Wikiprint Book

Title: Best practices when coding EwE6

Subject: Ecopath Developer Site - CodeBestPractices

Version: 37

Date: 2024-05-03 13:05:09

Table of Contents

Best practices when coding EwE6 **3**
User Interface design guidelines 3
Resources and localization 3
Target processor 3
Nasty experiences 3

Best practices when coding EwE6

This page attempts to convey some of the oddities and nasties that we have ran into over the years when building EwE6.

User Interface design guidelines

Please adhere to the User Interface Guidelines when building user interfaces.

Resources and localization

Theoretically, the EwE6 scientific interface can be localized to any language, although we do not envy the poor soul commissioned to perform such a job. In intention all language-specific elements in EwE are provided in either localizable forms or in localizable resource tables. We have tried to consistently implement this but exceptions may exist; please let us know if you find any.

The ScientificInterfaceShared assembly offers a whack of shared resources, such as strings and images, for plug-ins and the main Scientific Interface to share to reduce the amount of scattered resources that need localizing. In your assembly simply add a statement such as import ScientificInterfaceShared.My.Resources = SharedResources, and access all shared resources on the imported SharedResources thingy.

When you develop your own plug-ins with a user interface, please try to stick to the following resource guidelines:

- Use resources provided in ScientificInterfaceShared when possible,
- Set the 'localizable' property of any forms that you develop to True.

Target processor

To date, Microsoft has not updated its Access drivers, the core drivers that EwE6 needs to use its database, to 64-bit. EwE6 and ANY EWE6 PLUG-IN must be compiled at ANY TIME for 32 bit processors. Set all applications to build for x86 processors via: Menu > Build > Configuration Manager > x86 (new if not listed). See ?image.

Nasty experiences

- Always override Dispose(bDisposing) to clean up UI elements, do not use !OnHandleDestroyed? because the .NET framework, which wraps Win32 controls, may call this method during the regular life span of a .NET control to do housekeeping. The call may be followed by ! OnHandleCreated? - it's simply not a valid trigger to assume your control is dying.
- Note that the Visual Studio designer automagically places a Dispose method in its *.designer.vb files which is blocked from debugging. You may want to manually move this method to your main vb file and strip off ! DebuggerNonUserCode? tags that prevent the debugger from stepping through the code.