## *The Wonderful World of wxWidgets 3.0*

### What is wxWidgets?

Although it is quite unlikely that you'll read this document if you don't know what wxWidgets is, let's just briefly mention that wxWidgets is a C++ framework for building rich GUI applications from a single source which can then be compiled on different operating systems, resulting in a native application on each system. wxWidgets uses native controls (or widgets) and other native functions wherever possible so that the resulting applications will look and feel as native as possible, and they are usually not distinguishable from applications written using single platform toolkits such as MFC for Windows, GTK+ for Linux or Cocoa under OS X. In some areas (such as graphics art or the installer), adaptations to the individual platforms have to be made in order to achieve perfect integration with that platform.

The major operating system for which wxWidgets supports are Windows (Windows 95, NT, 2000, XP, Vista) including its mobile variants (Windows CE, PocketPC, Windows Mobile), Linux and Unix using the GTK+ 2 toolkit (minimum version is GTK+ 2.6, more recent features are used when available) and Mac OS X (minimum version 10.5 Tiger, both Intel, PPC and the Universal Binaries for both are supported). wxWidgets includes many code pieces for optimising dialog and general layout for small screens such as those of the recent netbooks and mobile phones and tablets.

There is varying support for other platforms or toolkits such as OS/2, Motif, GTK 1.2 and various mobile Linux variants using GTK+ or the Hildon framework and also a version for OS X using the Cocoa API and even the iPhone SDK.

### Documentation in Doxygen

Until wxWidgets 3.0 all documentation was written in a customized LaTeX variant created for the project years ago. Although there were tools which could parse classes automatically and create a documentation skeleton, class documentation was troublesome to update and therefore often outdated. In order to improve this situation, the entire documentation including references and overviews was converted to a customized Doxygen format inlined in a special set of headers. Although many classes were converted in a single automated step, every class documentation had to be corrected by hand making this effort one of the biggest in the development cycle leading up wxWidgets 3.0. Additionally, tools were written to automatically compare the signature of the many class methods to the documentation. The result is more correct documentation with better formating and built-in searching and screenshots of many controls. Since Doxygen is a wide-spread format and easy to learn, the new documentation is much easier to edit, correct and read. See the [wxWidgets on-line documentation](http://docs.wxwidgets.org/trunk/index.html) to which this document refers to in many places.

### C++ features and template support

The wxWidgets project tries to both move with new developments of the C++ language as well as to support older compilers to an extent which does not inhibit further development and indeed the usefulness of the entire project. Since support for templates used to be limited to a few compilers and was often buggy even in them, wxWidgets initially stayed away from using templates entirely including the use of the Standard Template Library (STL). In the meantime nearly all compilers have gained solid template support and therefore wxWidgets is now using templates for container classes (such as [wxVector<T>](http://docs.wxwidgets.org/trunk/classwx_vector_3_01_t_01_4.html)), smart pointers (such as [wxSharedPtr<T>](http://docs.wxwidgets.org/trunk/classwx_shared_ptr_3_01_t_01_4.html)), weak references (see [wxWeakRef<T>](http://docs.wxwidgets.org/trunk/classwx_weak_ref_3_01_t_01_4.html)) and many other places where templates are useful. This means that very old compilers won't be able to compile wxWidgets anymore or only in a degraded way (such as Visual C++ 6.0).

### Platform features and backwards compatibility

In the same way wxWidgets tries to both make use of new features of the different operating systems and support older systems for as long as possible and as long as supporting them does not hinder development for up-to-date systems. This is especially true for OS X and GTK+ 2 and it was therefore decided that OS X versions older than 10.5 Leopard and GTK+ 2 version older than 2.6 are no longer supported. The wxWidgets team also realized that it could not do everything and that support for a cross-platform database API was beyond the scope and focus of the project so that its old wxODBC database connectivity classes were removed from the project. There are many cross-platform database libraries available and many of them are better than the old wxODBC and all of them are better maintained.

### Unicode: A Single Build for Everyone

Until version 3.0 there have always been two different versions (or builds) of wxWidgets: one with full support for Unicode where each character was represented by a wchar\_t internally (using two bytes under Windows and four bytes almost everywhere else) and another called the „ANSI“ build where each character was represented by a single byte. This model was chosen following the original Windows API model and at a point of time when Unicode support was hardly present anywhere else. In the meantime, the Windows world together with projects such as Java have chosen UTF-16 as the native representation for Unicode strings whereas much of the free software world including GTK+ and parts of Mac OS X have chosen UTF-8. It was therefore decided to drastically change the implementation of wxWidgets' string class and make it use UTF-16 under Windows (mostly as before) but UTF-8 elsewhere (instead of wide character strings using wchar\_t) so that strings received from and sent to Unix and GTK+ library calls would no longer have to be converted back and forth between different Unicode representations (see [wxString](http://docs.wxwidgets.org/trunk/classwx_string.html) and [Unicode overview](http://docs.wxwidgets.org/trunk/overview_unicode.html)). Additionally, the „ANSI“ mode was removed and the wxString class as well as some other classes were modified to accept and return both Unicode and 8-bit string literals if required. The same was done to functions like wxPrintf() etc. Although this change will eventually not be seen by the end user of an application written using wxWidgets, it is such a fundamental change that it was the primary reason to give wxWidgets the new major version number 3.

### New 2D Drawing Code

Although a 2D drawing API has always been part of wxWidgets (using so-called device contexts such as a window or a bitmap and pens and brushes to draw into them, see [wxDC](http://docs.wxwidgets.org/trunk/classwx_d_c.html), [wxPen](http://docs.wxwidgets.org/trunk/classwx_pen.html), [wxBrush](http://docs.wxwidgets.org/trunk/classwx_brush.html)), it has not changed much since its initial inception and so the code was completely reorganized using a single set of frontend classes and different backends which will make maintainance much easier without having to care for binary backwards compatibility and it also helped isolate a number of subtle platform differences. The old drawing API is good enough for many tasks and reflects the drawing capabilites of the 1990's but it didn't make use of advanced features such as transparency, anti-aliasing and free matrix transforms of modern 2D graphics systems such as GDI+ on Windows, Cairo on Linux (and elsewhere) and CoreGraphics on OS X. Therefore a completely new drawing API (the so called graphics contexts, see [wxGraphicsContext](http://docs.wxwidgets.org/trunk/classwx_graphics_context.html)) was added to wxWidgets making use of modern drawing engines. This is complemented by a bitmap class with alpha channel support and fast raw access to the bitmap's internal data representation. Additionally the API of all existing GDI class constants was corrected so that wxMODERN becomes wxFONTFAMILY\_MODERN, wxSOLID becomes wxBRUSHSTYLE\_SOLID etc. and the reference counting system was streamlined and made identical on all platforms.

### Changes to wxBase

wxBase is the name of the non-GUI part of wxWidgets libary which provides basic class such as the aforementioned wxString class, container classes, as well as classes for threading, networking, XML parsing, path and configuration management, logging, debugging etc. These functions and classes have been separated into their own library both for being able to write non-GUI apps as well as to make maintainance easier through reduced interdependence.

Many of the changes to wxString and the container classes are located in wxBase, but on top of that support to wxBase was added for events loops, timers and sockets for writing event-based client or server apps with wxWidgets 3.0. The socket code itself has been reorganized removing a lot of duplicated code and dropping the previous implementation which was separated into a C and a C++ part.

### New controls and other major GUI additions for all ports

This document cannot list every bug fix and minor change. Rather, this paragraph summarizes the most relevant changes to the GUI classes of wxWidgets. Given wxWidgets' nature as a GUI library, these changes are also most likely to be visible to the user and may thus be the most important changes from a user's perspective (although not necessarily from a developer's perspective):

* wxDataViewCtrl and wxDataViewTreeCtrl: this control can partially replace both wxListCtrl and wxTreeCtrl (for which there only was a native version of Windows and partially for OS X) but also extends and combines the classes by being able to display a hierarchy and list at the same time and by offering a much more flexible way to display and edit data on a per column basis. Reimplementing wxTreeCtrl and possibly wxListCtrl in terms of wxDataViewCtrl was considered, but this was dropped as certain special features are not available on all platforms (or differently). See also [wxDataViewCtrl](http://docs.wxwidgets.org/trunk/classwx_data_view_ctrl.html), [wxDataViewListCtrl](http://docs.wxwidgets.org/trunk/classwx_data_view_list_ctrl.html) and [wxDataViewTreeCtrl](http://docs.wxwidgets.org/trunk/classwx_data_view_tree_ctrl.html).
* The tabular view of wxGrid has been improved including a native header control, which has been separated into a new control. See also [wxGrid](http://docs.wxwidgets.org/trunk/classwx_grid.html) and [wxHeaderCtrl.](http://docs.wxwidgets.org/trunk/classwx_header_ctrl.html)
* Added wxPropertyGrid which is a big generic control used to display lists and hierarchies of name-value pairs. Like wxDataViewCtrl, it offers a number of ready-to-use editors for editing text, numbers, lists, fonts, file names etc. using in-place editing or using pop-up dialog and combo boxes. Development of wxPropertyGrid has so far taken place outside of wxWidgets as a separate project, but it has not been included in wxWidgets per se. See also [wxPropertyGrid](http://docs.wxwidgets.org/trunk/classwx_property_grid.html).
* wxHyperlinkCtrl added, implemented natively under GTK+ and in a generic way on other platforms. It can be used to represent a hypertext link, for example to the homepage of the developer or company. See also [wxHyperlinkCtrl](http://docs.wxwidgets.org/trunk/classwx_hyperlink_ctrl.html).
* wxFileCtrl for constructing fully customized file dialogs. Complementary to this, the possibility to add custom control to wxFileDialog has been added. See [wxFileCtrl](http://docs.wxwidgets.org/trunk/classwx_file_ctrl.html) and [wxFileDialog](http://docs.wxwidgets.org/trunk/classwx_file_dialog.html).
* Several enhancements to wxRichTextCtrl including support for super- and subscript and many speed-ups. See [wxRichTextCtrl](http://docs.wxwidgets.org/trunk/classwx_rich_text_ctrl.html).
* The possibility to display state icons has been added to wxTreeCtrl. This can also be used to implement check-box like behaviour. See [wxTreeCtrl](http://docs.wxwidgets.org/trunk/classwx_tree_ctrl.html).
* wxCalendarCtrl has been rewritten using native code under MSW and GTK+ and enhanced in many ways (for example displaying week numbers). See [wxCalendarCtrl](http://docs.wxwidgets.org/trunk/classwx_calendar_ctrl.html).
* Implemented support for auto-completion for wxTextCtrl and wxComboBox.
* Added wxAUIToolBar to the set of wxAUI classes, which is better integrated and more flexible than the standard wxToolBar.
* Reimplemented wxBitmapComboBox using native code under MSW and GTK+. See also [wxBitmapComboBox](http://docs.wxwidgets.org/trunk/classwx_bitmap_combo_box.html).
* Added wxBitmapToggleButton on all platforms. See also [wxBitmapToggleButton](http://docs.wxwidgets.org/trunk/classwx_bitmap_toggle_button.html).
* Added support for ellipsization on all platforms and for mark-up formatting under GTK+ to wxStaticText. See [wxStaticText](http://docs.wxwidgets.org/trunk/classwx_static_text.html).
* Rewritten the selection event emission logic of wxListBox on all platforms to more exactly match each other when selecting and deselecting certain items.
* Implemented wxCollapsiblePane natively for GTK and OS X. See [wxCollapsiblePane](http://docs.wxwidgets.org/trunk/classwx_collapsible_pane.html).
* Added a new sizer which can wrap across multiple lines. See [wxWrapSizer](http://docs.wxwidgets.org/trunk/classwx_wrap_sizer.html).
* Added multi-sample and anti-aliasing support to the OpenGL canvas and separated wxGLCanvas and wxGLContext. See [wxGLCanvas](http://docs.wxwidgets.org/trunk/classwx_g_l_canvas.html).
* Added wxNativeContainerWindow in order to construct a wxTopLevelWindow from a native window handle (MSW and GTK+).
* The [wxVScrolledWindow](http://docs.wxwidgets.org/trunk/classwx_v_scrolled_window.html) class has been completely rewritten to accommodate the addition of the new horizontal scrolling variants ([wxHScrolledWindow](http://docs.wxwidgets.org/trunk/classwx_h_scrolled_window.html) and [wxHVScrolledWindow](http://docs.wxwidgets.org/trunk/classwx_h_v_scrolled_window.html)) while still providing complete backwards compatibility for wxVScrolledWindow.

### wxMac specific changes (now called wxOSX)

One important change of the wxMac port is that the port is not called wxMac anymore. Instead, the more appropriate term wxOSX should be used as the operating system is called OS X nowadays and – more importantly – wxWidgets now has partial support for iPhone and iPod, and these are devices are clearly not Macs. Apart from the name change – wxMac has undergone the most fundamental changes of the three main ports, even if some of the changes were mostly reorganizing code instead of writing new code. The code has been reorganized into common code (common to Carbon, Cocoa, and Cocoa Touch) including both general wrapping or front-end classes for much of the GUI code as well as a wrapper for the so called CoreFoundation classes of OS X, which are responsible on all OS X variants for string manipulation, font support, graphics and other basic functionality and toolkit dependent code for the Carbon, Cocoa, and Cocoa Touch API. wxOSX/Carbon is the core of what used to be wxMac and is now deprecated in favour of wxOSX/Cocoa. Existing applications are encouraged to switch to wxOSX/Cocoa as Carbon is a deprecated OS X feature, not available for 64-bit GUI applications, and not available for iOS devices at all.

As part of the restructuring, all remaining drawing code using the old QuickDraw API has been removed (it was only an option before) and drawing now always takes place using CoreGraphics. Likewise, all code using Carbon functions no longer present in OS X 10.4 and 10.5 has been removed to clean-up the code greatly. This is turn means, as mentioned above, that applications will require a minimum of OS X 10.5 in order to run.

Apart from these large changes, these additional features can be noted:

* Better support for IconRef
* A fix for duplicate menu entries in non-English locales
* Accelerators allowed to be used for buttons
* wxLocale::GetInfo() implemented using CFLocale

### wxGTK specific changes

The task of the GTK+ port of wxWidgets is to keep up with the development of the GTK+ library since it has the habit of adding new controls or new APIs if the existing code is too limited and cannot be fixed in a backward compatible way. The main problem of this approach is that applications written using wxGTK should work with relatively old versions of GTK+ but should also make use of recent features. In some cases, supporting an old version of GTK+ hinders development so we decided to declare GTK+ 2.6 the minimum toolkit version that is supported. As an example, this made it possible to always use the GTK+ file dialog instead of the old generic file dialog which had to be used when GTK+ didn't have a usable file dialog.

Other parts of wxGTK that were rewritten or which underwent a major update include, but are not limited to:

* wxToolbar now uses the „new“ GTK+ toolbar API
* wxChoice now uses GtkComboBox instead of the deprecated GtkOptionMenu
* wxComboBox now always uses GtkComboBox instead of the deprecated GtkCombo class
* URL dragging using the „text/x-moz-url“ in wxURLDataObject
* Added a completely new printing backend using with dialogs GtkPrint and Cairo
* Rewritten idle event generation code
* Tab traversal is now done natively by GTK+ instead of by wxWidgets
* Rewrote layout of wxFrame's menubar, toolbar, client window and statusbar using a GtkVBox instead of our own calculation
* Correctly implemented SetSize() and GetSize() for toplevel windows in spite of the dreaded problems with window decorations belonging to the Window Manager and not the window itself
* Added an asynchronous API to wxClipboard to avoid having to call wxYield() from within it (which causes reentrance problems).
* Some support for Hildon control from the Maemo platform used for Nokia tablets
* Rewritten the wxTaskBarIconIcon class using GtkStatusIcon if available.

### wxMSW specific changes

wxMSW is the most mature platform, mostly because it is used most often and thus has the biggest user, tester and developer base, but also because the underlying Windows system has been more successful at preserving backwards compatibility. Therefore, the list of wxMSW-specific changes is smaller and the changes usually minor details when compared to the changes of the other two main ports:

* Implemented more native looking wxCheckListBox and add ability to store client data in it
* Allow longer tooltips
* Support for multiline labels in wxCheckBox and wxToggleButton
* More precise print preview
* Show resize gripper in resizable dialogs