OWL Quality Plan

Final

This document incorporates all previous Elvis quality assurance documents. It is an analysis of the tasks necessary to assure quality for Elvis. It has been reviewed by Tech. Support, and reflects the concerns of our customers.

This document includes the following sections:

- Risk and Task Correlation
- Component Breakdown
- Ongoing Tasks

Resource loading and open issues are not included, due to time constraints, and the need for broader review by management.

Risk and Task Correlation

This table relates risk areas to specific quality assurance tasks. Any tasks listed on the right which are not completed will increase the likelihood of customer dissatisfaction in the associated risk area on the left.

Source Code Usability	• Review code for comments, style, formatting, and comprehensibility. • Review makefiles for simplicity, documentation, and consistency.
Performance	• Benchmark performance of low level encapsulation and high-order functionality versus • OWL 1.0x • MFC • Native Windows apps • Actively solicit Beta tester feedback, design questionnaire, tabulate/analyze results.
Internationalization	• Verify international enabling of the following: • Stored strings (window titles, diagnostics, etc.) • Menus items and accelerators • Cutting and pasting text (clipboard support) • Printing • Localized versions of common dialogs • Status line code • Input validation (proper uppercasing, etc.) • filenames/streaming
Design Quality	• Inspect code for appropriate use of C++ idioms. • Participate in discussions to promote: • Design simplicity • Backward compatibility • Appropriate feature set • Flexibility for future technologies
Documentation Quality *Reference Guide*	• Confirm API coverage with latest available header files. • Check completeness of information for each API, member function, and data item. • Review material for overall usability/organization.
Programmer's Guide	• Check for missing pieces: • Versus MFC – • Versus Petzold (native Windows) • Versus our provided examples • Revealed by beta survey feedback • RTL/Classlib functionality used by Elvis • C SDK methods compared with Elvis methods • Review example code versus: • Code style/readability/comprehensibility • Compile-time errors/warnings • Run-time bugs • Review material for overall usability/organization.

Tutorial	• Actively solicit feedback from neophyte Elvis users. • Review example code versus: • Code style/readability/comprehensibility. • Compile-time errors/warnings. • Run-time bugs.
Application size and efficiency	• Benchmark Elvis size (DGROUP, .EXE) and performance vs.: • Elvis 1.0x • MFC • Native Windows apps • Check diagnostics • Measure effect of varying levels of diagnostics • Determine optimum/shipping versions of final vs. 'debug' libraries, re: size/efficiency • Actively solicit Beta feedback from • Power Users (substantial/industrial strength apps.) • Users of C++ that don't tend to write "optimal" code (e.g., reviewers)
Debugger support	• Review comprehensiveness and appropriateness of diagnostics on a class by class basis • Verify debugger support for • Special Elvis needs: entry point/Winmain issues, Elvis diagnostics, etc. • Any debugging problems highlighted by Elvis: heavily templatized code, exceptions, RTTI, linker capacity, etc. • Lobby for debugger features needed to enhance Elvis debugging, e.g., memory mgmt. diagnostics, heap walking capability, etc.
Portability across platforms, APIs, and compilers	• Review Elvis source to assure appropriate use of APIs:: • `#ifdef` or remove Win16-specific calls • `#ifdef` full Win32-specific calls • `#ifdef` Win16 calls which have better Win32/s equivalents • Execute test suites to verify that examples and other suites produce the same output for both static and dynamic libs. • Investigate the following C++ Compilers for Elvis compatibility: • Symantec • MetaWare • Microsoft • CFront • Execute test suites to verify that examples and other suites produce appropriate output for the following (using debug kernel): • Win 3.1 • Win32s on Win 3.1 • Win32/s on Windows NT • Win 3.1 on Windows NT • Win 3.1 on OS/2 • Investigate Elvis compatibility using Mirrors on OS/2.

High-order functionality *System level*	• Review specifications to assure that the following functionality is supported • OLE • VBX • GDI • BWCC • CTRL3D • Track support issues for 3rd party: • Frameworks • Class libraries (Rogue Wave, etc.) • Custom control (widget) collections • Track interoperability issues for Borland products: • Class libraries (Classlib, RTL iostreams, etc.) • Engines (Pdox, BOLE2, etc.) • Internal and external tools (WMonkey, WinSight, Tarzan, Lucy, CBT, etc.)
Feature level	• Verify that examples exist that use features of the 32bit platforms and that include the following functionality: • Event response tables to replace DDVTs • Windows' resources from multiple DLLs; TLibManager • Document View model • OLE DocFile support • Common dialogs • Clipboard support • Floating palette • Window decorations/gadgets (tool bars/status bars) • Input validation support • Printer support • Use of C++ exceptions • Menus (including OLE 2.0 support) • GDI (fonts, brushes, pens, palettes, bitmaps, regions, icons, cursors, DIBs, complete device context encaps.) • Virtual listboxes (1,000,000,000 items) • Edit control without limits • Outliner/Tree structure listbox • Edit control that will take multiple fonts • Print Preview • Edit control like QPW's • Gauges, sliders, spin buttons, split panes • Example(s) showing use of ODAxxxxx (OwnerDrawAccess APIs) • Workshop aware custom controls (there's already a hack on CIS) • OWL custom control(s) that are usable by 'C SDK' style applications

Low-level API encapsulation	• Review message response macros for coverage. • Verify that all appropriate APIs (i.e., OS features) are encapsulated. • Compare item-by-item to MFC and other competitors • Verify that API functionality is fully accessible and fully usable. • Check internal data structures for completeness. • Verify consistency of Elvis abstractions (i.e., compared to the native API parameter order, data types, etc.). • Actively solicit feedback on ease-of-use/friendliness of enabling layer Elvis API.
Backward compatibility and upgradeability	• Assure that the BC4 toolset will work with OWL 1.0x • Assure that OWL 1 apps are upgradeable to Elvis vis-a-vis: ◦ Documentation (usability testing, beta banging, careful inhouse review) ◦ Automated conversion tool works intuitively ◦ Usability and documentation of design changes ◦ A comparison of 'major' techniques used in OWL 1.0x with their current method in Elvis (Are they unnecessarily different? Are they so much better that they're worth the pain to switch? Are the above questions/answers/design decisions fully doc'ed?)
Reliability	• Measure code coverage of examples to determine what should be stressed by new tests. • Create or collect special test code, including at least one large-scale omnibus application. • Create and maintain smoke tests runnable by Integration. • Build OWL library, after each delivery that has changes in source or include files, for:[*] ◦ 16bit small static ◦ 16bit medium static ◦ 16bit large static ◦ 16bit large DLL ◦ 32bit flat static ◦ 32bit flat DLL ◦ All of the above in diagnostic/debugging mode. • Build selected models with -Vf, -O2, -xd, -3, -dc and -po:‡ ◦ 16bit large/medium static (switch every other time between medium and large) ◦ 16bit large DLL ◦ 32bit flat fully optimized for speed and/or size (if not already delivered that way) • Verify that user built libs are identical to 'delivered' libs (except paths and time stamps). • Build all examples in all models listed above and run automated regressions • Verify that OWLCVT converts its test suite correctly.

† These first 12 will all be delivered to customers, on CD-ROM, the first 6, at least, on diskette.

[*] The following configurations may also be delivered on CD-ROM, if sufficient testing can be done.

Component Breakdown

This is a breakdown of OWL components to a reasonable granularity:

1. **TEventHandler**
2. **TStreamable**
3. **TModule**
    - 3.1. TApplication
    - 3.2. TLibManager
    - 3.3. TResId
    - 3.4. TLibId
4. **TDocManager**
5. **TDocTemplate**
6. **TDocument**
    - 6.1. TFileDocument
    - 6.2. TDocFileDocument
7. **TView (TEditSearch and TListBox parentage)**
8. **TWindow**
    - 8.1. TDialog
        - 8.1.1. TInputDialog
        - 8.1.2. TPrinterDialog
        - 8.1.3. TCommonDialog
    - 8.2. TControl
        - 8.2.1. TSScrollBarData
        - 8.2.2. TScrollBar
        - 8.2.3. TGauge
        - 8.2.4. TGroupBox
        - 8.2.5. TStatic
        - 8.2.6. TButton
        - 8.2.7. TListBox
    - 8.3. TMDIClient
    - 8.4. TFrameWindow
        - 8.4.1. TMDIChild
        - 8.4.2. TMDIFrame
        - 8.4.3. TDecoratedFrame
        - 8.4.4. TDecoratedMDIFrame
    - 8.5. TLayoutWindow
    - 8.6. TClipboardViewer
    - 8.7. TKeyboardModeTracker
    - 8.8. TFloatingPalette
    - 8.9. TGadgetWindow
9. **TScrollerBase**
    - 9.1. TScroller
10. **TValidator**
11. **TPrinter**
12. **TPrintout**
13. **TGadget**
14. **TException**
15. **TMenu**
16. **TClipboard**

17. **TGdiBase**
    - 17.1. TGDIObject
        - 17.1.1. TRegion
        - 17.1.2. TBitmap
        - 17.1.3. TFont
        - 17.1.4. TPalette
        - 17.1.5. TBrush
        - 17.1.6. TPen
    - 17.2. TIcon
    - 17.3. TCursor
    - 17.4. TDib
    - 17.5. TDC
        - 17.5.1. TWindowDC
        - 17.5.2. TPaintDC
        - 17.5.3. TCreatedDC
        - 17.5.4. TMetafileDC
18. **TPoint**
19. **TRect**
20. **TMetaFilePict**
21. **TDropInfo**
22. **TResponseTableEntry**
23. **TClipboardFormatIterator**
24. **TLayoutMetrics**
25. **Diagnostics support**
26. **Streaming/object persistence support**
27. **Error handling & exceptions**
28. **BOLE2 client/container support**
    - 28.1. Elvis support classes
    - 28.2. BOLE2.DLL component
    - 28.3. ObjectPort interface class
29. **VBX support classes**
30. **OWLCVT porting tool**
    - 30.1. DDVTs to response table entries conversion
    - 30.2. Class name and other text substitutions
31. **Makefiles**
    - 31.1. Library source
    - 31.2. Examples
32. **Examples**
    - 32.1. Large scale (large/complex/high-order feature set)
    - 32.2. Miscellaneous (small size/low-level feature set)
    - 32.3. Non-shipping (but may move into above categories)
33. **Documentation**

    - 33.1. Programmer's Guide
    - 33.2. Reference Guide
    - 33.3. Tutorial
    - 33.4. Online Doc Files
    - 33.5. Online Help