Advanced Error Handler for LabVIEW Projects

Introduction

Handling the errors efficiently in the code is very important for success of any software development project. Efficient error handling can drastically reduce development time; and can make maintenance much more streamlined.

Many a times enough attention is not given to error handling in design as well as development phases. The end result is unsatisfied end user because many often the application crashes without giving any feedback to the end user.

Reasons behind little or no error handling are as follows

- During the design phase many times more emphasis is given to write algorithms of the process. While considering the algorithms no or little attention is paid towards exceptional conditions (e.g. if one is using USB DAQ card for data acquisition then little attention is paid to "What will happen if user plug out the USB card?")
- If we go through book of any programming language, generally no attention is paid towards error handling (Maybe because what they want to teach us is syntactical stuff)
- Even if training sessions give stress on error handling, they fail to show solid benefits obtained by error handling.
- In tight deadline situations, developer thinks he can save some time if he can avoid some error handling stuff.
- Many Developers think that code Developed by them is perfect and they can make no mistakes, so error handling is not part of their game.
- When problems start appearing developers concentrate only on removing (or more preciously bypassing) the bugs which have been observed, at this stage developers think its too late to apply error handling, often at this time project is more than 90% complete and adding error handling may mean major change in the code.

Why Error Handling is so much important

- If given proper consideration in design phase, error handling can save hours of debugging
- Error handling makes application much more robust

- In case of problem application should attempt for recovery itself.
- Even if your application crashes it can be closed gracefully. (E.g. it can pop up some message explaining the cause of crash etc.)
- Error handling can guide end user to diagnose the system on his own, hence reducing your support cost.
- Error handling routines(more specifically error logs) can give you the list of your most favorite mistakes (e.g. most of the times you forget to generate required folder structure before creating file)
- Error handling provides systematic way to provide more data about bugs, so it helps developer to remove bugs.

Expectations from error handler

Now if we need to apply error handling in each and every project day in and day out. Then why it can't be a set of routines which can be reused over and over.

What are the expectations from such error handler routines?

- These routines should be simple to use (should contain few interfaces)
- Learning curve for using these routines should be small
- These routines should serve us in every project and in every situation without much customization
- These routines should provide systematic way to provide details about bug
- These routines should help developer to write routines for self recovery
- These routines should help end user to resolve the problem on his own
- These routines should be configurable to the highest extent
- These routines should never allow application to crash without giving any feedback to the end user
- These routines should be capable of providing patches for application

Advanced Error Handler

Advanced Error handler is set of routines (VIs) which is developed to serve the purposes mentioned above.

It provides a simple interface which needs almost no training

Advanced Error handler provides multiple ways to handle errors, which are as follows

- Displaying the detail information about the error to the end user by means of a dialog box
- Logging the error information along with a time stamp so that situation can be analyzed later with more systematic approach
- In case of critical error; gracefully closing the application
- Attempting self recovery using custom routines and allowing patch development using custom routines.
- Allowing developer (and end user; if developer wishes) to configure how to handle errors

Even though above features are available, all features should not be used for each and every error. E.g. you don't want to log the cancel error (error generated due to cancel button key press by the end user) in file. So developer should be able to configure (Yes CONFIGURE, he should not write code for each and every error) how to handle each error.

To incorporate error handling using advanced error handler developer needs to use only two sub VIs from the library.

ErrorInit.vi

This VI initializes the error handling routine hence it should be called in the beginning of the application. This routine loads the required files into the memory.

Once this VI is executed, Error handler is ready to handle the errors as per configuration.

Error init loads following files into the memory

- Settings.ini: this file contains the information related to error handler configuration (from where it should read the error handling configuration, should end user be allowed to configure the error handling, where the error information should be logged, what should be done if un-configured error occurs etc.)
- Error description file: this file contains the information regarding how each error should be handled

Errortrap.vi

This VI contains the error in and error out terminals, if the error in terminal gets the error, then it looks up the configuration file and then it handles the error as per configuration.

This VI should be dropped in the code at various places where code is expected to trap the errors.

Following methods are possible if an error is trapped

- Show dialog: displays a dialog to the end user which specifies error code, error description and VI hierarchy of occurrence of error.
- Log Error: write an entry to error log file which contains time stamp, error code, error description, and VI hierarchy of occurrence of error.
- Abort application: (this method should be used only for critical errors) displays a dialog to the end user specifying that due to critical error, application needs to be closed down.
- Invoke routine: specifying the routines for this method runs the specified routine(VI)
- Clear Error: this routine clears the occurrence of error, it should be used to ignore Error, or clear error after successful recovery.

If un-configured error is trapped in this routine, this routine asks the end user to configure how to handle this situation (this happens only if Unknown Err Dialog flag is true in settings.ini. If this flag is false then default settings are used for error handling)

Error Editor.vi

Error editor is a user interface for editing the way errors are handled. Using this interface user can add new error codes to the error description file or he can remove or change existing error code methods.

System Requirements: Advanced Error Handler

Introduction

Advanced Error handler should be a set of easy to use VIs; which will help developer to incorporate error handling in the project. These routines should be developed in such a way that it should fit in every project's error handling requirements without customization.

These routines should also help developer to incorporate detailed error handling routines (will be called as dynamic VIs) into software even after release.

Error Handler Requirements

Error Handling Methods

For each error code following error handling methods should be possible

- Displaying Dialog to show error code, error description and VI hierarchy.
- Logging error code, error description and VI hierarchy along with time stamp
- Aborting application for critical errors
- Clear error options for ignoring errors
- Routine execution, i.e. dynamically running a VI which can handle the error

Miscellaneous Features

- Above methods should be configurable separately for each error code
- In case un-configured error occurs user should be able to specify the handling methods for it (This feature should be protected, and should be exposed to end user only if allowed by the developer)
- There should be default method for handling the error
- There should be a user interface for editing the error handling methods

Support file requirements

Settings.ini

This file will store the configuration information about the error handler. The file will be stored in standard ini format

This file must be located in the data folder (with name settings.ini) as this file will be read in application using relative path.

The file will look like as follows

```
[Settings]
Err Desc Path=/C/ErrorDesc.txt
Log Path=/C/ErrorLog.txt
Unknown Err Dialog=TRUE
[ErrHandler]
Routine=/C/routine.vi
Show Dialog=TRUE
Log Error=TRUE
Clear After Handling=TRUE
Abort Application=FALSE
```

Error description file

This file will store the error handling configuration for every error code. The information will be stored in tab delimited spreadsheet format.

File will have following columns in given order

- Error Code
- Show dialog
- Log error
- Clear Error
- Abort error
- Routine

For Boolean fields(Show dialog, Log error, Clear Error, Abort error) “True” and “False” Strings will be used to convey state

The routine field will contain the path of dynamic VI; to keep this feature unused routine field should be empty

This file can be placed anywhere on the disk with any name, but should be referenced accordingly in settings.ini file

Error Log File

Error log file will log the errors (if configured accordingly) the log will be stored in tab delimited spreadsheet format with following columns (In Given order)

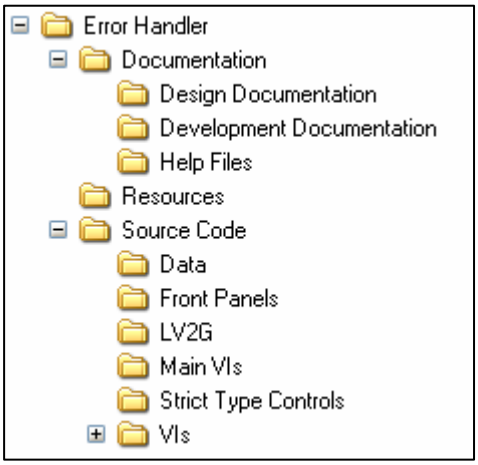
- Date

- Time
- Error code
- Error description
- VI hierarchy

This file can be placed anywhere on the disk with any name, but should be referenced accordingly in settings.ini file

Project structure

The project will have following folder structure



Documentation folder

This folder will contain the development documentation of the error handler system

Development documentation

This folder will contain documents related to developed VIs, bug reports, To Do list, checklist reports, schedule docs, status documents etc.

Help files

This folder will contain the help files for the developed application

Resources Folder

This folder will contain documents related to ideas, searched documents from web, client etc.

Source code

This folder will contain actual source code of application

The source code will be stored in following format

- Main VIs: this folder will contain the VI which are expected to be used as top level VIs (in this case these will be VIs which will be used as SubVIs in other projects)
- Front Panels: Front panels are all those VIs which pops up the panel for user interaction.
- Data: this folder will contain all the support files required for this project
- LV2G: this folder will contain all the VIs which will be used to store data in memory in way similar to global variable.
- Strict type controls: this folder will contain strict type controls required in the project
- VIs: this folder will contain all the remaining SubVIs required in this project. These VIs will be further classified by type (e.g. file I/O, Array handling, mathematical processing etc.)

User Documentation: Advanced Error Handler

Errorinit.vi

Error in (no error): error information from previous code, if it contains Error then error handler will not get initialized

Error out: will notify current error status

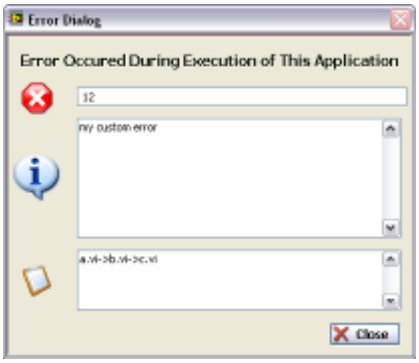
Error Trap.vi

Error in (no error): if this terminal gets the error then error trap will look for appropriate handling method and work accordingly.

Error out: will notify current error status

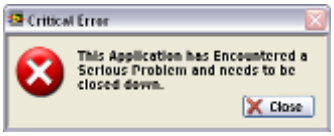
ErrorDialog.vi

This VI pops up the Dialog to display Error code, Error Description and VI Hierarchy



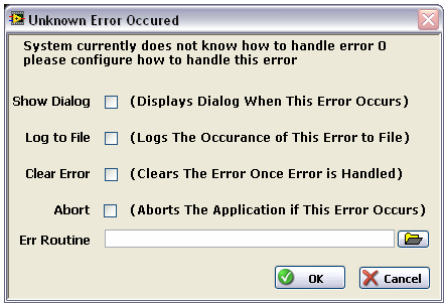
Critical Error Dialog

This Dialog is displayed for critical error Abort. It gives the abort warning to end user and after acknowledgement quits the application.



AddUnknownError.vi

This dialog is popped up when un-configured error occurs, and when user is allowed to configure the un-configured errors.



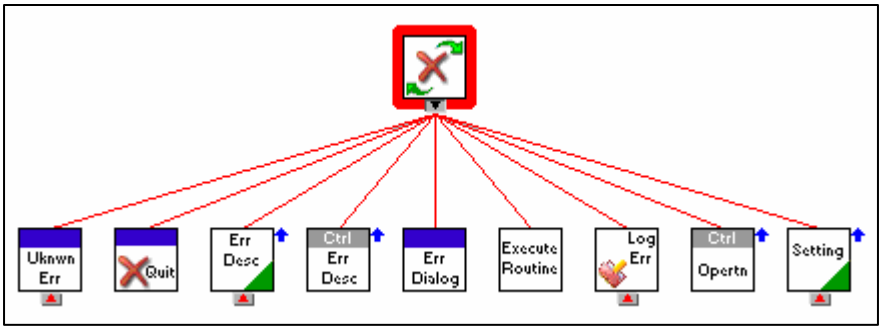
- Show Dialog: user should select this check box if he wish to see error dialog every time particular error occurs
- Log to file: user should select this check box if he wish to log error every time particular error occurs
- Clear Error: User should select this check box to ignore this error
- Abort: user should select this check box if particular is critical, and it needs to close application
- Error Routine: user should type in the path of VI here which can handle this error in more detailed manner, if there is no such routine available then user should leave this field empty.

When this dialog box pops up the default options (as specified in settings.ini) will get selected automatically.

Code Documentation: Advanced Error Handler

Error Trap.vi

VI Hierarchy



SubVI List

Error Description.ctl: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\Strict Type Controls\Error Description.ctl

Operation.ctl: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\Strict Type Controls\Operation.ctl

ErrDesc.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\LV2G\ErrDesc.vi

settings.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\LV2G\settings.vi

AddUnknownError.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\Front Panels\AddUnknownError.vi

ErrorDialog.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\Front Panels\ErrorDialog.vi

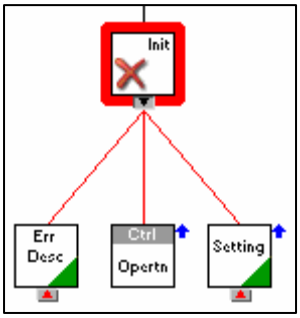
LogError.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\VIs\LogError.vi

Execute Routine.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\VIs\Execute Routine.vi

CriticalAbortDialog.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\Front Panels\CriticalAbortDialog.vi

ErrorInit.vi

Vi Hierarchy



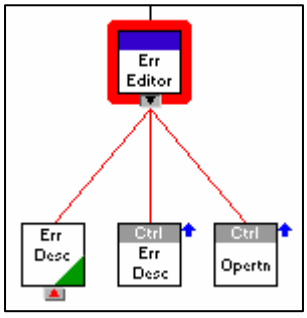
SubVI List

Operation.ctl: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\Strict Type Controls\Operation.ctl

settings.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\LV2G\settings.vi

ErrDesc.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\LV2G\ErrDesc.vi

Error Editor.vi



VI Hierarchy

SubVI List

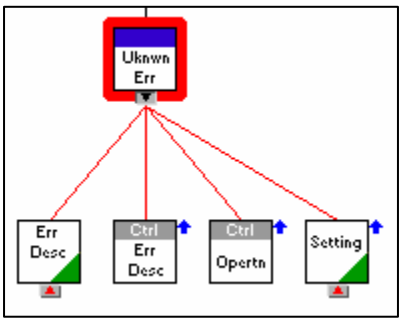
Error Description.ctl: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\Strict Type Controls\Error Description.ctl

Operation.ctl: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\Strict Type Controls\Operation.ctl

ErrDesc.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\LV2G\ErrDesc.vi

AddUnknownError.vi

VI Hierarchy



SubVI List

Error Description.ctl: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\Strict Type Controls\Error Description.ctl

Operation.ctl: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\Strict Type Controls\Operation.ctl

settings.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\LV2G\settings.vi

ErrDesc.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\LV2G\ErrDesc.vi

CriticalAbortDialog.vi

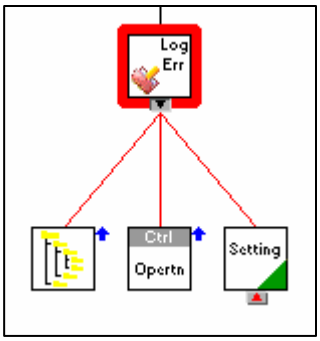
VI hierarchy

ErrorDialog

Vi Hierarchy

Log Error.vi

VI Hierarchy



SubVI List

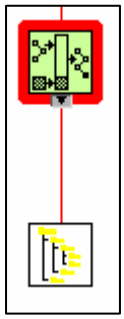
Operation.ctl: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\Strict Type Controls\Operation.ctl

settings.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\LV2G\settings.vi

Create Directory Structure.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\VIs\File IO\Create Directory Structure.vi

StripNLevelsAndBuildPath.vi

VI Hierarchy

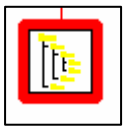


SubVI List

Create Directory Structure.vi: D:\LabVIEW Applications\Inhouse\Error Handler\Source Code\VIs\File IO\Create Directory Structure.vi

Create Directory Structure

VI Hierarchy



Execute Routine.vi

VI Hierarchy

LV2G Details

ErrDesc.vi

Array of Error Description.ctl

Error Description control is cluster of

- Err Code (I32)
- Show Dialog (Boolean)
- Log (Boolean)
- Clear Err (Boolean)
- Abort (Boolean)
- Routine (Path)

Settings.vi

Settings is cluster of

- Err Desc Path (Path)
- Log Path (Path)
- UnknownErrAdd (Boolean)
- Err Methods (Cluster)
    - Dialog (Boolean)
    - Log (Boolean)
    - Clear (Boolean)
    - Abort (Boolean)
    - Routine (Path)

Strict Type Controls Details

ErrDesc.ctl

Error Description control is cluster of

- Err Code (I32)
- Show Dialog (Boolean)
- Log (Boolean)
- Clear Err (Boolean)
- Abort (Boolean)
- Routine (Path)

Settings.ctl

Settings is cluster of

- Err Desc Path (Path)
- Log Path (Path)
- UnknownErrAdd (Boolean)
- ErrMethods(Cluster)
    - Dialog (Boolean)
    - Log (Boolean)
    - Clear (Boolean)
    - Abort (Boolean)
    - Routine (Path)

Operation.ctl

Operation is enumeration with following details

Name	Value
Get Value	0
Set Value	1
Read File	2
Write File	3

Contact Information

Vidisha Innovative Solutions

50, Jai-Jui, Mahesh Soc., Bibwewadi Pune-411037

Ph: +91-02024212661

web: www.vispune.com

Contact Person

Mr. Tushar Jambhekar

Cell: +91-9850611332

Email: tushar@vispune.com