Checklist For Developers

Here is a checklist for developers who are developing UI plugins. This *could* be used for certification purposes.

General UI Guidelines

The Spirit of Eclipse

☐ **Guideline 1.1**

Follow and apply good user interface design principles: user in control, directness, consistency, forgiveness, feedback, aesthetics, and simplicity.

☐ **Guideline 1.2**

Follow the platform lead for user interface conventions.

☐ **Guideline 1.3**

Be careful not to mix UI metaphors. It may blur the original concept, and your own application.

☐ **Guideline 1.4**

If you have an interesting idea, work with the Eclipse community to make Eclipse a better platform for all.

Capitalization

☐ **Guideline 1.5**

Use Headline style capitalization for menus, tooltip and all titles, including those used for windows, dialogs, tabs, column headings and push buttons. Capitalize the first and last words, and all nouns, pronouns, adjectives, verbs and adverbs. Do not include ending punctuation.

☐ **Guideline 1.6**

Use Sentence style capitalization for all control labels in a dialog or window, including those for check boxes, radio buttons, group labels, and simple text fields. Capitalize the first letter of the first word, and any proper names such as the word Java.

Language

☐ **Guideline 1.7**

Create localized version of the resources within your plug-in.

Error Handling

☐ **Guideline 1.8**

When an error occurs which requires either an explicit user input or immediate attention from users, communicate the occurrence with a modal dialog.

☐ **Guideline 1.9**

If a programming error occurs in the product, communicate the occurrence with a dialog, and log it.

UI Graphics

Design

☐ **Guideline 2.1**

Follow the visual style established for Eclipse UI graphics.

☐ **Guideline 2.2**

Use a common color palette as the basis for creating graphical elements.

☐ **Guideline 2.3**

Re-use the core visual concepts to maintain consistent representation and meaning across Eclipse plug-ins.

☐ **Guideline 2.4**

Re-use existing graphics from the Common Elements library or other Eclipse-based plugins.

☐ **Guideline 2.5**

Create and implement the graphical versions of the disabled state of toolbar and local toolbar icons.

☐ **Guideline 2.6**

Use the design templates for creating and maintaining UI graphics to facilitate easy file sharing and efficient production of a large set of graphics.

Specifications

☐ **Guideline 2.7**

Use the file format specified for the graphic type.

❑ **Guideline 2.8**

Use the appropriate graphic type in the location it is designed for within the user interface.

❑ **Guideline 2.9**

Follow the specific size specifications for each type of graphic.

❑ **Guideline 2.10**

Cut the graphics with the specific placement shown to ensure alignment in the user interface.

Implementation

❑ **Guideline 2.11**

Use the cutting actions provided to increase the speed and efficiency of cutting a large number of graphics.

❑ **Guideline 2.12**

Abbreviate file name instead of using the full icon name, e.g. New Interface becomes "newint".

❑ **Guideline 2.13**

Use lower case characters in your file names, e.g. DTD becomes "dtd".

❑ **Guideline 2.14**

Use 10 characters or less in your file names if possible (underscores count as a character).

❑ **Guideline 2.15**

Use a file name suffix that describes its location or function in the tool, e.g. newint_wiz, or its size in the case of icons that require multiple sizes.

❑ **Guideline 2.16**

Keep the original file names provided.

❑ **Guideline 2.17**

Follow the predefined directory structure and naming convention.

❑ **Guideline 2.18**

Keep the original directory names provided.

❑ **Guideline 2.19**

Minimize duplication of graphics within a plugin by keeping all graphics in one, or few, first level user interface directories.

❑ **Guideline 2.20**

Use the active, enabled, and disabled states provided.

Component Development

Commands

❑ **Guideline 3.1**

Each command must have a label, tool tip, and full color image. The label and tool tip must use Headline style capitalization.

❑ **Guideline 3.2**

The command tooltip should describe the result of the command, not the current state of the command. Use the text same as that for the command label.

❑ **Guideline 3.3**

Adopt the labeling terminology of the workbench for New, Delete and Add commands.

❑ **Guideline 3.4**

An command should only be enabled if it can be completed successfully.

❑ **Guideline 3.5**

Command enablement should be quick. If command enablement cannot be quick, enable the command optimistically and display an appropriate message if the command is invoked, but cannot be completed.

Dialogs

❑ **Guideline 4.1**

When a dialog opens, set the initial focus to the first input control in the container. If there are no input controls, the initial focus should be assigned to the default button.

❑ **Guideline 4.2**

Slush Bucket widget (or Twin Box) should flow from left to right with the source objects on the left hand side. It should have the >, <, >>, << control buttons in this order.

Wizards

❑ **Guideline 5.1**

Use a wizard for any task consisting of many steps, which must be completed in a specific order.

❑ **Guideline 5.2**

Each wizard must contain a header with a banner graphic and a text area for user feedback. It must also contain Back, Next, Finish, and Cancel buttons in the footer.

❑ **Guideline 5.3**

Start the wizard with a prompt, not an error message.

❑ **Guideline 5.4**

Seed the fields within the wizard using the current workbench state.

❑ **Guideline 5.5**

Validate the wizard data in tab order. Display a prompt when information is absent, and an error when information is invalid.

❑ **Guideline 5.6**

Only enable the Next / Finish buttons if all required information in the dialog is present and valid.

❑ **Guideline 5.7**

Remove all programming message ID's from wizard text.

❑ **Guideline 5.8**

Use a Browse Button whenever an existing object is referenced in a wizard.

❑ **Guideline 5.9**

If a new file is created, open the file in an editor. If a group of files are created, open the most important, or central file in an editor. Open the readme.html file upon creation of an example project.

❑ **Guideline 5.10**

If a new project is created, prompt users and change the active perspective to suit the project type.

❑ **Guideline 5.11**

If a new object is created, select and reveal the new object in the appropriate view.

❑ **Guideline 5.12**

Create folder objects in a wizard if reasonable defaults can be defined.

❑ **Guideline 5.13**

Use the term "Project name" for the input field label when the item must be a Project; otherwise, use the term "Folder name". Do not qualify the term.

Editors

❑ **Guideline 6.1**

Use an editor to edit or browse a file, document, or other primary content.

❑ **Guideline 6.2**

Modifications made in an editor should follow an open-save-close lifecycle model.

❑ **Guideline 6.3**

Only one instance of an editor may exist, for each editor input, within a perspective.

❑ **Guideline 6.4**

It must be possible to open a separate instance of an editor for each different input.

❑ **Guideline 6.5**

The editor should be labeled with the name of the file, document, or input being edited.

❑ **Guideline 6.6**

In multipage editors, use a tab control for page activation.Tab labels should be kept to one word, and two words at most.

❑ **Guideline 6.7**

All of the commands, except for the obvious commands, available in the editor should be added to the window menu bar.

❑ **Guideline 6.8**

Use the standard format for editor contributions in the window menu bar.

❑ **Guideline 6.9**

If an editor has support for Cut, Copy, Paste, or any of the global commands, these commands must be executable from the same commands in the window menu bar and toolbar.

❑ **Guideline 6.10**

Fill the editor toolbar with the most commonly used items in the view menu.

❑ **Guideline 6.11**

Fill the context menu with selection oriented commands.

❑ **Guideline 6.12**

Use the standard format for editor context menus.

❑ **Guideline 6.13**

Fill the context menu with a fixed set of commands for each selection type, and then enable or disable each to reflect the selection state.

❑ **Guideline 6.14**

Register all context menus in the editor with the platform.

❑ **Guideline 6.15**

Implement an Command Filter for each object type in the editor.

❑ **Guideline 6.16**

If the input to an editor is deleted, and the editor contains no changes, the editor should be closed.

❑ **Guideline 6.17**

If the input to an editor is deleted, and the editor contains changes, the editor should give the user a chance to save their changes to another location, and then close.

❑ **Guideline 6.18**

If the resource is dirty, prefix the resource name presented in the editor tab with an asterisk.

❑ **Guideline 6.19**

Treat read-only editor input as you would any other input. Enable the Save As if possible. Display "Read-only" in the status bar area.

❑ **Guideline 6.20**

If the data within an editor is too extensive to see on a single screen, and will yield a structured outline, the editor should provide an outline model to the Outline view.

❑ **Guideline 6.21**

Notification about location between an editor and the Outline view should be two-way. A context menu should be available in the Outline view as appropriate.

❑ **Guideline 6.22**

An error or warning image should be added to items with the error or warning respectively. A container should have a red X if it there are errors on the container itself, a gray X if any of its descendents have errors (but not the container itself), and no X if neither the container nor any of its descendents have errors.

❑ **Guideline 6.23**

If appropriate, implement the "Add Task" feature in your editor.

❑ **Guideline 6.24**

If appropriate, implement the "Add Bookmark" feature in your editor.

❑ **Guideline 6.25**

Editors with source lines of text should show the current line and optionally column numbers the status line. It's optional for the editor to show line numbers for each line in the editor itself.

❑ **Guideline 6.26**

Table cell editors should support the single-click activation model, and in edit mode, they should render complex controls upon single-click.

❑ **Guideline 6.27**

Changes made in a table cell editor should be committed when a user clicks off the cell or hits the "Enter" key. Selection should be cancelled when user hits the "Esc" key.First letter navigation should be supported as a cursoring mechanism within a cell.

❑ **Guideline 6.28**

When performing fine-grain error validation in an editor, use red squiggles to underline the invalid content. When users move the mouse over the red squiggles, display the error text in a fly-over pop up box.

❑ **Guideline 6.29**

Use the Task view to show errors found when the Save command is invoked.

❑ **Guideline 6.30**

If modifications to a resource are made outside of the workbench, users should be prompted to either override the changes made outside of the workbench, or back out of the Save operation when the Save command is invoked in the editor.

Views

❑ **Guideline 7.1**

Use a view to navigate a hierarchy of information, open an editor, or display the properties of an object.

☐ **Guideline 7.2**

Modifications made within a view must be saved immediately.

☐ **Guideline 7.3**

Only one instance of a view may exist in a perspective.

☐ **Guideline 7.4**

A view must be able to be opened in more than one perspective.

☐ **Guideline 7.5**

A view can be opened from the Window > Show View menu.

☐ **Guideline 7.6**

The view label in the title bar must be prefixed with the label of the view in the Perspective > Show View menu.

☐ **Guideline 7.7**

If a view contains more than one control, it may be advisable to split it up into two or more views.

☐ **Guideline 7.8**

When a view first opens, derive the view input from the state of the perspective.

☐ **Guideline 7.9**

If a view displays a resource tree, consider using the window input as the root of visible information in the view.

☐ **Guideline 7.10**

Use the view pulldonw menu for presentation commands, not selection-oriented commands.

☐ **Guideline 7.11**

Use the standard order of commands for view pulldown menus.

☐ **Guideline 7.12**

Put only the most commonly used commands on the toolbar. Any command on a toolbar must also appear in a menu, either the context menu or the view menu.

☐ **Guideline 7.13**

Fill the context menu with selection oriented actions, not presentation actions.

☐ **Guideline 7.14**

Use the standard order of commands for view context menus.

☐ **Guideline 7.15**

Fill the context menu with a fixed set of commands for each selection type, and then enable or disable each to reflect the selection state.

☐ **Guideline 7.16**

If an object appears in more than one view, it should have the same context menu in each.

☐ **Guideline 7.17**

Register all context menus in the view with the platform.

☐ **Guideline 7.18**

Implement an Command Filter for each object type in the view.

☐ **Guideline 7.19**

If a view has support for Cut, Copy, Paste, or any of the global commands, these commands must be executable from the same commands in the window menu bar and toolbar.

☐ **Guideline 7.20**

Persist the state of each view between sessions.

☐ **Guideline 7.21**

Navigation views should support "Link with Editor" on the view menu

Perspectives

☐ **Guideline 8.1**

Create a new perspective type for long lived tasks, which involve the performance of smaller, non-modal tasks.

☐ **Guideline 8.2**

If you just want to expose a single view, or two, extend an existing perspective type.

☐ **Guideline 8.3**

The size and position of each view in a perspective should be defined in a reasonable manner, such that the user can resize or move a view if they desire it. When defining the initial layout, it is important to consider the overall flow between the views (and editors) in the perspective.

Guideline 8.4

If a perspective has just one part, it may be better suited as a view or editor.

Guideline 8.5

If it is undesirable to have an editor area in a perspective, hide it. Do not resize the editor area to the point where it is no longer visible.

Guideline 8.6

Populate the window menu bar with commands and command sets which are appropriate to the task orientation of the perspective, and any larger workflow.

Guideline 8.7

A new perspective should be opened only if the user explicitly states a desire to do so. In making this statement, the user agrees to leave their old context, and create a new one.

Guideline 8.8

If a new perspective is opened as a side effect of another command, the user should be able to turn this behavior off.

Guideline 8.9

If a new perspective is opened, it should be opened within the current window, or in a new window, depending on the user preference.

Guideline 8.10

The list of shortcuts added to the New, Open Perspective, and Show View menus should be at most 7 plus / minus 2 items.

Windows

Guideline 9.1

Use an Action Set to contribute actions to the window menu bar and toolbar.

Guideline 9.2

Follow the platform lead when distributing actions within an Action Set.

Guideline 9.3

Contribute actions to the window menu bar first, and then to the window toolbar if they will be frequently used.

Guideline 9.4

Define each action set with a specific task in mind.

Guideline 9.5

An action set should contain the smallest possible semantic chunking of actions. Avoid the temptation to provide only one action set for an entire plug-in.

Guideline 9.6

Use an action set to share a set of actions which are useful in two or more views or editors.

Guideline 9.7

Let the user control the visible action sets. Don't try to control it for them.

Guideline 9.8

"Open Object" actions must appear in the Navigate pulldown menu of the window.

Guideline 9.9

Always use the global status bar to display status related messages.

Properties

Guideline 10.1

Use the Properties view to edit the properties of an object when quick access is important, and you will switch quickly from object to object.

Guideline 10.2

Use a Properties Dialog to edit the properties of an object which are expensive to calculate.

Guideline 10.3

Use a Properties Dialog to edit the properties of an object which contain complex relationships to one another.

Guideline 10.4

Properties Dialog should contain the superset of items shown in the Properties view.

Widgets

Guideline 11.1

For Tree and Table widgets that have a checkbox associated with a cell item, changing the current selection should not automatically change the check state of the

For Tree and Table widgets that have a checkbox associated with a cell item, changing the current selection should not automatically change the check state of the selected item. However, the current selection should be set to a given item when its check state is changed.

Standard Components

Guideline 12.1

If appropriate, add actions to standard components of Eclipse using the plug-in registry.

Guideline 12.2

If you subclass or copy any of the standard components, always carry over the standard components' characteristics.

The Navigator View

Guideline 13.1

Add actions to the Navigator View menu, toolbar, and context menu using the plug-in registry.

Guideline 13.2

Use the attributes defined in IResourceActionFilter.java and IProjectActionFilter.java to control the visibility of context menu actions in the Navigator.

Guideline 13.3

Use a "Navigate -> Show In Navigator" command in each view, to link resources back to the Navigator.

The Tasks View

Guideline 14.1

Add markers (tasks, errors and warnings) to the Tasks view using the Marker Manager services from the Core Resources Management plugin.

Guideline 14.2

The description text of each marker should be short and concise, so that it will fit in the status line of Eclipse.

Guideline 14.3

Add actions to the Tasks view menu, toolbar, and context menu using the plug-in registry.

Guideline 14.4

Use the attributes defined in IMarkerActionFilter.java to control the visibility of context menu actions in the Tasks view.

Guideline 14.5

Support F1 keyboard command and link it to an infopop that gives a detailed description of the selected item in the Task view.

The Preference Dialog

Guideline 15.1

Global options should be exposed within the Preferences Dialog.

Guideline 15.2

Expose the preferences for a particular view, editor or window in the view itself, via a menu or tool item.

Guideline 15.3

Start out with a single preference page. Then evolve to more if you need to.

Guideline 15.4

If you create a preference group, use the root page for frequently used preferences, or those preferences which have wide spread effect. Specialize within the sub pages. The root preference page should not be blank.

Guideline 15.5

Attempt to integrate plug-in preferences, wizards, and views into existing categories for a new plug-in first, before considering the creation of a new category.

Flat Look Design

Guideline 16.1

Use Flat Look design for user scenarios that involves extensive property and configuration editing.

Guideline 16.2

Have the core sections on the overview page expanded, and provide a "Home" icon on other pages to take users back to the overview page.

Guideline 16.3

Use grouping elements corresponding to tabs in the Flat Look content editor for the organization of the tree view in outline view.

The Tao of Resource

The Tao of Resource

❑ **Guideline 17.1**

Expose the resource for resource equivalent model objects using an IContributorResourceAdapter.

Accessibility

❑ **Guideline 18.1**

All of the features provided by a tool should be accessible using a mouse or the keyboard.

Glossary

Command
: A *command*, which is invoked by a user to carry out some specific functions, may appear as an item in a menu, or an item in a toolbar. In reflection of this, it has attributes for the menu or tool item label, tooltip, and image. As a plug-in developer, you can contribute commands to the window menu bar and toolbar, or to individual views and editors. Contribution to the window is performed using an *action set*, a set of task oriented commands which the user can show or hide. Contribution to a view or editor is performed using individual command.

Bookmarks View
: A view used to browse the bookmarks in the workbench.

Editor
: An editor is a visual component within a workbench page. It is typically used to edit or browse a document or input object. The input is identified using an `IEditorInput`. Modifications made in an editor part follow an open-save-close lifecycle model (in contrast to a view part, where modifications are saved to the workbench immediately).

File
: An object in the workspace, analogous to files in the file system.

Folder
: A container for files in the workspace.

Navigator View
: A view used to browse the files in the workspace

Outline View
: A view, commonly used to view the outline of the active editor.

Perspective
: A perspective is a visual container for a set of views and editors (parts). These parts exist wholly within the perspective and are not shared. A perspective is also like a page within a book. It exists within a window along with any number of other perspectives and, like a page within a book, only one perspective is visible at any time.

Platform
: A generic framework for the integration of tools.

Preferences
: A Preference Page is used to edit the preferences for a feature in the platform.

Project
: A group of files and folders within the workspace. Each project maps to a corresponding user specified directory in the file system.

Properties View
: A view, typically used to browse the properties for an object in the active editor or view.

Properties Dialog
: A dialog for editing the properties of an object.

Property Page
: A page within a Properties Dialog.

Resource
: The generic name for projects, folders and files.

Tasks View
: A view used to browse the tasks, errors, and warnings within the workspace.

View
: A view is a visual component within a workbench page. It is typically used to navigate a hierarchy of information (like the workspace), open an editor, or display properties for the active editor. Modifications made in a view are saved immediately (in contrast to an editor part, which conforms to a more elaborate open-save-close lifecycle).

Wizard
: A Wizard is typically used to create new resources, import resources, or export resources.

Workbench
: The Workbench provides the user interface structure for Eclipse. The purpose of the Workbench is to facilitate the seamless integration of tools. These tools contribute to extension points defined by the Workbench. The Workbench is responsible for the presentation and coordination of the user interface.

Workspace
: The various tools plugged in to the Eclipse Platform operate on regular files in the user's workspace. The workspace consists of one or more top level projects, where each project maps to a corresponding user specified directory in the file system. Each project contains a collection of folders and files.

Acknowledgement

Screenshots contributed to Eclipse.org and used in this document, originate from plugins released or under development by the following teams:

- Java Development Tooling, Eclipse Subproject
- WebSphere® Studio Application Developer, IBM Corporation
- Rational XDE Professional, IBM Corporation.

By agreeing to share selected elements of their user interface designs (both positive and negative), we feel that these teams have helped make the UI guidelines stronger.

Version 2.1 February 2004

Eclipse v3.x UI Guidelines Updates (appended)

(This section needs to be integrated into the above document.)

The following is a draft of ongoing Eclipse v3.x UI Guideline updates.

Top Ten

This is a starter set of top ten Eclipse UI Guidelines and Violations. The purpose of these lists is to be a starting point for Eclipse UI designers and developers, providing the most important practices to follow and to avoid, respectively.

To see all considerations to-date go to the Top Ten Lists Working Page.

Top Ten Eclipse UI Guidelines

1. Use the Eclipse look and feel if extending or plugging into Eclipse
2. Use common SWT controls to get what SWT offers for cross-platform adaptability and accessibility
3. Be familiar with APIs for the UIs you are building
4. Use icons and graphics consistent with the Eclipse style, decorations, states, and quality
5. Understand the conventions of the OSs you are developing for
6. Use understandable messages to help people recover from error conditions
7. Don't initiate dialogs or wizards in an error state
8. Use quick fix and quick assist mechanisms
9. Reserve time for "polish"

Top Ten Eclipse UI Violations

1. Low quality graphics or not consistent with the Eclipse style
2. Poorly organized or sized dialogs and wizards
3. Useless dialogs
4. Cryptic error messages
5. Messages with concatenated strings
6. Property pages that don't adhere to platform uses (normal and tabbed)
7. Assuming more importance than other contributions (see John/Mik comments below)

General UI Guidelines

Common Error Messages

Summary

Use the common structure of error messages to help users diagnose and recover from errors. This will increase user's self-sufficiency and reduce support costs for Eclipse-based products.

Problem Description

Across Eclipse-based products, error messages are displayed inconsistently. This inconsistency makes it difficult for users to understand error conditions, and to diagnose and recover from problems.

Best Practice

Eclipse-based products should help users to diagnose the underlying problem when one of our error messages is displayed. We need to consolidate, transform or coordinate errors bubbled up through various sub-systems or dependent components. For example, users would currently see a (cryptic) SQL error message when they have provided an incorrect userid or password for a SQL query. In this case, users would have no idea how to fix this problem or where to look for possible solutions. In the ideal scenario, if our tools coordinate or transform the error code returned from DB2, and inform users that there's a problem with the supplied userid or password, we would provide a good experience for our users. We should generally leverage the symptom database for diagnoses and error recovery. One possible approach for the error message transformation or coordination is this: leverage the common error message logging facility. That is, use CBE infrastructure in code to log all error or warning conditions.

[TODO: need more detailed info on the Eclipse v3.3 error message infrastructure...TBD]

Apply the four components of the format for common error messages, to make these messages easily understood by users and to ensure consistency. Each error message should include the following four main components:

• message ID
• message text
• explanation
• action

The **message ID** provides a quick means to distinguish one message from another. The **message text** briefly describes the problem. The **explanation** includes the reason that the message was generated and the condition that caused the message. The **action** suggests ways to resolve the problem. Error messages are intended for end users; with this in mind, we should transform programmatic error text into user-comprehensible description in most cases.

For error messages shown in the user interface (UI), do not show the message's ID, but make it accessible. UI error messages are intended for end users, not developers. Therefore we should not intimidate end users with a (cryptic) message ID that add no immediate value to their understanding of the error condition. However, we should make the unique message ID accessible in the detailed explanation of the message, so that users can search on the Web using the message ID, or submit the ID to IBM's support team for further assistance.

further assistance.

Log each error message shown in the UI to a log file also. Error or warning messages shown to end users in the UI should also be logged in a file, so that these messages can be traced at a later time for diagnosis and recovery. The CBE infrastructure provides a logging facility for this purpose.

For error messages shown in the Console view or logged in files, show the message ID as the first item and make it a hyperlink to additional information. Error messages displayed in the Console view should continue to show the message ID as the first item, but it should be a hyperlink to additional information, so that users can diagnose a problem quickly.

For error messages shown in the Problems view, exploit the QuickFix feature to offer solutions to users. Since limited space is available in the Problems view, for errors or warnings shown in that view, we should exploit the QuickFix feature to offer solutions to users where possible.

Tips and Tricks

• When it is not feasible to consolidate, transform or coordinate every error bubbled up through the various sub-systems or dependent components, focus work effort on the errors frequently encountered in typical user scenarios.
• Obtain a list of top PMRs from our support team, and address those errors as a first priority.
• Leverage the symptom database, information developers and user experience designers to transform error text into meaningful and understandable descriptions for users.
• Use contextual information in the Eclipse IDE workbench, such as currently enabled capabilities, or current user role or persona (e.g. Java developer), to scope the error text.