Document Type: Example Program
NI Supported: Yes
Publish Date: Jul 29, 2011

Structured Error Handler (SEH) Reference Library

Overview

The Structured Error Handler (SEH) reference library provides tools for handling errors in an organized fashion. The SEH consists of a configurable Express VI that helps you handle specific errors, a communication mechanism for transmitting errors, a template for a central error handler, and various supporting VIs and utilities.

Table of Contents

Downloads

Filename: ni_seh-2.0.6.9.vip
Requirements: View

Filename: seh_1_0_1.zip
Requirements: View

ni_seh-2.0.6.9.vip - The Structured Error Handler reference library can be installed into your LabVIEW development environment using VI Package Manager™ (VIPM). VI Package Manager is an installer for LabVIEW VIs and toolkits. It places VIs directly into the palettes, allows you to install VIs into multiple LabVIEW versions, and ensures that any dependency VIs and toolkits (provided in other VI packages) are also installed. VI Package Manager is a product and trademark of JKI Software.

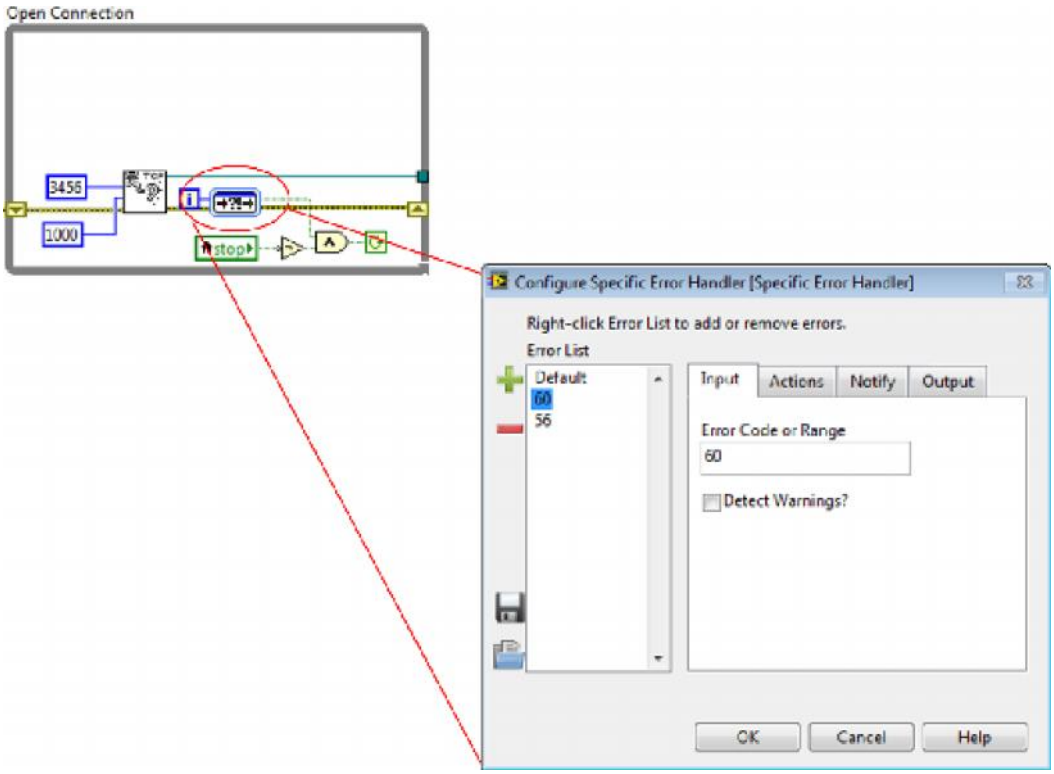
seh_1_0_1.zip - Version 1.0.1 of the library is also attached, which supports LabVIEW 2009 and does not require VI Package Manager to install but lacks some of the functionality described below. Version 2.x of the library will not be back-saved for previous versions of LabVIEW.

Introduction

While LabVIEW provides basic tools for error and exception handling, implementing a comprehensive error handling strategy is challenging and requires significant programming effort. A comprehensive error handling strategy requires both the ability to respond to specific error codes, for example: ignoring an error that does not affect the system, and the ability to take general actions based upon the type of error that occurs, for example: log all "warnings" to a file. This reference library endeavors to simplify these two tasks by providing tools to aid in handling specific errors and defining a central error handling strategy.

Specific Error Handling

The Specific Error Handler is a configurable Express VI which allows you to configure common responses to an error.



Use

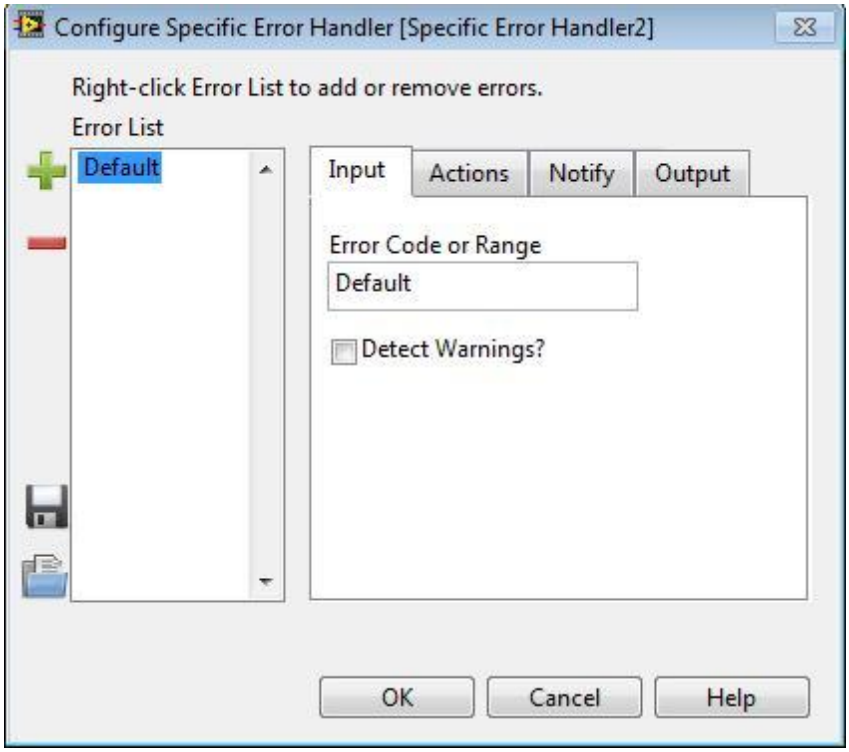
As suggested by its name, the Specific Error Handler is designed to handle specific error codes. This means that the Express VI is best used as close to the source code which throws the error as possible. There are multiple reasons for this. First, by handling the error close to its origin, you avoid any confusion over which VI or section of code threw the error. Because VIs are often used in more than one code section of a system and because LabVIEW shares error codes between VIs, it can often become challenging to identify exactly where an error occurred if it is not handled close to its origin. The second reason for handling an error close to its origin is that some error correction actions, for example: retrying the code section that threw the error, are only effective when the error is detected and handled immediately. The reason the Specific Error Handler is presented as an Express VI is to make it feasible to configure many instances and configurations of the Specific Error Handler with minimal effort.

With the above in mind, it is best to place the Specific Error Handler either directly after the VI which throws the configured errors, or after a short segment of related code. When using the Specific Error Handler to retry, it should be placed in a loop with the code of interest. See the Retrying Code section for more information.

To use the Express VI:

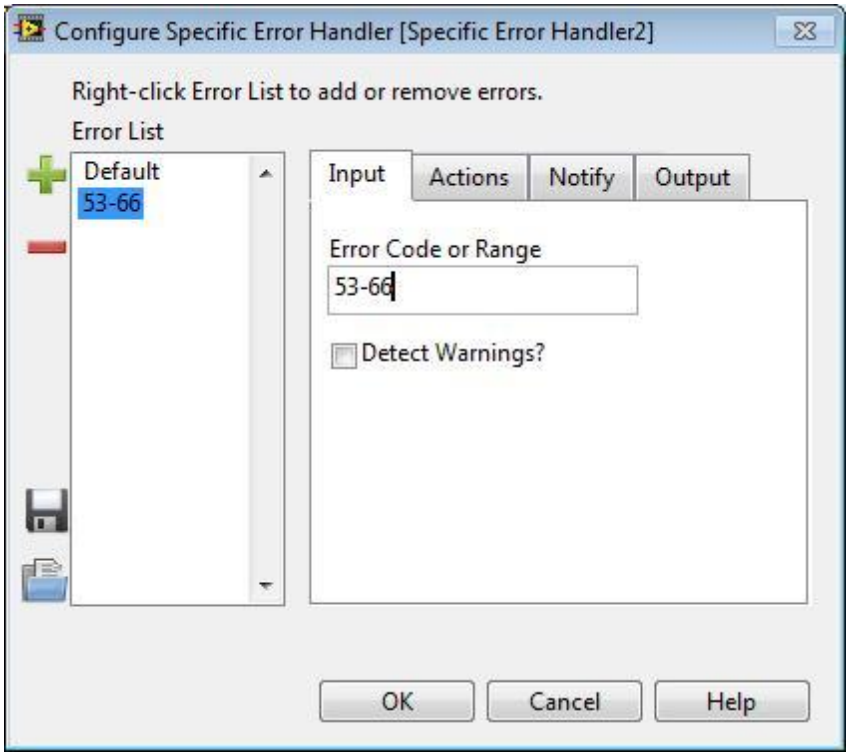
1. Place the VI on the block diagram.

2. Double click to open the configuration window.



3. Press the green + button to add an error code.

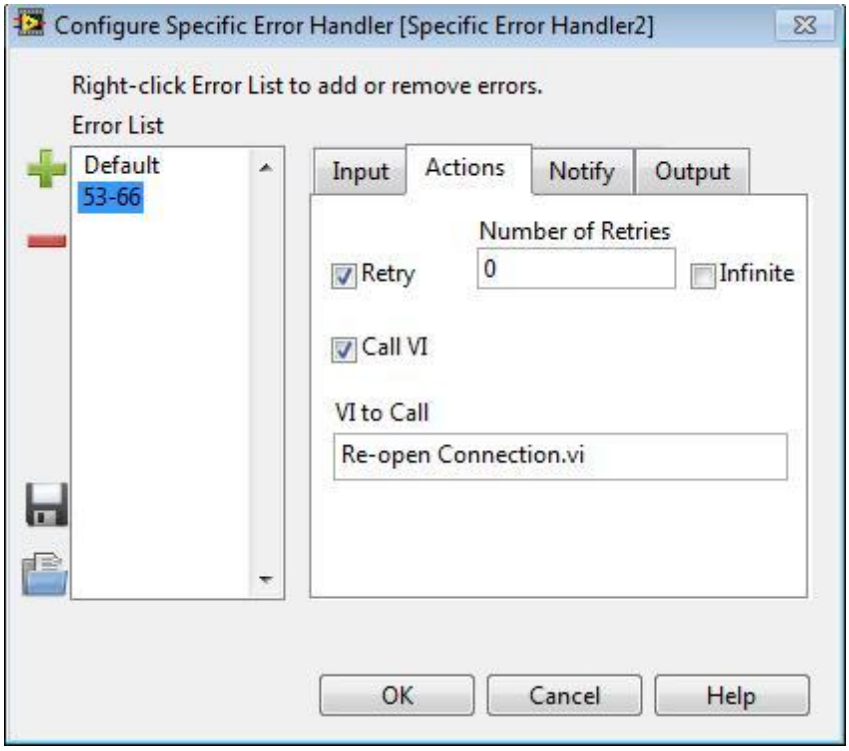
4. Enter the code of the first error you want to handle in the Error Code or Range field. You can enter a single error code or a range of error codes defined by a "-" character. Click outside of the field or press tab to complete the entry.



5. Configure the Input tab for this error.

Detect Warnings? - A warning in LabVIEW is defined by an error cluster that has a non-zero error code, but a status of false. This option determines whether the Specific Error Handler will execute the specified actions if it detects a warning with the correct code. Checking this box will cause the actions to execute any time an incoming error has the correct code. Unchecking the box (default) will cause the actions to execute only if the incoming error has the correct code and a status of true.

6. Configure the Actions tab for this error.



Retry - Determines whether the Specific Error Handler should attempt to re-execute code. This option toggles the Loop? output of the express VI node. See the Retry section for more information.

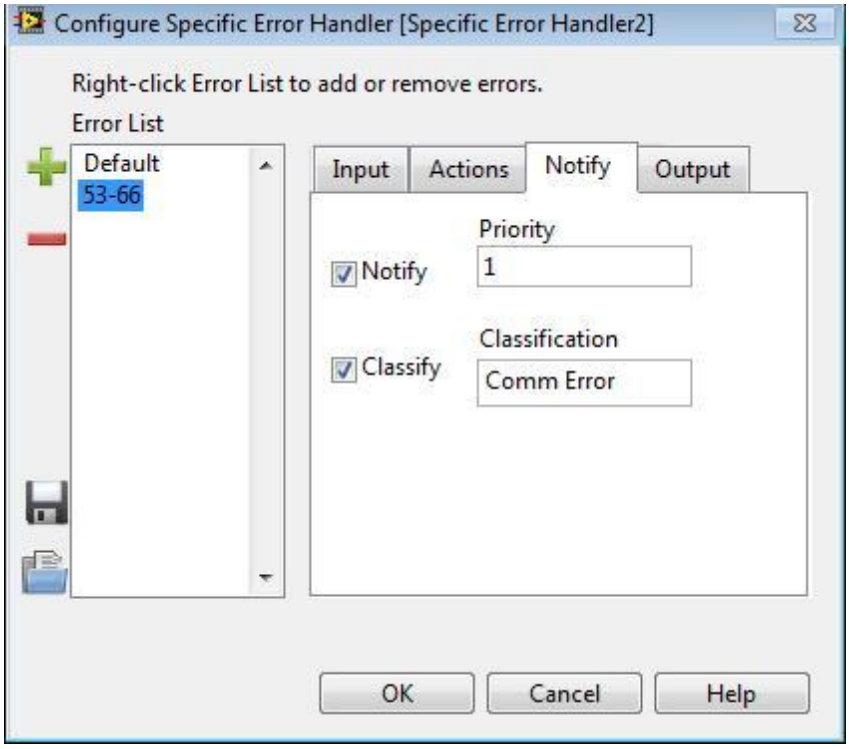
Number of Retries - Determines how many times to retry the code. This option is only visible if **Retry** is enabled. See the Retry section for more information.

Infinite - Indicates that there is no limit to the number of times to retry. This option is only visible if **Retry** is enabled. Enabling this option disables and ignores the **Number of Retries** option. Note that you should usually define some other condition to exit the retry loop to avoid halting execution indefinitely. See the Retry section for more information.

Call VI - Calls a VI from memory. See the Call a VI section below for more information. Note that this option does not exist in the Real-Time version of the library. See the Real-Time Error Handler section for more information.

VI to Call - Determines which VI should be called. This option is only visible if **Call VI** is enabled. Type the VI name, with no path information into this field. Make sure to check for spelling, punctuation, or capitalization mistakes, as there is no edit-time validation that the VI to be called exists. See the Call a VI section for more information.

7. Configure the Notify tab for this error.



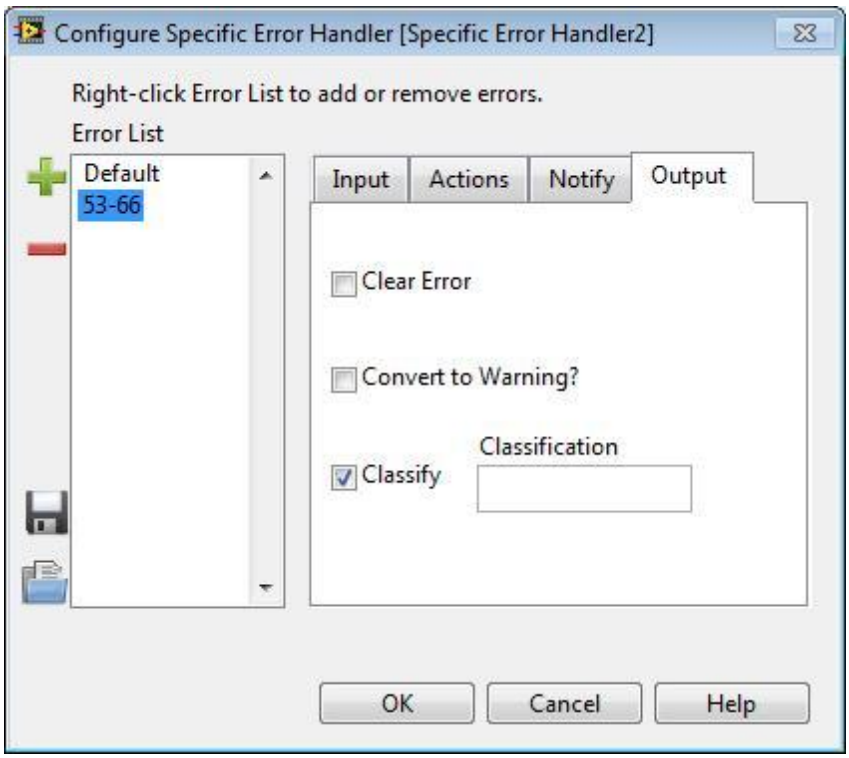
Notify - Transmits the error to a Central Error Handler. See the Error Communication section for more details.

Priority - Determines the order in which notifications should be processed. Lower priority numbers have higher priority, with 0 being the highest priority.

Classify - Classifies the notification. Note that this classification applies only to the notification and will not be applied to the error output. See the Classify Error section for more details.

Classification - A string description of the type of error. See the Classify Error section for more details. Note that in the Real-Time version of the library the string must be four or less characters. See the Real-Time Error Handler section for more information.

8. Configure the Output tab for this error.



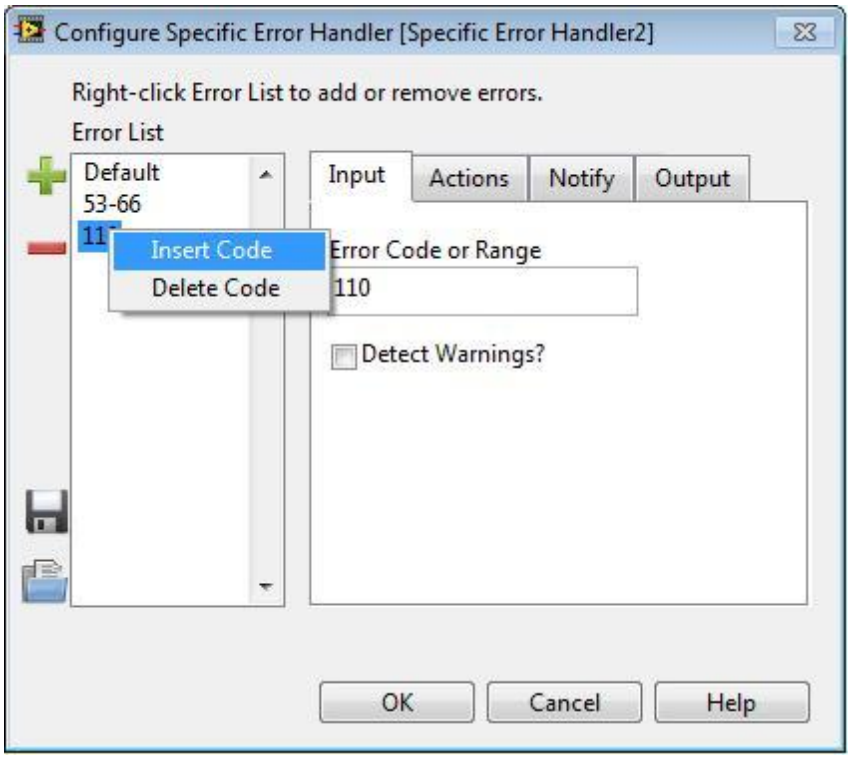
Clear Error - Sets the error output to a "no error" status. See the Ignore section below for more information.

Convert to Warning - Preserves the error code and source, but sets the status to false. See Convert to Warning for more information.

Classify - Classifies the output cluster. Note that this classification applies only to the output and will not be applied to a notification. See the Classify Error section for more details.

Classification - A string description of the type of error. See the Classify Error section for more details. Note that in the Real-Time version of the library the string must be four or less characters. See the Real-Time Error Handler section for more information.

9. Add additional errors by using the green + button or by right-clicking on the Error List and selecting Insert Code. You can also delete errors by selecting Delete Code or using the red - button. To change the settings of a previously configured error, left click it in the list to select the error.



10. Select the Default entry and configure it. The default entry determines what action, if any, the Specific Error Handler takes if it detects an error or warning that is not in the error list.

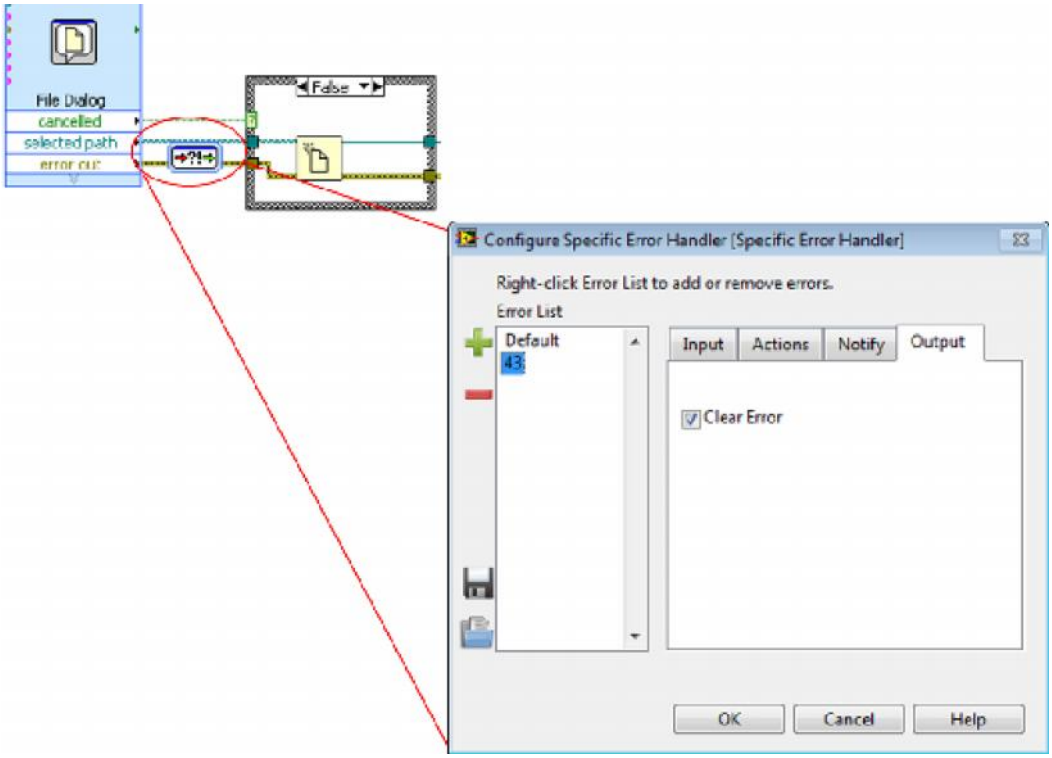
11. Hit OK when finished.

Common Uses

This section provides details and examples for Specific Error Handler functions.

Ignore

Many libraries or functions in LabVIEW throw errors which may be expected behavior in your program. For example, the File Dialog VI returns error 43 when the user cancels a dialog but also returns a cancelled output. If you are handling the cancelled output, you should clear error 43 to prevent it from affecting subsequent functions.

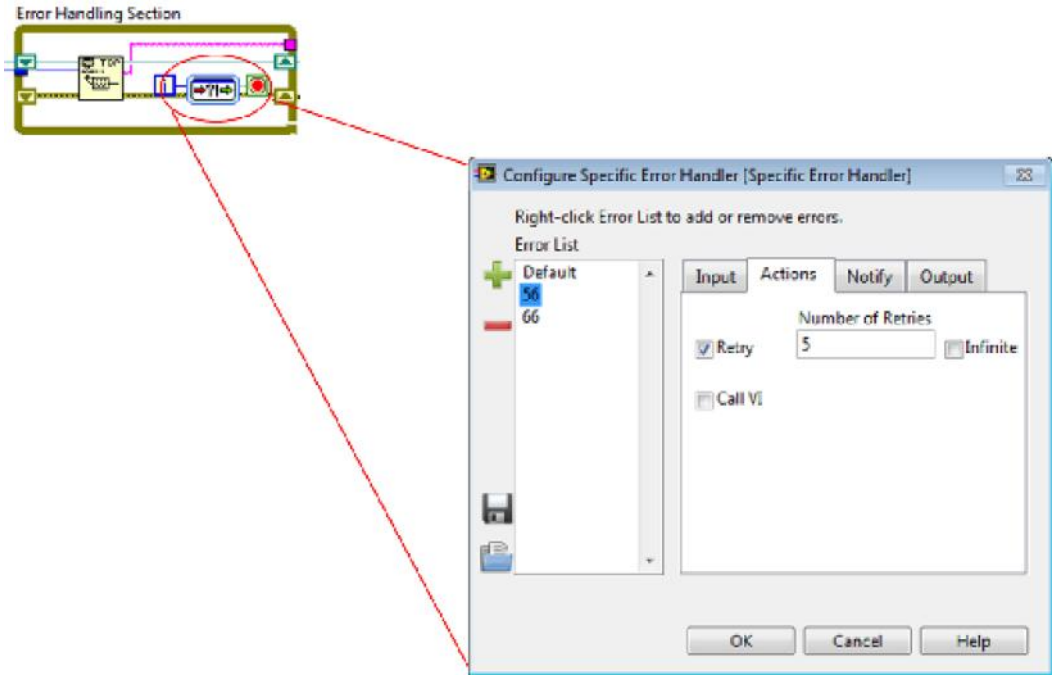


Retry

The Retry option allows you to retry a specific section of code until it works. This is often useful in communication code, where a collision or timing violation may cause a transaction to fail. In many cases, simply retrying the transaction results in success. By implementing these type of simple error corrections, you can substantially increase the reliability and usability of your code.

In order to retry code, you must specify a Number of Retries, which may be infinite. If the code is retried this number of times and does not succeed, the error is preserved and passed out through the output terminal for handling.

Retrying code requires that you place the Specific Error Handler in a while loop along with the code to be retried. The iteration and stop terminals of the loop should be wired to the appropriate inputs of the SEH. The Repeating Error Handler option on the SEH palette will place the Express VI, along with a pre-wired while loop, on your diagram. You may then add you code to the loop as normal.



Convert to Warning

This option will change the status of the error from true to false but preserve all other error cluster fields. This is useful for preserving an

This option will change the status of the error from true to false but preserve all other error cluster fields. This is useful for preserving an error's information for logging or display while preventing it from affecting subsequent code.

Classify Error

The SEH reference library provides the ability to classify errors, a tool to assist you in developing a central error handler. It is undesirable for a central error handler to respond to specific error codes, both because maintaining a list of all errors that could occur in the application (which may include source written by a variety of developers) is difficult, and because error codes may require different responses depending upon where and when they occur. For example, a "file not found" error when opening a critical configuration file implies a very different response than a "file not found" error when opening a log file. Instead of forcing the central error handler to identify specific codes, specific error handlers can notify the central error handler that an action needs to be taken by passing messages embedded within the error source. This is referred to as classifying the error.

Errors can be classified by calling the Classify Error VI or by selecting one of the Classify Error options of the Specific Error Handler Express VI. Either technique requires a string to represent the new classification. The SEH can either implement classifications by adding a special <append> tag to the source field of the error cluster or send the classification along with the error when performing notification. Text in the classification will be displayed on built-in dialogs, such as the Simple Error Handler VI, but will not interfere with the error message or call chain. Classifying an error will replace any existing classification present but will not replace other text using the <append> tag.

Although they add some information to error prompts, error classifications add little value unless a central error handler is created. See the Central Error Handler section for more information.

Call a VI

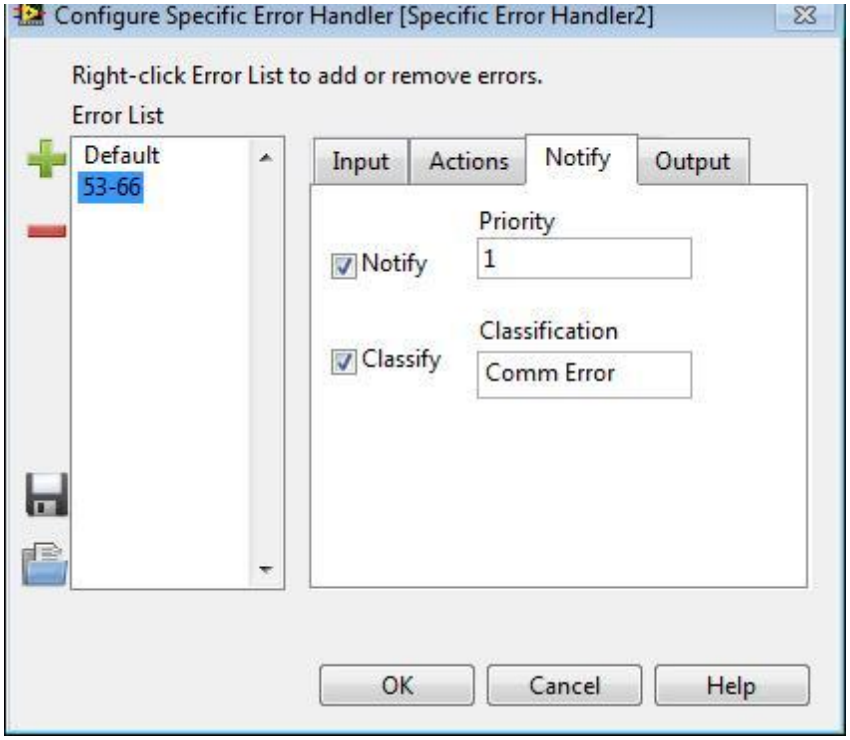
You can dynamically call a VI in response to an error. Use this capability when you need to perform an action other than those implemented in the Specific Error Handler in response to a specific error code.

The VI to call is always specified with a string, and must be spelled, capitalized, and punctuated correctly. As the VI is called dynamically, you will not be notified of any errors until running the error handling code (at which time, you will receive an error from the Internal Error terminal of the Specific Error Handler if the VI to call can not be found). The VI to call must be in memory, and is therefore referenced by the VI's name rather than the VI's path. Entering the path to a VI will result in an error. If you are not otherwise calling the VI in your program, you can load it by using VI server calls or by placing the VI in a case that will never be called.

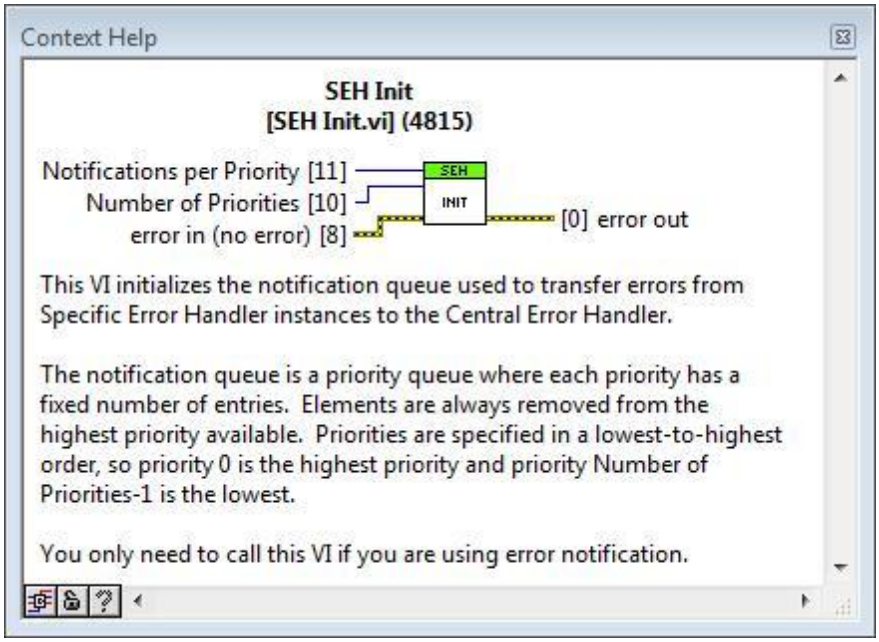
If the VI to call has an error in terminal, the Specific Error Handler will pass the error cluster to the VI to call. If the VI to call has an error out terminal, the Specific Error Handler will replace the error out with the cluster from the VI to call. Using these terminals, you can access and modify the error information. An example of this might be converting one error code to another to provide a more descriptive error message.

Error Communication

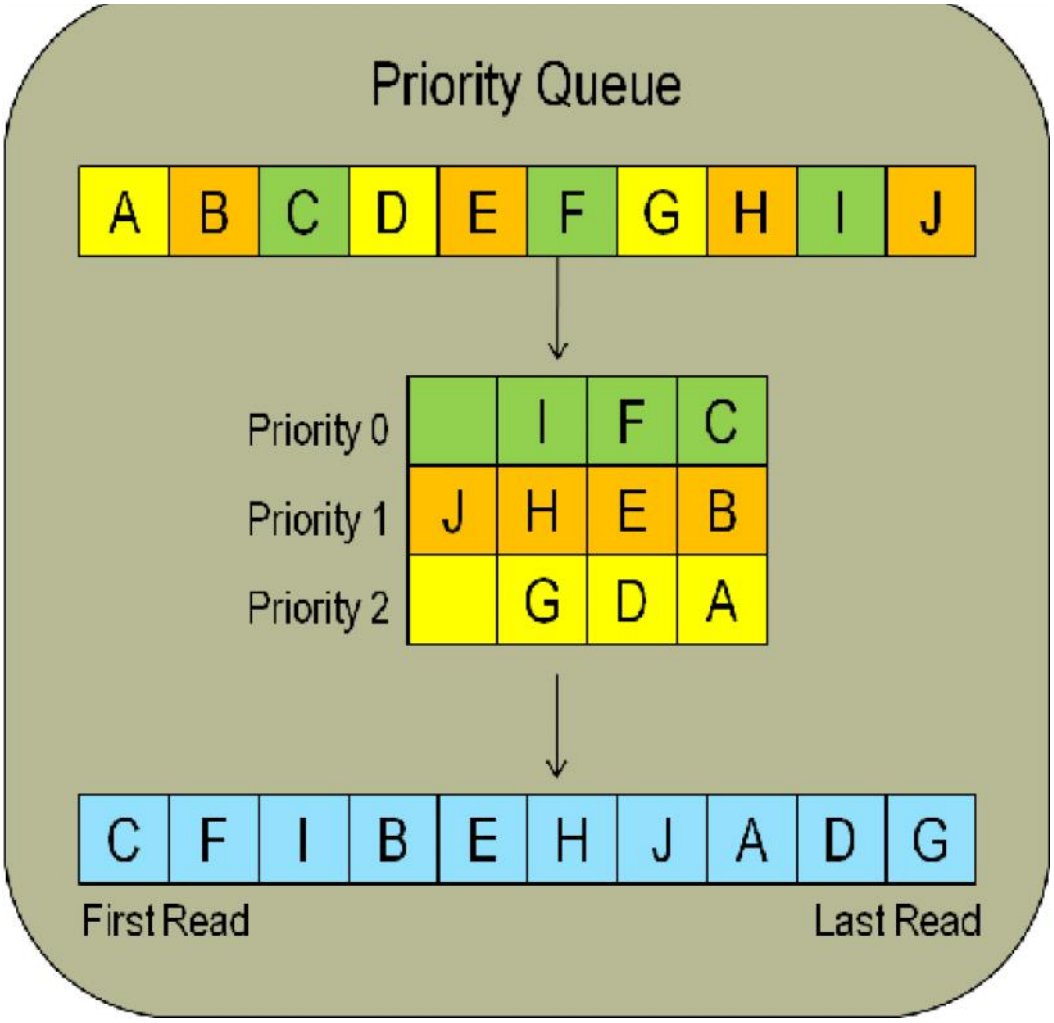
The Notify tab of the Specific Error Handler allows you to enqueue the error to be processed by a central error handler.



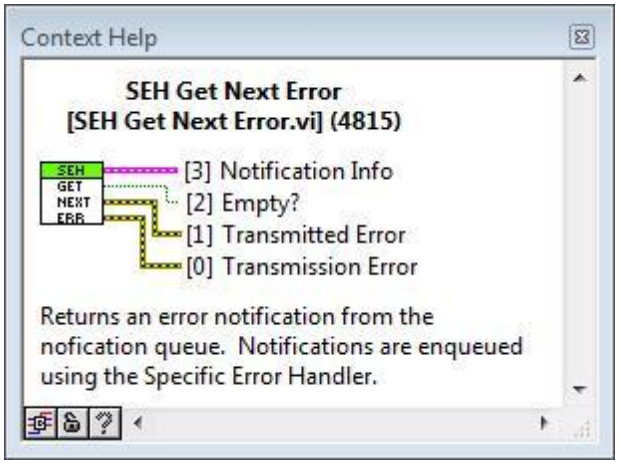
The SEH library uses a Functional Global Variable to implement error communication. The Functional Global Variable must be initialized by calling SEH Init before the first error notification is called. The SEH Init function allows you to specify the size of the error buffer by defining a number of priority levels and the number of errors that each priority level can hold.



The Functional Global Variable implements a simple priority queue. A priority queue is a queue where each enqueued element is given a priority and high priority elements are always returned from the queue before low-priority ones. Two elements with the same priority will be returned in a first-in-first-out (FIFO) manner much like a normal queue. To improve performance, **the priorities for the SEH communication mechanism are implemented in reverse order with 0 being the highest priority and number of priorities - 1 being the lowest.**

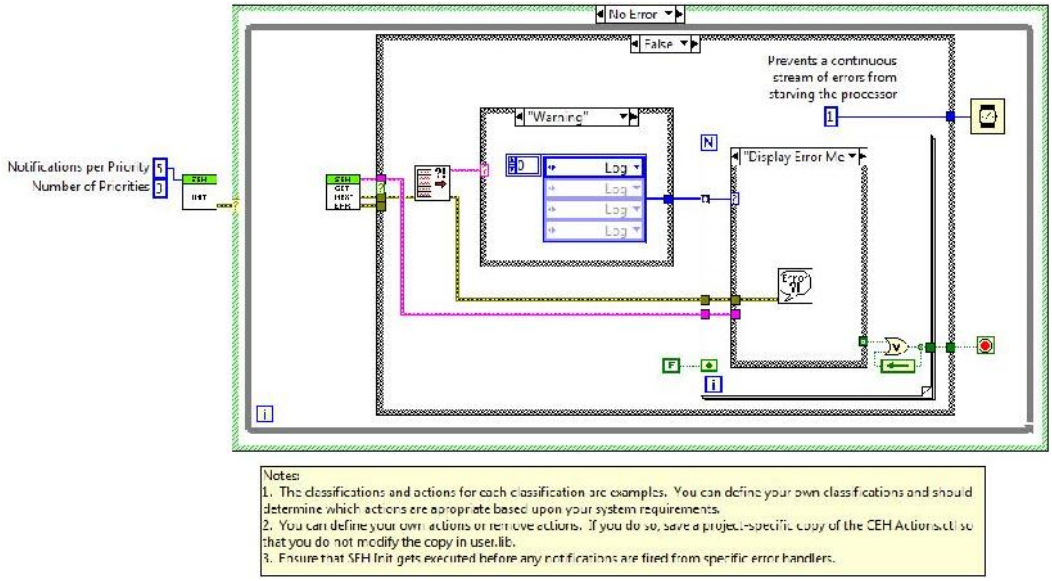


You can dequeue an error from the communication system by calling SEH Get Next error. This VI is already included in the Central Error Handler template.



Central Error Handler

A central error handler consists of high-level code that checks for errors in an entire system. The central error handler should pull errors from the error communication system and use the De-classify error VI to return the error classification string. The central error handler should then use the classification to determine which actions to take. Note that the central error handler does need to know what potential classification strings are possible. Code written with the Specific Error Handler should document which error classifications it uses to aid the developer of the central error handler. The central error handler should also handle the case of an unclassified error or an error with an unrecognized classification and react accordingly. Examples of actions a central error handler might take to respond to error classification (or the lack thereof) include logging the error, displaying error information in an error dialog, sending a message to an HMI such as a touch panel, sending an email/SMS, placing the system outputs in a safe state, and/or initiating a system shutdown/reboot.



The SEH library contains a template for a central error handler that allows you to modify the possible classifications, the actions taken for each classification, and the code executed for each action.

Real-Time Error Handler

The SEH library includes a modified library optimized for handling errors on Real-Time targets. When editing a Real-Time VI, the functions palette for SEH automatically adapts to display the Real-Time version of the library.

Real-Time targets must often handle and communicate errors from time-critical code, they often have more constrained processor resources than other targets, and real-time targets should minimize dynamic memory allocation to reduce memory fragmentation. The real-time error handler (RTEH) attempts to operate within these constraints at the cost of certain functionality.

While the RTEH is recommended for real-time applications and the SEH is recommended for other applications, either error handler can technically execute on any target. Some non-RT applications which need to minimize processor usage may use the RTEH, whereas some RT applications with lower performance requirements and a need for more robust functionality may use SEH.

For most purposes the RTEH behaves exactly as the SEH, with the exception of the following changes:

Source Strings in RTEH

When sending error notifications, the RTEH discards the error source string to avoid dynamically allocating memory in the priority queue.

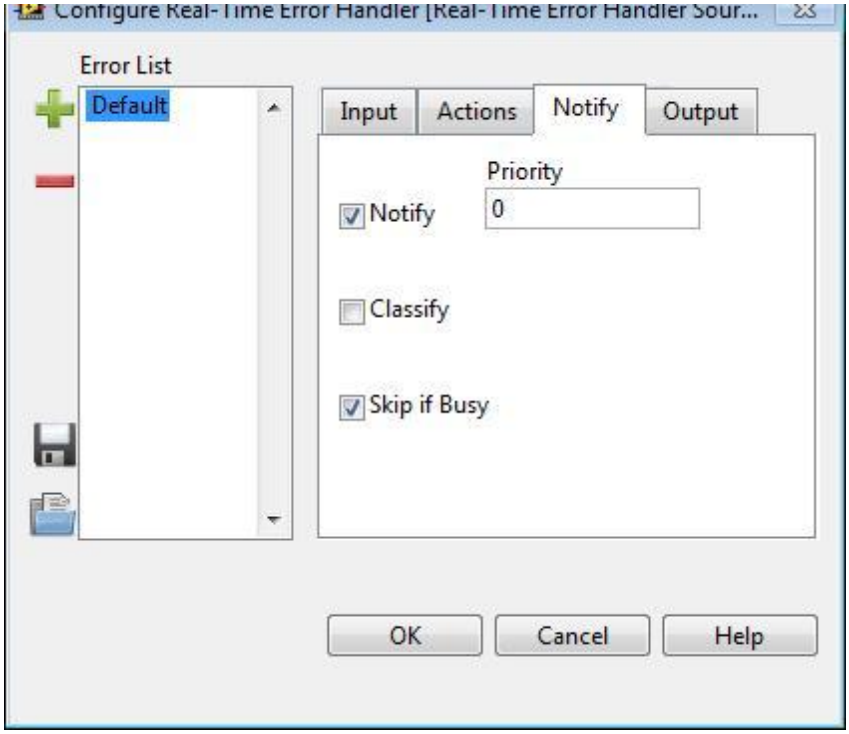
Real-Time Error Classification

To avoid dynamically-sized string manipulation, the RTEH accepts only 4 character strings for classifications. Within the notification system, these strings are transmitted as 32-bit integers. Combined with the lack of source strings, as described above, this prevents the notification system from ever having to allocate memory after the SEH Init VI is called. RTEH classifications are transmitted and returned separately from the error cluster rather than embedded in the <append> tag of the source string.

Calling VIs in RTEH

Because calling a VI would involve a dynamic call that could introduce significant jitter into a program, the RTEH does not have the Call a VI capabilities of the SEH.

Skip If Busy



The error communication system represents a shared resource; therefore, the RTEH adds an additional option to the Notification tab that indicates whether the RTEH should skip the notification operation if the shared resource is in use. Use this option only to avoid introducing jitter into time-critical loops. Note that using this may cause you to lose error notifications unless you monitor the Internal Status output of the Real-Time Error Handler and buffer any skipped notifications until the shared resource is free.

Options

The SEH library contains settings that are configured by using Conditional Disable Symbols. You can set these through the configuration of a target in the LabVIEW Project. The following symbols are defined.

SEH_XMIT_ERR

Determines how to return any internal errors when transmitting an error.

RECV - When checking a notification, the last transmit error is returned and then cleared.
SEND - A transmit error will be returned to the sender via the Transmission Error terminal of the SEH express VI.
BOTH - Applies the effects of both SEND and RECV. (default)
NONE - Transmit errors are not reported.

SEH_DEC_PR

Determines how to handle a full priority in the priority queue.

TRUE - When sending a notification, if the requested priority is full, attempt to send the notification with a lower priority. (default)
FALSE - Notifications will only be sent with the requested priority.
WARN - Same functionality as TRUE, but a warning will be returned the next time a notification is checked. A warning is never returned to the sender (even if SEH_XMIT_ERR = BOTH). This option is only meaningful if SEH_XMIT_ERR is RECV or BOTH.

Internal Errors

The SEH library can throw the following errors:

537600 - Invalid command or option selected. This may indicate that enumerated types are out of sync.
537601 - The error notifier queue is full. The send error request failed.
537602 - The requested priority value is not valid. Valid priorities range from 0 to num of priorities - 1.
537603 - The requested priority is full. The notification was transmitted at a lower priority. (warning)

Feedback

This reference application was created by the NI Systems Engineering group.

This reference application was created by the NI Systems Engineering group.

We **do not** regularly monitor Reader Comments posted on this page.

Please submit your feedback in the SEH Reference Library discussion forum so that we can improve this component for future applications.

Please direct support questions to NI Technical Support.

Requirements

Filename: ni_seh-2.0.6.9.vip

Software Requirements

Application Software: LabVIEW Professional Development System 2010 SP1
Language(s): LabVIEW

Filename: seh_1_0_1.zip

Software Requirements

Application Software: LabVIEW Full Development System 2009
Language(s): LabVIEW