétés de widget, appeler l'outil de rendu (508) pour rendre l'autre widget tel qu'il apparaîtrait sur la plate-forme native ; et
> afficher l'autre widget rendu dans l'interface utilisateur de constructeur de GUI.

12. Procédé selon la revendication 10, dans lequel l'outil de rendu comprend un traducteur (510) et un enveloppeur de bibliothèque (512) autour du code d'exécution d'une bibliothèque de widgets (120), et l'appel de l'outil de rendu comprend :

> fournir au traducteur le type de widget et les paramètres de widget, le traducteur traduisant le type de widget et les paramètres de widget en structures de données pour l'enveloppeur de bibliothèque ; et
> en utilisant les structures de données, l'enveloppeur de bibliothèque rendant le widget et fournissant le widget rendu à l'interface utilisateur de constructeur de GUI.

13. Procédé selon la revendication 12, comprenant en outre :

> extraire la ou les images du fichier de widget et les convertir en un ou plusieurs objets dans une classe commune ;
> créer un ou plusieurs pointeurs vers le ou les objets ; et
> fournir le ou les pointeurs conjointement avec le type de widget et les paramètres de widget à l'enveloppeur de bibliothèque par l'intermédiaire du traducteur, le code d'exécution de la bibliothèque de widgets utilisant un pointeur pour déterminer une taille d'origine d'une image, une taille rendue de l'image et une position de l'image par rapport à un ou plusieurs éléments graphiques rendus par l'outil de rendu.

14. Procédé selon la revendication 12, comprenant en outre :

> recevoir une entrée d'utilisateur mettant à jour une ou plusieurs valeurs de paramètre de widget ;
> en réponse à la ou aux valeurs de paramètre de widget mises à jour, appeler l'outil de rendu pour rendre à nouveau le widget ; et
> afficher en temps réel le widget rendu à nouveau dans l'interface utilisateur de constructeur de GUI.

15. Procédé selon la revendication 10, dans lequel l'outil de rendu comprend un traducteur et un enveloppeur de bibliothèque autour du code d'exécution d'une bibliothèque de widgets pour rendre des widgets, le procédé comprenant en outre :

> extraire la ou les images du fichier de widget dans un ou plusieurs fichiers d'image individuels ;
> fournir au traducteur le fichier de projet, le traducteur traduisant le fichier de projet en code HTML ; et
> fournir le code HTML et le ou les fichiers d'image à un compilateur, le compilateur compilant le code HTML et le ou les fichiers d'image en code μHTML.

16. Procédé selon la revendication 15, comprenant en outre :
programmer une puce de contrôleur (122) avec le code μHTML ou exécuter le code μHTML sur un simulateur (509) de la puce de contrôleur, le simulateur comprenant un enveloppeur d'application autour du code d'exécution d'une bibliothèque de widgets pour rendre des widgets.

FIG. 1

Graphics file — 200

Image — 204-1

.
.
.

Image — 204-i

Comments
containing
widget
properties — 202
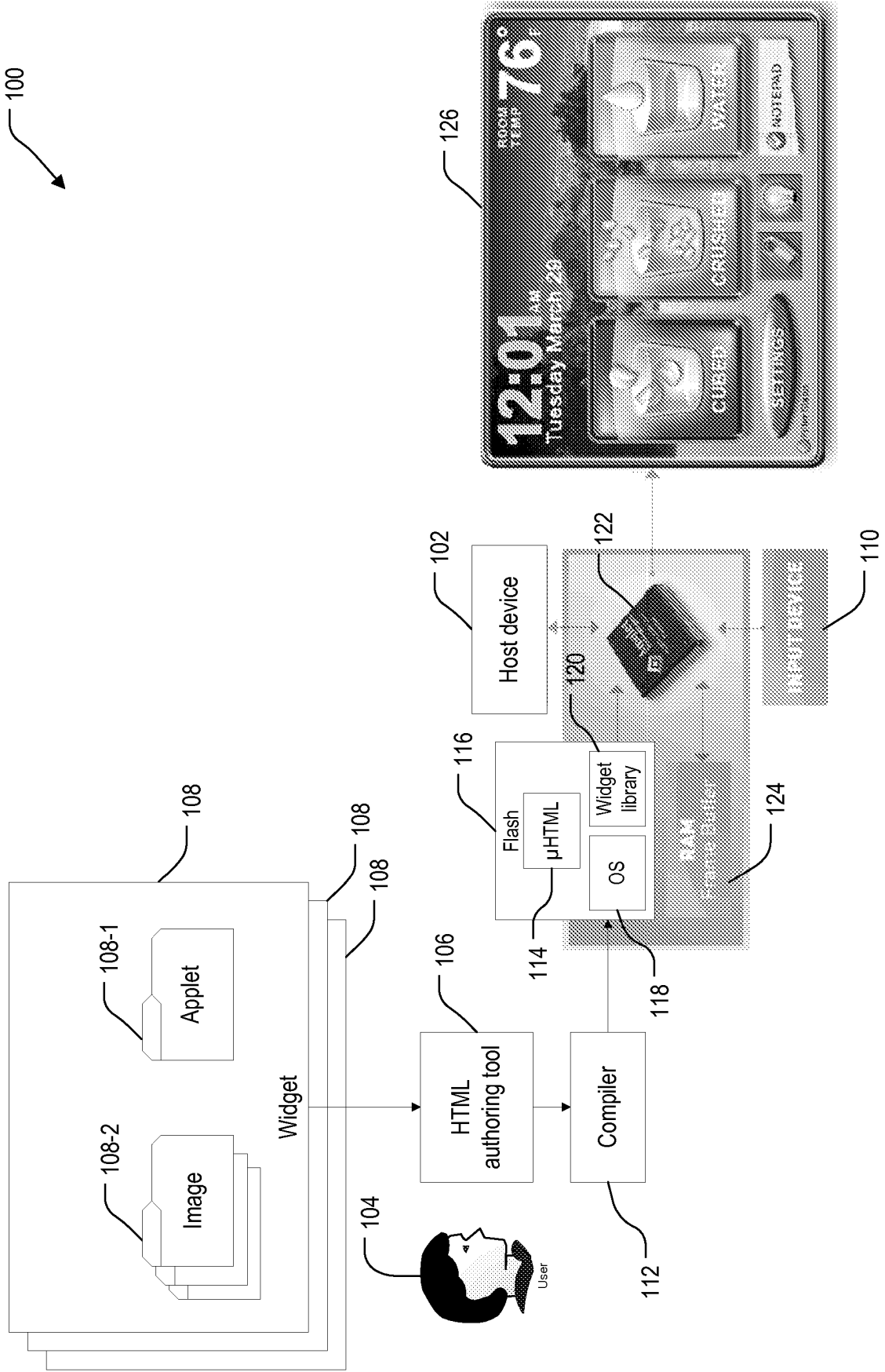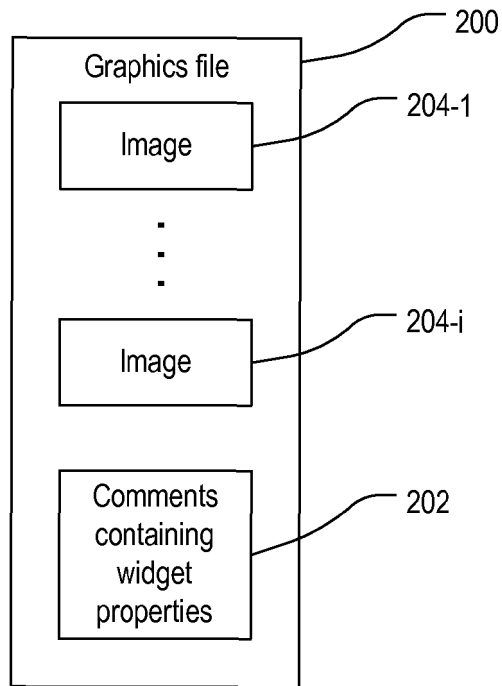
FIG. 2
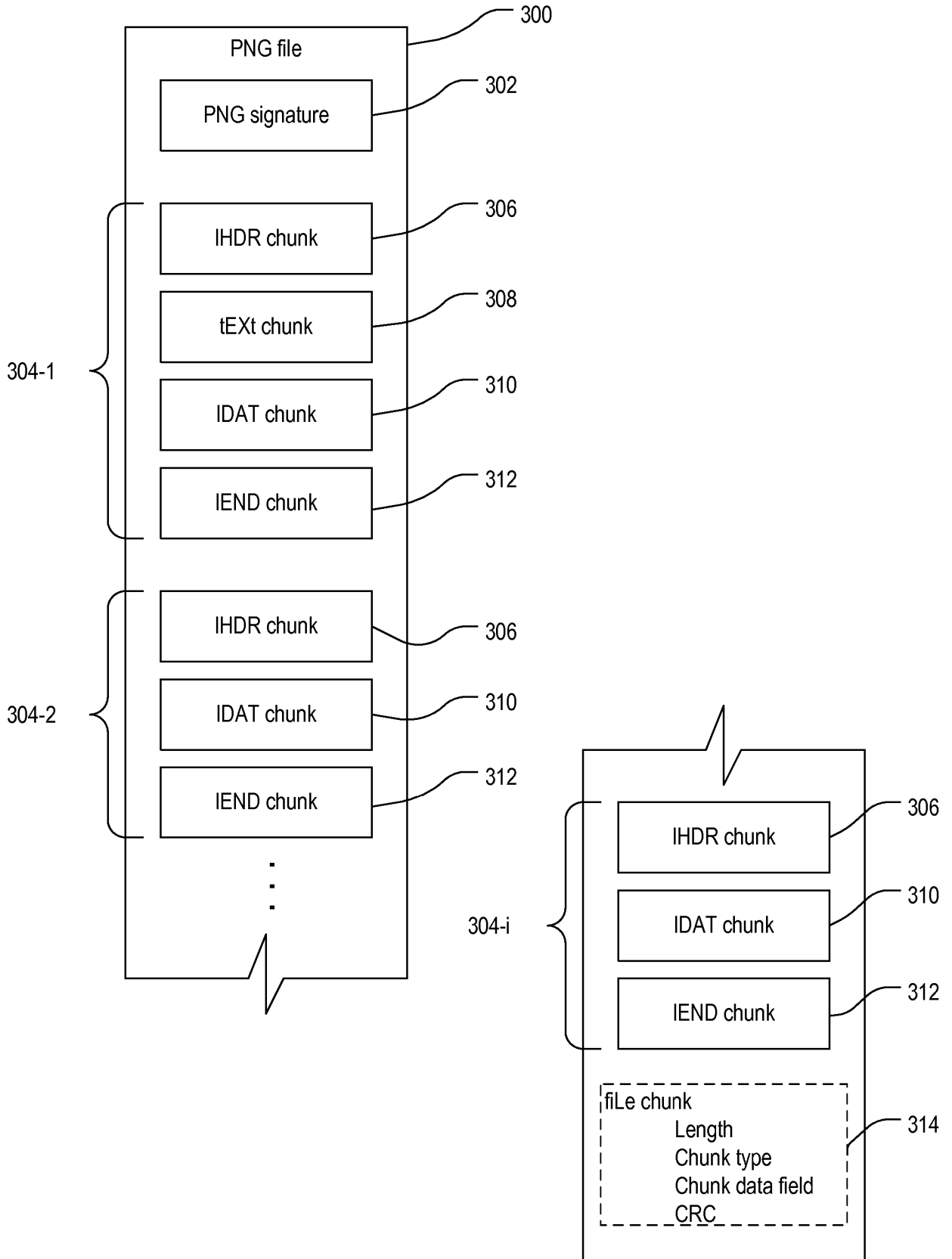
FIG. 3

```
402   <?xml version="1.0" encoding="UTF-8"?>
404   <Template>
406         <RadioButton>
408               <Name Type="12" Editable="True" Optional="False" >RB1</Name>
410               <label Type="12" Editable="True" Optional="False">Button Label</label>
412               <initialCondition Type="10" Editable="True" Optional="False">Off   On
414   Off</initialCondition>
416               <groupName Type="12" Editable="True" Optional="False">rbg1</groupName>
418               <href Type="15" Editable="True" Optional="False" >Amulet:Nop</href>
420               <Width Type="13" Optional="False">162</Width>
422               <Height Type="13" Optional="False">34</Height>
424               <fontStyle Type="18" Editable="True" Optional="False">Plain        Plain    Bold
426   Italic</fontStyle>
428               <Font Type="14" Editable="True" Optional="False">BitstreamVera Sans, 12pt</Font>
430               <emptyImage Type="19" Optional="True"></emptyImage>
432               <fullImage Type="19" Optional="True"></fullImage>
434               <TrackingImage Type="19" Optional="True"></TrackingImage>
436         </RadioButton>
438   </Template>
```

400

FIG. 4

FIG. 5

600

602

604

Property	Value	Static
Name	CrushedIce	
initialCondition:	Off ▼	☐
groupName:	rbg1	☐
href:	Amulet:Nop ▼	☐
Width:	88	☑

Name — 109
label — CrushedIce_down.... ➕
groupName — CrushedIce_up.png ➕
href
Width — CrushedIce_trackin... ➕
Height
fontStyle
Font
emptyImage
fullImage
TrackingImage

608

Image Preview

CrushedIce_down.png
CrushedIce_tracking.png
CrushedIce_up.png

Add   Delete   Add

606   610   612   614

Cancel   Save

FIG. 6

700

Provide user interface for widget builder — 702

New or existing template widget? — 704

Existing

New

Populate properties editor with widget properties from a widget file — 706

Receive input for widget type — 712

Receive input for widget properties — 708

Receive input for widget properties — 714

Save widget properties and image components in a new widget file — 710

Save widget properties and image components in a new widget file — 716

FIG. 7

800

812   813   802

808

LCD: Hantronix HDA570ST-VH   Front   Forward   Back   Backward   Show Grid

Property	Value
Page: 1	
IMAGE	Fridge
Invisible	False
Cache Image	No
RDBTN	Water
Invisible	
initialCondition	Off
groupName	IceAndWater
href	Amulet:Nop
Invisible	
initialCondition	Off
groupName	IceAndWater
href	Amulet:Nop
RDBTN	CubedIce
Invisible	
initialCondition	On
groupName	IceAndWater
href	Amulet:InternalRAM.byte(0).setValue()

New Page
Text
Controls
Image
Background

806

804   810

+   -

Top,Left,Bottom,Right: (56, 211, 174, 298) - Width x Height: (88 x 109)

Program   Run

814   816

FIG. 8

FIG. 9

1000

```
1001  <?xml version="1.0" encoding="UTF-8"?>
1002  <GEMProject GEMstudioVersion="0.2.1.x" CompilerVersion="V1.1.x">
1003    <PAGE NAME="Page_1">
1004      <PROPERTIES>
1006        <HEAD>
1007          <TRANSITIONS>Don't Clear Previous Screen</TRANSITIONS>
1008        </HEAD>
1009      </PROPERTIES>
1010      <BKGND>
1011        <IMAGE>
1012          <Name>fridge</Name>
1013          <invisible Type="2" Editable="True">False    True    False</invisible>
1014          <FILE Type="16" Editable="True">.Library\Images\Backgrounds\fridge.png</FILE>
1015          <CACHED Type="10" Editable="True">No    Yes    No</CACHED>
1016          <MidX Type="13">160</MidX>
1017          <MidY Type="13">120</MidY>
1018          <Width Type="13">326</Width>
1019          <Height Type="13">246</Height>
1020        </IMAGE>
1021      </BKGND>
1022      <BODY>
1023        <RadioButton imageSource=".Library\Controls\Buttons\CrushedIce.png">
1024          <Name>CrushedIce</Name>
1025          <invisible Type="2" Editable="True">False    True    False</invisible>
1026          <initialCondition Type="10" Editable="True">On    On    Off    FromInitHref</initialCondition>
1027          <groupName Type="12" Editable="True">IceAndWater</groupName>
1028          <href Type="15" Editable="True">Amulet:Nop</href>
1029          <MidX Type="13">60</MidX>
1030          <MidY Type="13">130</MidY>
1031          <Width Type="13">94</Width>
1032          <Height Type="13">115</Height>
1033        </RadioButton>
1034        <FunctionButton>
1035          <Name>InvisibleButton</Name>
1036          <invisible Type="2" Editable="True">False    True    False</invisible>
1037          <fontStyle Type="18" Editable="True">Bold    Bold    Italic</fontStyle>
1038          <buttonType Type="10" Editable="True">TOGGLE    Spring-Loaded    Toggle</buttonType>
```
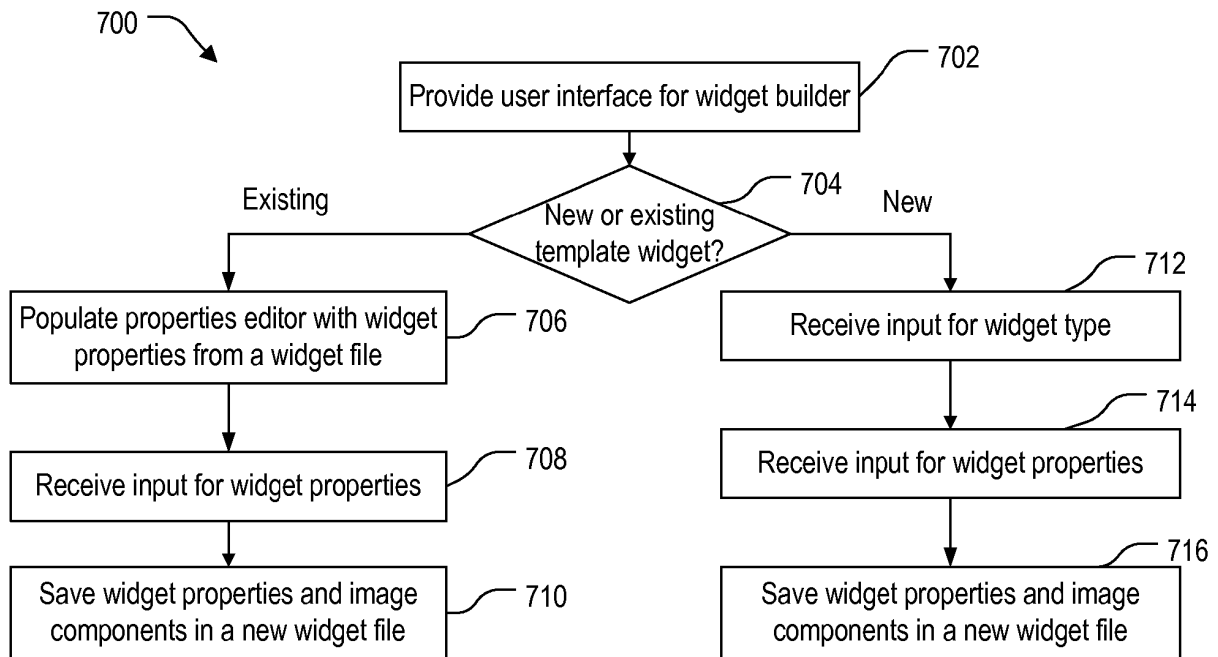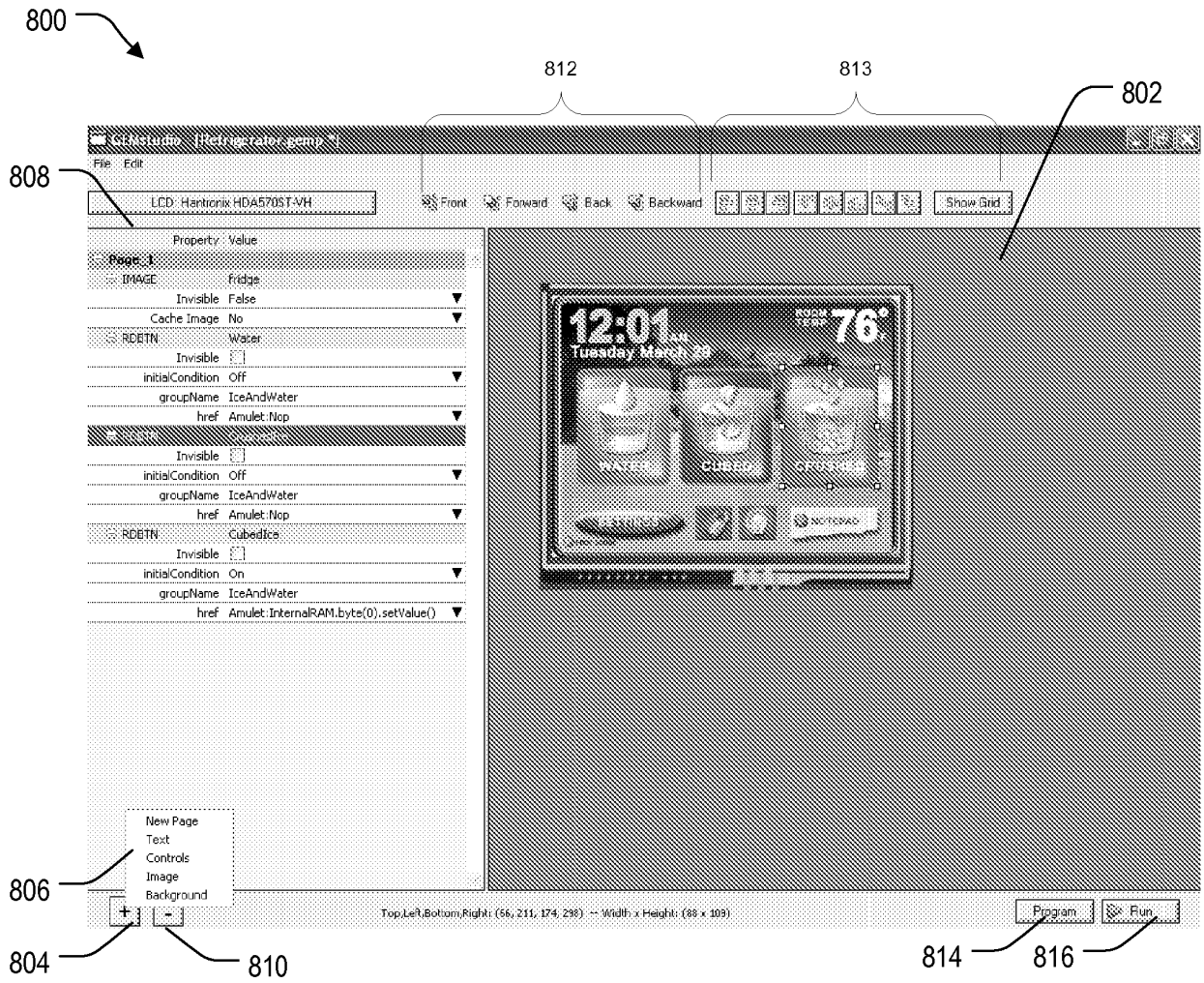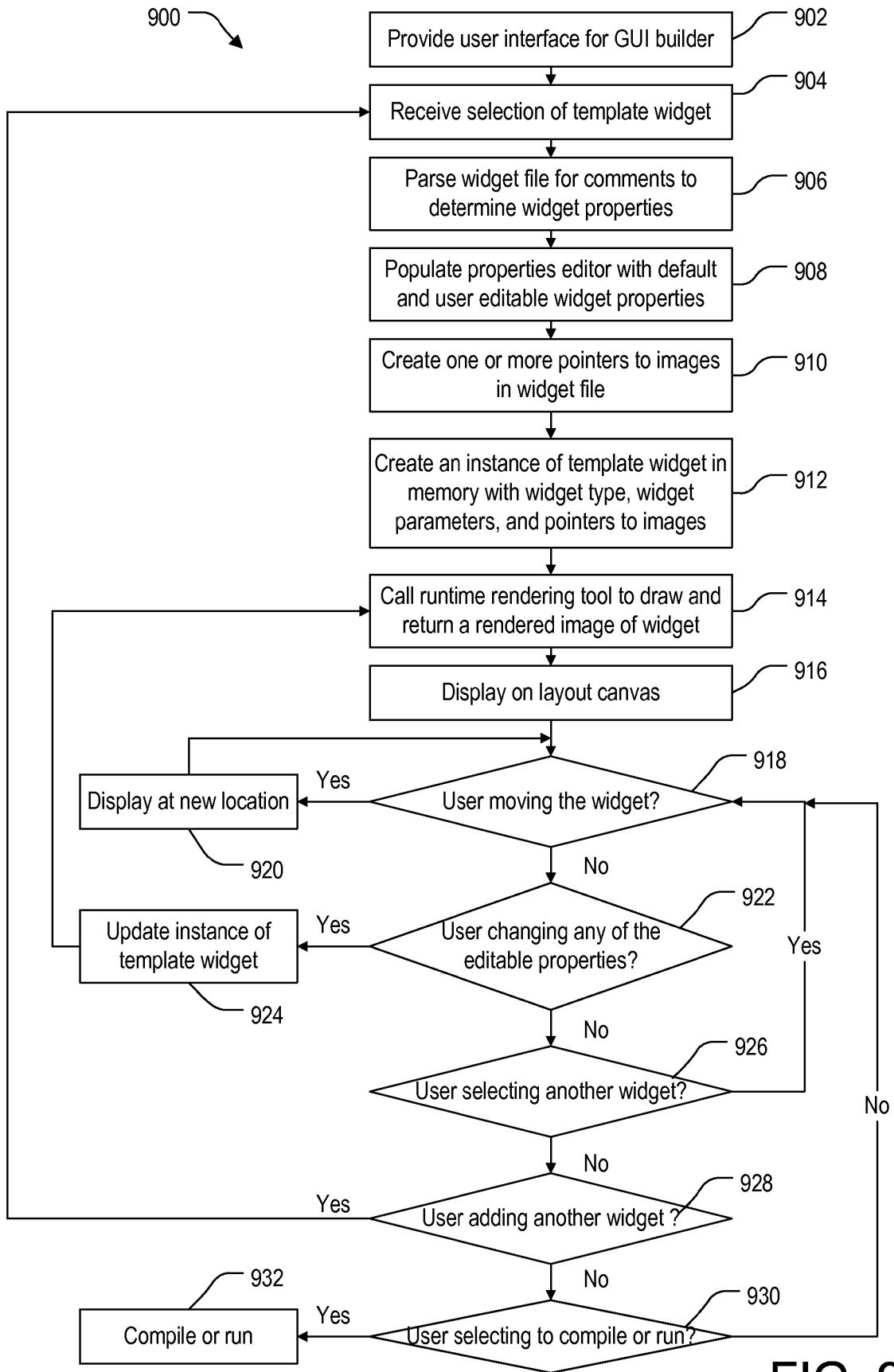
FIG. 10A

25

```
1039        <onButtonPress Type="10" Editable="True">DEPRESS      Alpha      Depress Gray</onButtonPress>
1040        <href Type="15" Editable="True">Page_2</href>
1041        <fillColor Type="11">#FFFFFF00</fillColor>
1042        <borderColor Type="11">#00000000</borderColor>
1043        <label Type="12" Editable="True"/>
1044        <font Type="14" Editable="True">Bitstream Vera Sans, 12pt</font>
1045        <fontColor Type="11" Editable="True">#000000</fontColor>
1046        <MidX Type="13">259</MidX>
1047        <MidY Type="13">207</MidY>
1048        <Width Type="13">89</Width>
1049        <Height Type="13">37</Height>
1050      </FunctionButton>
1051    </BODY>
1052  </PAGE>
1053  <PAGE NAME="Page_2">
1054    <PROPERTIES>
1055      <HEAD/>
1056    </PROPERTIES>
1057    <BKGND/>
1058    <BODY>
1059      <FunctionButton>
1060        <Name>GoBackButton</Name>
1061        <invisible Type="2" Editable="True">False      True      False</invisible>
1062        <label Type="12" Editable="True">Back</label>
1063        <Font Type="14" Editable="True">Tahoma, 12pt</Font>
1064        <fontColor Type="11" Editable="True">#000000</fontColor>
1065        <fillColor Type="11" Editable="True">#316AC5FF</fillColor>
1066        <borderColor Type="11" Editable="True">#000000FF</borderColor>
1067        <href Type="15" Editable="True">Page_1</href>
1068        <MidX Type="13">160</MidX>
1069        <MidY Type="13">120</MidY>
1070        <Width Type="13">73</Width>
1071        <Height Type="13">32</Height>
1072      </FunctionButton>
1073    </BODY>
1074  </PAGE>
1075 </GEMProject>
```
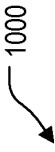
FIG. 10B

FIG. 11

1200

Select one GUI page — 1202

Select one widget from GUI page — 1204

Parse XML widget properties and load them into memory — 1206

Save images as individual image files in HTML temp folder — 1208

Call translator to translate XML widget properties into HTML code — 1210

Call compiler to compile HTML code into µHTML code — 1211

Any remaining widget? — 1212

Yes

No

Any remaining GUI page? — 1214

Yes

No

Wait for compiler to finish — 1216

Call simulator or program controller chip — 1218

Yes

FIG. 12

```
<Template>
        <Slider>
                <Name Type="12" Editable="True" Optional="False">MySlider</Name>
                <Width Type="13" Editable="True" Optional="False">50</Width>
                <Height Type="13" Editable="True" Optional="False">50</Height>
                <channelColor Type="11" Editable="True"
Optional="False">#000000</channelColor>
                <handleColor Type="11" Editable="True"
Optional="False">#FF0000</handleColor>
                <handleFrameColor Type="11" Editable="True"
Optional="False">#000000</handleFrameColor>
                <handleTrackingColor Type="11" Editable="True"
Optional="False">#FFFF00</handleTrackingColor>
                <href Type="15" Editable="True" Optional="False">Amulet:NOP</href>
                <hrefEvent Type="10" Editable="True" Optional="False">onChange
        onChange        onPenUp</hrefEvent>
                <initialCondition Type="12" Editable="True"
Optional="False">hello</initialCondition>
                <max Type="13" Editable="True" Optional="False">253</max>
                <min Type="13" Editable="True" Optional="False">1</min>
                <minAt Type="10" Editable="True" Optional="False">LEFT LEFT    RIGHT
        TOP    BOTTOM</minAt>
                <tickColor Type="11" Editable="True" Optional="False">#000000</tickColor>
                <tickCount Type="13" Editable="True" Optional="False">1</tickCount>
                <handleOffset Type="13" Editable="True" Optional="True">1</handleOffset>
                <handleThickness Type="13" Editable="True"
Optional="True">4</handleThickness>
                <font Type="14.0" Editable="True" Optional="False">Amulet Sans Serif</font>
                <fontSize Type="14.1" Editable="True" Optional="False">12pt</fontSize>
                <fontColor Type="11" Editable="True" Optional="False">#000000</fontColor>
                <fontStyle Type="18" Editable="True" Optional="False">PLAIN        PLAIN
        BOLD    ITALIC UNDERLINE    STRIKE</fontStyle>
                <hrefOnHitOnly Type="2" Editable="True" Optional="True">FALSE  TRUE
        FALSE</hrefOnHitOnly>
                <initHref Type="15" Editable="True" Optional="True">Amulet:NOP</initHref>
                <orientation Type="10" Editable="True" Optional="True">HORIZONTAL
HORIZONTAL    VERTICAL</orientation>
                <tickLength Type="13" Editable="True" Optional="True">2</tickLength>
                <tickPosition Type="10" Editable="True" Optional="True">CENTER  CENTER
        TOP    BOTTOM        LEFT    RIGHT</tickPosition>
                <waitForInit Type="2" Editable="True" Optional="True">FALSE       TRUE
        FALSE</waitForInit>
                <transparency Type="2" Editable="True" Optional="True">FALSE       TRUE
        FALSE</transparency>
                <clearFirst Type="2" Editable="True" Optional="True">FALSE        TRUE
        FALSE</clearFirst>
        </Slider>
</Template>
```

FIG. 13

```
<?xml version="1.0" encoding="UTF-8"?>
<Template>
        <List>
                <Name Type="12" Editable="True" Optional="False">MyList</Name>
                <Width Type="13" Editable="True" Optional="False">50</Width>
                <Height Type="13" Editable="True" Optional="False">50</Height>
                <fillColor Type="11" Editable="True" Optional="False">#FFFFFF</fillColor>
                <font Type="14.0" Editable="True" Optional="False">Amulet Sans Serif</font>
                <fontColor Type="11" Editable="True" Optional="False">#000000</fontColor>
                <fontSize Type="14.1" Editable="True" Optional="False">12pt</fontSize>
                <fontStyle Type="18" Editable="True" Optional="False">PLAIN        PLAIN
        BOLD   ITALIC UNDERLINE      STRIKE</fontStyle>
                <href Type="15" Editable="True" Optional="False">Amulet:NOP</href>
                <initialCondition Type="12" Editable="True"
Optional="False">FromInitHref</initialCondition>
                <options Type="12" Editable="True"
Optional="False">string1,string2,string3</options>
                <selectionColor Type="11" Editable="True"
Optional="False">#0000FF</selectionColor>
                <selectionFontColor Type="11" Editable="True"
Optional="False">#0000FF</selectionFontColor>
                <downArrow Type="19" Editable="True" Optional="True"></downArrow>
                <downArrowAlt Type="19" Editable="True"
Optional="True"></downArrowAlt>
                <initHref Type="15" Editable="True" Optional="True">Amulet:NOP</initHref>
                <scrollRate Type="20" Editable="True" Optional="True"></scrollRate>
                <upArrow Type="19" Editable="True" Optional="True"></upArrow>
                <upArrowAlt Type="19" Editable="True" Optional="True"></upArrowAlt>
                <useAlphaArrow Type="2" Editable="True" Optional="True">FALSE TRUE
        FALSE</useAlphaArrow>
                <alphaColor Type="11" Editable="True"
Optional="True">#000000</alphaColor>
                <borderColor Type="11" Editable="True"
Optional="True">#000000</borderColor>
        </List>
</Template>
```
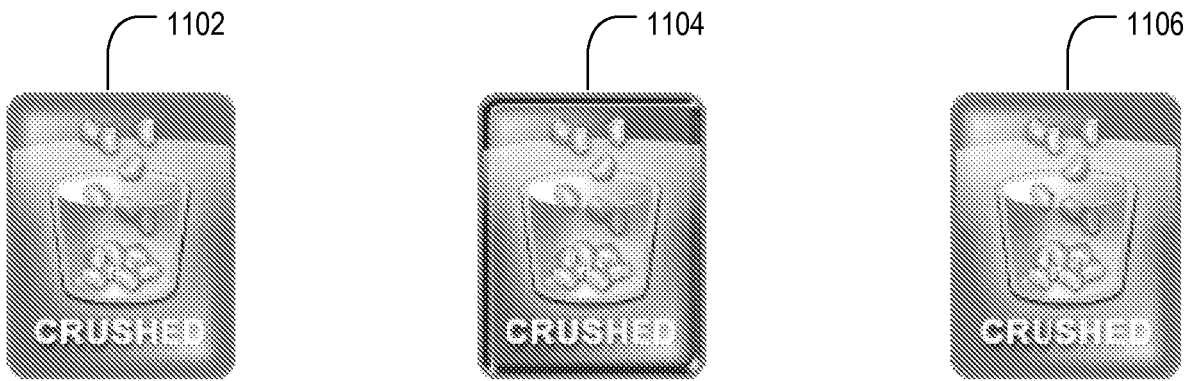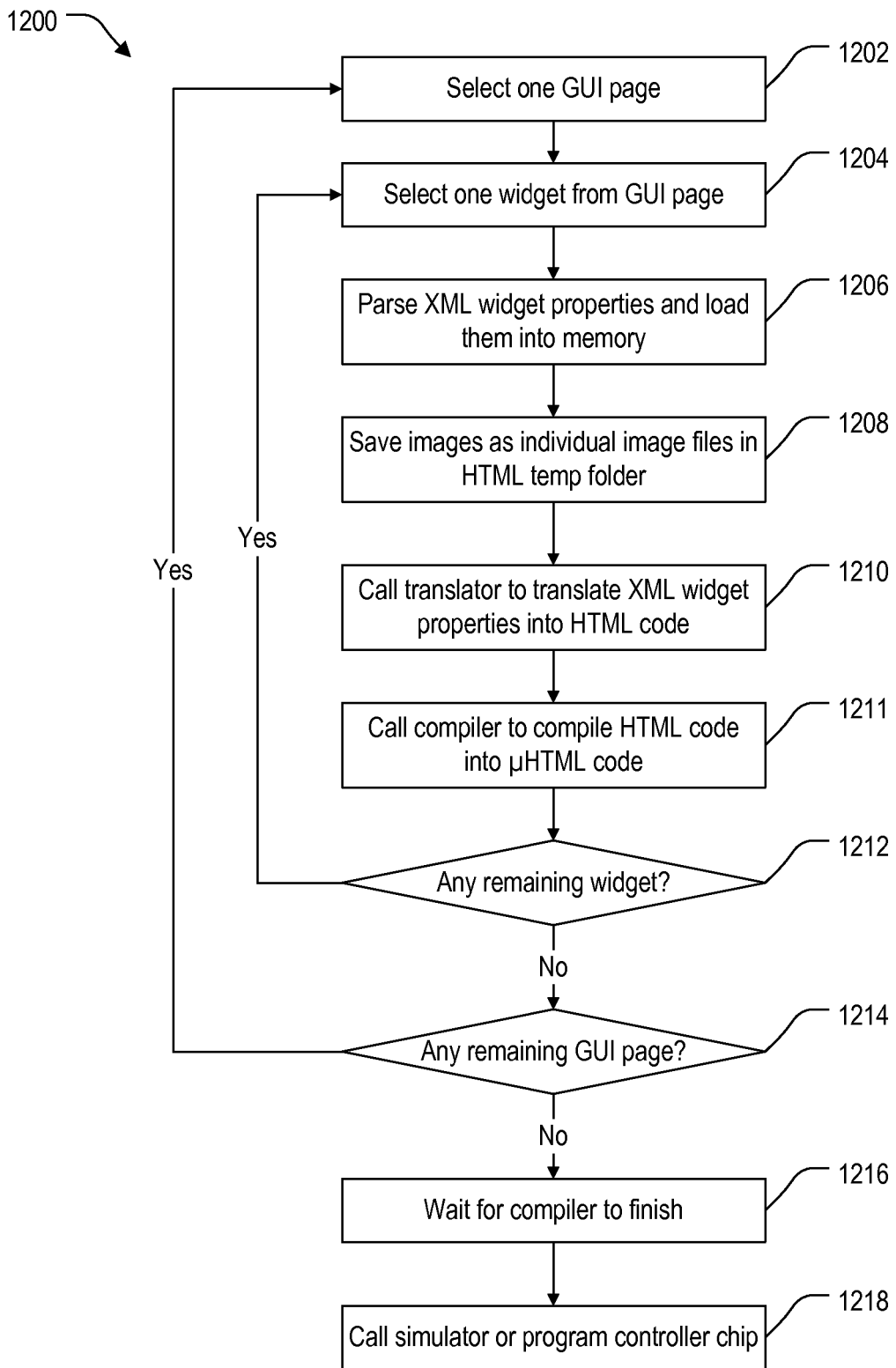
FIG. 14

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Patent documents cited in the description

- US 2006005207 A **[0002]**
- US 7100118 B **[0009]**
- US 502071 **[0009]**