Systems Alliance

VPP-3.3: Instrument Driver Interactive Developer Interface Specification

Revision 5.1

April 14, 2008

VPP-3.3 Revision History

This section is an overview of the revision history of the VPP-3.3 specification.

Revision 1.0, May 1, 1995

Original draft.

Revision 2.0, February 2, 1996

Updated and moved sections from VPP-3.2 on function panel usage.
Added new instrument model.
Changed allowable number of levels in function panel tree.
Added section on G Frameworks interactive Developer's Interface.
Technical changes agreed upon in Technical Working Group.

Revision 3.0, December 4, 1998

Added rules regarding the FP Auto Load List feature and the search path for function panel files. Also, added chapter 7 which specifies the `.sub` file format. Information regarding contacting the alliance was updated. References to the VPP-5 Component Knowledge Base specification, which was obsoleted by the alliance, were removed.

Revision 4.0, December 7, 1999

Updated the function panel file format section to include a 5.1 version of the .fp file format. Also updated the .sub file format section to (a) include an overview and (b) add rules and recommendations for ADE use.

Revision 4.0.1 Draft, November 30, 2000

Removed reserved4 field in the FunctionTreeHdr field for ver 5.1fp file formats. Removed reference to MAX_FUNTREE_NAME_LEN is in description column for FunctionTreeHdr found in *Observation 6.2* of *section 6.8: File Record Formats.*

Document formatting changes: In Revision History, moved the line "Revision 4.0.1 November 30, 2000" to the next page so it is with its descriptive paragraph; Changed footers throughout to Revision 4.0.1; Changed page numbers in headers to include chapter number with page number; Regenerated Table of Contents and added automated numbering to tables and figures; Deleted manual section numbers for 2.8, 3.2.2., and 3.3 because they conflicted with auto-generated section numbers in table of contents.

Revision 4.0.1 Draft, July 26, 2001

Fixed typo – the date on the cover pages needed to be consistent with each other.

Revision 4.0.1, December 13, 2001

Removed the word "Draft" from the document and modified the date to show when it was final.

Revision 5.0, October 30, 2006

Updated the function panel file format section to include a 9.0 version of the .fp file format. The new .fp file format provides support for 64-bit integers. Also updated the .sub file format section to support 8 levels of hierarchy and 64-bit integers.

Revision 5.1, February 14, 2008

Updated the introduction to reflect the IVI Foundation organization changes. Replaced Notice with text used by IVI Foundation specifications..

Revision 5.1, April 14, 2008

Editorial change to update the IVI Foundation contact information in the Important Information section to remove obsolete address information and refer only to the IVI Foundation web site.

NOTICE

VPP-3.3: *Instrument Driver Interactive Developer Interface Specification* is authored by the IVI Foundation member companies. For a vendor membership roster list, please visit the IVI Foundation web site at `www.ivifoundation.org`.

The IVI Foundation wants to receive your comments on this specification. You can contact the Foundation through the web site at `www.ivifoundation.org`.

Warranty

The IVI Foundation and its member companies make no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The IVI Foundation and its member companies shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Trademarks

Product and company names listed are trademarks or trade names of their respective companies.

No investigation has been made of common-law trademark rights in any work.

Contents

Figures

Tables

Section 1
Introduction to the VXI*plug&play* Systems Alliance and the IVI Foundation

The VXI*plug&play* Systems Alliance was founded by members who shared a common commitment to end-user success with open, multivendor VXI systems. The alliance accomplished major improvements in ease of use by endorsing and implementing common standards and practices in both hardware and software, beyond the scope of the VXIbus specifications. The alliance used both formal and de facto standards to define complete system frameworks. These standard frameworks gave end-users "plug & play" interoperability at both the hardware and system software level.

The IVI Foundation is an organization whose members share a common commitment to test system developer success through open, powerful, instrument control technology. The IVI Foundation's primary purpose is to develop and promote specifications for programming test instruments that simplify interchangeability, provide better performance, and reduce the cost of program development and maintenance.

In 2002, the VXI*plug&play* Systems Alliance voted to become part of the IVI Foundation. In 2003, the VXI*plug&play* Systems Alliance formally merged into the IVI Foundation. The IVI Foundation has assumed control of the VXI*plug&play* specifications, and all ongoing work will be accomplished as part of the IVI Foundation.

All references to VXI*plug&play* Systems Alliance within this document, except contact information, were maintained to preserve the context of the original document.
.

Section 2
Overview of Instrument Driver Interactive Developer Interface Specification

2.1 Introduction

This section summarizes the *Instrument Driver Interactive Developer Interface Specification* and contains general information that the reader may need in order to understand, interpret, and implement individual aspects of this specification. These aspects include the following:

- The objectives of this specification
- The audience for this specification
- The scope and organization of this specification
- Application of this specification
- References
- Definitions of terms and acronyms
- Conventions

2.2 Objectives of This Specification

The *Instrument Driver Interactive Developer Interface Specification* defines the Interactive Developer Interface as well as the implementation of the required functions.

2.3 Audience for This Specification

This specification has two audiences. The first audience consists of instrument driver developers--either instrument vendors, system integrators, or end users--who want to implement instrument driver software that is compliant with this specification. The second audience consists of instrumentation end users and application programmers who want to implement applications that use instrument drivers compliant with this specification.

2.4 Scope and Organization of This Specification

This specification is organized in sections. Each section discusses a particular aspect of the VXI*plug&play* Systems Alliance standard for instrument drivers.

Section 1 explains the VXI*plug&play* Systems Alliance and its relation to the IVI Foundation.

Section 2 summarizes this specification and discusses its objectives, scope and organization, application, references, definition of terms and acronyms, and conventions.

Section 3 defines the way a function panel should be designed by the driver writer.

Section 4 shows the layout of each of the template functions required of all VXIplug&play drivers for the WINNT framework.

Section 5 shows the layout of each of the template functions required of all VXIplug&play drivers for the GWINNT framework.

Section 6 defines the format of the function panel file.

2.5 Application of This Specification

This specification is intended to be used by developers of VXI*plug&play* instrument drivers. It is also useful as a reference for end users of VXI*plug&play* instrument drivers. This specification is intended to be used in conjunction with the *Instrument Drivers Architecture and Design Specification* (VPP-3.1), the *Instrument Driver Developers Specification* (VPP-3.2), the *Instrument Driver Programmer Interface Specification* (VPP-3.4), and the *VISA Library Specifications* (VPP-4.x). These related specifications describe the implementation details for specific instrument drivers that are used with specific system frameworks. VXI*plug&play* instrument drivers developed in accordance with these specifications can be used in a wide variety of higher-level software environments, as described in the VXI*plug&play System Frameworks Specification* (VPP-2).

2.6 References

Several other VXI*plug&play* Systems Alliance documents and specifications are related to this specification. These other related documents include the following:

- VPP-1, VXI*plug&play* Charter Document
- VPP-2, *System Frameworks Specification*
- VPP-3.1, *Instrument Drivers Architecture and Design Specification*
- VPP-3.2, *Instrument Driver Functional Body Specification*
- VPP-3.4, *Instrument Driver Programmatic Developer Interface Specification*
- VPP-4.*X*, *VISA Specifications*

- VPP-6, *Installation and Packaging Specification*
- VPP-7, *Soft Front Panel Specification*
- VPP-9, *Instrument Vendor Abbreviations*

2.7 Definitions of Terms and Acronyms

The following are some commonly used terms within this document.

• ADE	Application Development Environment
• LabWindows/CVI	C-based ADE
• LabVIEW	Graphical programming ADE
• Agilent VEE	Graphical programming ADE
• Instrument Driver	Library of functions for controlling a specific instrument
• Template or Required Function	Instrument Driver function required to be implemented in all VXI*plug&play* instrument drivers.
• Application Function	A high-level, test-oriented, instrument driver function. It is typically developed from the instrument driver subsystem functions.
• VISA	Virtual Instrument Software Architecture
• VI	LabVIEW program or Virtual Instrument
• LLB	LabVIEW VI library
• DLL	Dynamic Link Library

2.8 Conventions

The following headings appear on paragraphs throughout this specification. These headings give special meaning to the paragraphs.

Rules must be followed to ensure compatibility with the system framework. A rule is characterized by the words **SHALL** or **SHALL NOT** in bold upper case characters. These words are not used in this manner for any other purpose.

Recommendations contain advice to implementors. This advice affects the usability of the final device. Recommendations are included in this standard to draw attention to particular characteristics that the authors believe to be important to end-user success.

Permissions authorize specific implementations or uses of system components. A permission is characterized by the word **MAY** in bold upper case characters. These permissions are granted to ensure that specific system framework components are well defined and can be tested for compatibility and interoperability.

Observations spell out implications of rules and bring attention to details that might otherwise be overlooked. They also give the rationale behind certain rules so that the reader understands why the rule should be followed.

A Note on the text of the specification: Any text that appears without heading should be considered a description of the standard and how the architecture was intended to operate. The purpose of this text is to give the reader a deeper understanding of the intentions of the specification, including the underlying model and specific required features. The implementor of this standard should ensure that a particular implementation does not conflict with the text of the standard.

Section 3
Function Panel Requirements for the Instrument Driver Writer

3.1 Introduction

The Function Panel Interface is implemented by instrument drivers within the WINNT framework. The function panel provides a visual representation of the C language function interface to the instrument.

This section discusses how the function panel interface is implemented by the driver writer. While other sections discuss the internal structure of the function panel file and are primarily of use to those implementing applications that read and edit function panels, this section provides guidelines and requirements for creating function panels.

3.2 Function Panel Tree Structure

A function panel tree is not a flat structure, but rather one that groups various functions. This grouping provides valuable information to the user. Standardizing on a particular tree organization allows users to learn the structure quickly and thus is more friendly.

The driver's function panel is organized around the model presented in VPP-3.1:

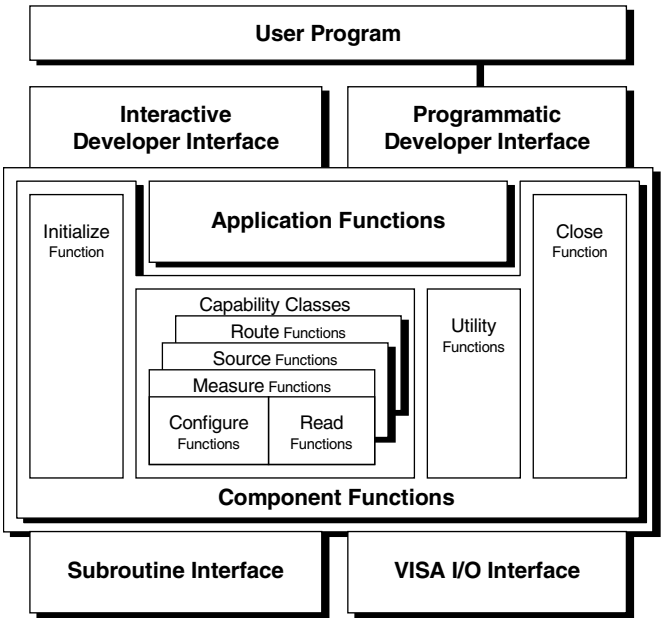

Figure 3-1 Instrument Driver Internal Design Model

3.2.1 Notation Conventions

Tree levels will be designated by indentation. A grouping of functions (classes) will be designated by italicizing the node. A function is specified by non-italic printing. Angle brackets <> designate groupings of functions or classes.

3.2.2 Minimal Function Tree Organization

The minimal function tree contains the template functions required in VPP-3.1. These functions are init, close, error_message, error_query, reset, self_test, and revision_query. The minimal function tree will also contain at least one capability class, and at least one application function. The structure of this minimal set is described in the following rule:

RULE 3.1

The minimal function panel tree **SHALL** have the following organization, i.e. the functions **SHALL** be present and be listed in the order shown:

Table 3-1 Minimal Function Panel Tree Structure

Tree Node	Required Function (from VPP-3.1)
Initialize	PREFIX_init
Application Functions	
<application fns>	PREFIX_<device dependent>
<*Capability Class*>	
Utility	
Error Message	PREFIX_error_message
Error Query	PREFIX_error_query
Reset	PREFIX_reset
Self Test	PREFIX_self_test
Revision Query	PREFIX_revision_query
Close	PREFIX_close

3.2.3 Extending the Function Tree

As the function tree is extended, nodes may be added to the minimal function tree. These nodes correspond to the additional functions defined. These nodes must be extended in a manner that follows the model described above. The nodes corresponding to this model are described by the following rule. It is unlikely that any particular instrument driver would implement all of the nodes shown below, but the nodes must be placed in this structure and intermediate nodes used as appropriate.

PERMISSION 3.1

A class may be added after the template function in all cases that relate to that function.

RULE 3.2

As the function tree is extended, additional functions **SHALL** be placed into the tree so as to follow the following model.

Table 3-2 Extended Function Panel Tree Structure

Tree Node	Description
Initialize	PREFIX_init
Initialize	
<other initialize functions>	
Application Functions	
<Application functions>	(common usage functions. Value added above instrument's command set)
Measure	
<high level measurement functions>	(abstraction of instrument's command set)
Configure Measurement	
<high level configuration functions>	
Low Level Configure	
< configuration functions >	(low-level configuration functions, including measurement functions, triggering, and so on)
Read	
<high-level read >	(functions that initiate and fetch data)
Initiate	
<initiate functions >	(function that initiates the reading)
Fetch	
<fetch functions>	
Route	
< route functions>	(high-level signal routing functions)
Configure Route	
<configure functions>	
Initiate	
<initiate functions>	
Source	
<high level source functions>	(apply a signal)
Configure Source	
<configure functions >	(low-level source configuration functions)
Initiate	
<initiate functions>	
Utility	
Reset	PREFIX_reset
Self Test	PREFIX_self_test
Error Query	PREFIX_error_query
Error Message	PREFIX_error_message
Revision Query	PREFIX_revision_query
<other utility functions>	
Close	PREFIX_close

3.3 Function Panel Layout

This section describes the general layout of the function panel. Actual size of the panels is determined by the application and is beyond the scope of this section. Likewise, individual positions are only generally specified to allow maximum flexibility in panel layout, while still providing a consistent look and feel.

All VXI*plug&play* functions return a value of type ViStatus. Likewise, the first parameter of most functions is the ViSession pointer (see VPP-3.4 section 3.6.2). These parameters are consistently placed on the function panel for consistency and ease of use.

RULE 3.3

The visual representation of the session handle of a VXI*plug&play* function, when present, **SHALL** be placed in the lower left hand corner of that function's function panel. The control **SHALL** be labeled "Instrument Handle." If the first parameter of a VXI*plug&play* function is not of type ViSession, nothing **SHALL** be placed in the lower left hand corner of that function panel.

RULE 3.4

The visual representation of the return value of a VXI*plug&play* function **SHALL** be placed in the lower right hand corner of that function's function panel. The control **SHALL** be labeled "Status."

RECOMMENDATION 3.1

The visual representation of each subsequent parameter of a VXI*plug&play* function should be placed starting in the upper left hand corner, with successive parameters appearing left to right, top to bottom.

OBSERVATION 3.1

Labels on function panel controls are not specified (except for template functions) and do not need to reflect formal parameter names of the function prototype.

3.4 Function Panel Restrictions

Some, but not all ADEs allow multiple functions to be grouped on a function panel window. Thus this feature should not be used.

RULE 3.5

A function panel window **SHALL** have exactly one function panel.

RULE 3.6

The parameter type of each parameter of a VXI*plug&play* function panel **SHALL** exactly match the type described in the C function, except for arrays, which **SHALL** use only the [ ] syntax.

RECOMMENDATION 3.2

A parameter of a VXI*plug&play* function that represents an enumerated selection **SHALL** be represented by either a slide, binary, or ring control.

RULE 3.7

A VXI*plug&play* function panel **SHALL NOT** implement global variables.

RULE 3.8

A VXI*plug&play* function panel **SHALL NOT** implement the Autoloadlist feature described in Section 6 of this document until June 1, 1996.

RULE 3.9

If a VXI*plug&play* function panel contains controls whose values are enumerations, the definition of these enumerations **SHALL** be placed in either PREFIX.h, VISATYPE.h, or VPPTYPE.h.

RULE 3.10

A control within a VXI*plug&play* function panel **SHALL** be restricted with respect to its associated data type according to the following table:

Table 3-3 Allowed Function Panel Data Types Per Control

Control Type	Allowed Data Types
Ring Associative, Slide, Binary	`ViInt64, ViInt32, ViInt16, ViBoolean, ViReal64, ViString, ViRsrc, ViConstString, ViAttr`
Output	Any `Vi` type by reference except `ViConstString`
Return	`ViStatus`
Numeric	`ViInt64, ViInt32, ViInt16, ViReal64,`
Input	Any `Vi` type

3.5 Function Panel Help Information

The on-line help of an instrument function panel is a frequently used source of documentation. Relevant help information in a consistent format increases the ease of use of the instrument driver. Include on-line help at every level of the instrument driver.

3.5.1 Function Panel Help

Function Panel help describes the instrument driver and lists the functions and classes of functions in the driver.

RULE 3.11

All Function Panel implementations **SHALL** implement instrument driver help.

3.5.2 Function Class Help

Function class help, describes the subfunctions and subclasses that are beneath the selected function class.

RULE 3.12

All Function Panel implementations **SHALL** implement function class help.

3.5.3 Function Panel Help

Function panel help describes the function call and any associated secondary functions on the panel.

RULE 3.13

All Function Panel implementations **SHALL** implement function panel help.

3.5.4 Control Help

Control help lists the parameter name and describes the parameter, variable type, valid range, and default value.

RULE 3.14

All Function Panel implementations **SHALL** implement control help.

RECOMMENDATION 3.3

If the control represents an enumerated parameter, the enumerations should be documented here. Likewise, if a parameter is bound to a #DEFINE, those bindings should also be specified here.

3.5.5 Return Help

Return value help describes the return value variable type, and possible error values.

RULE 3.15

All Function Panel implementations **SHALL** implement return value help.

Section 4
Layout of Function Panels for Template Functions for WINNT Framework

This section shows the layout of each of the template functions required of all VXI*plug&play* drivers. All function panels for these functions must be laid out in a manner consistent with the definitions shown here.

OBSERVATION 4.1

Exact appearances of controls and panels may differ slightly between ADEs. It is not advisable to rely on exact character spacing when laying out a function panel because of font or control size differences.

OBSERVATION 4.2

The default values for parameters should be set to values appropriate for the particular instrument and function.

4.1 Init Function

RULE 4.1

The function panel for the PREFIX_init template function **SHALL** be implemented as follows:

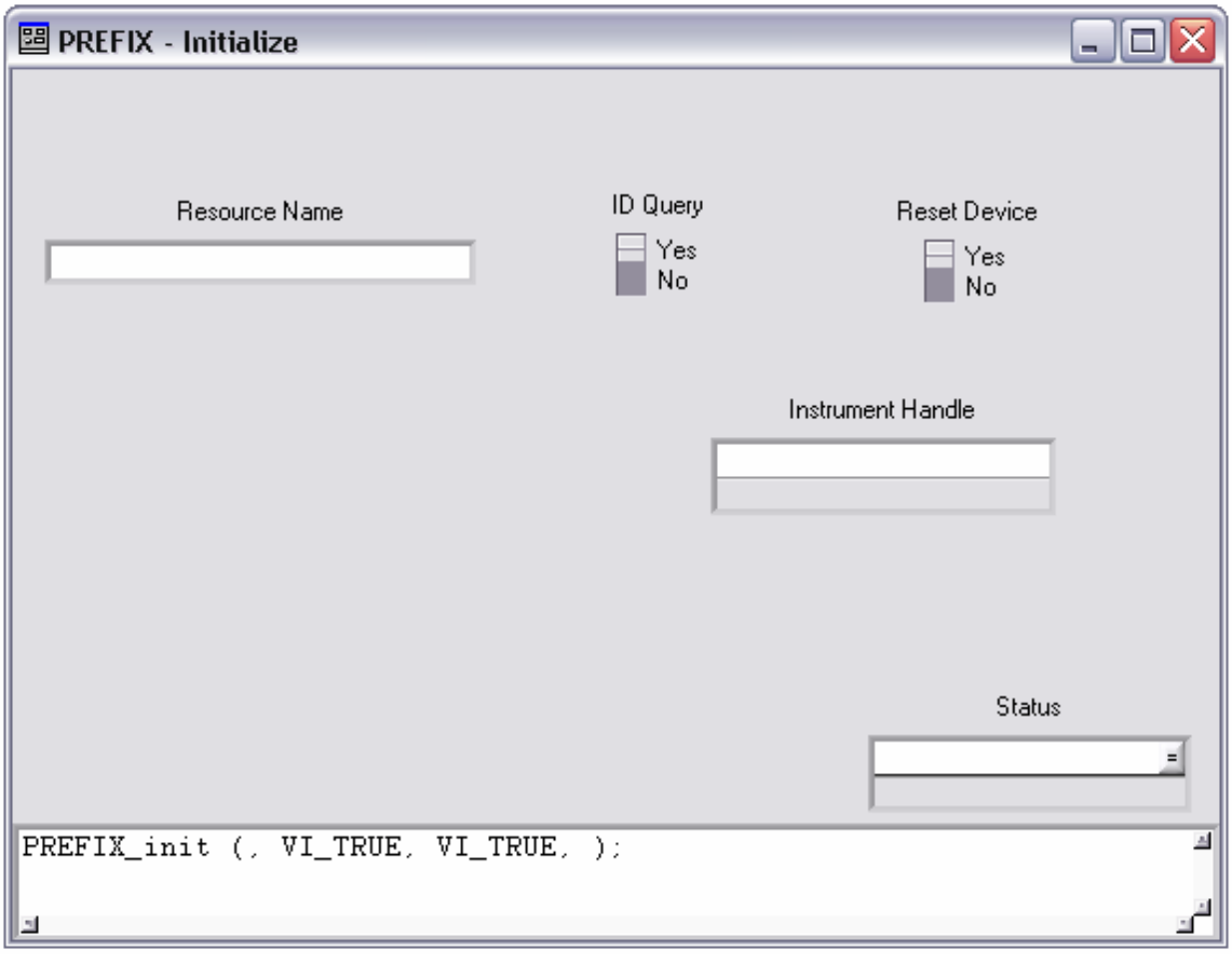

Figure 4-1 PREFIX_init Function Panel Layout

4.2 Reset Function

RULE 4.2

The function panel for the PREFIX_reset template function **SHALL** be implemented as follows:

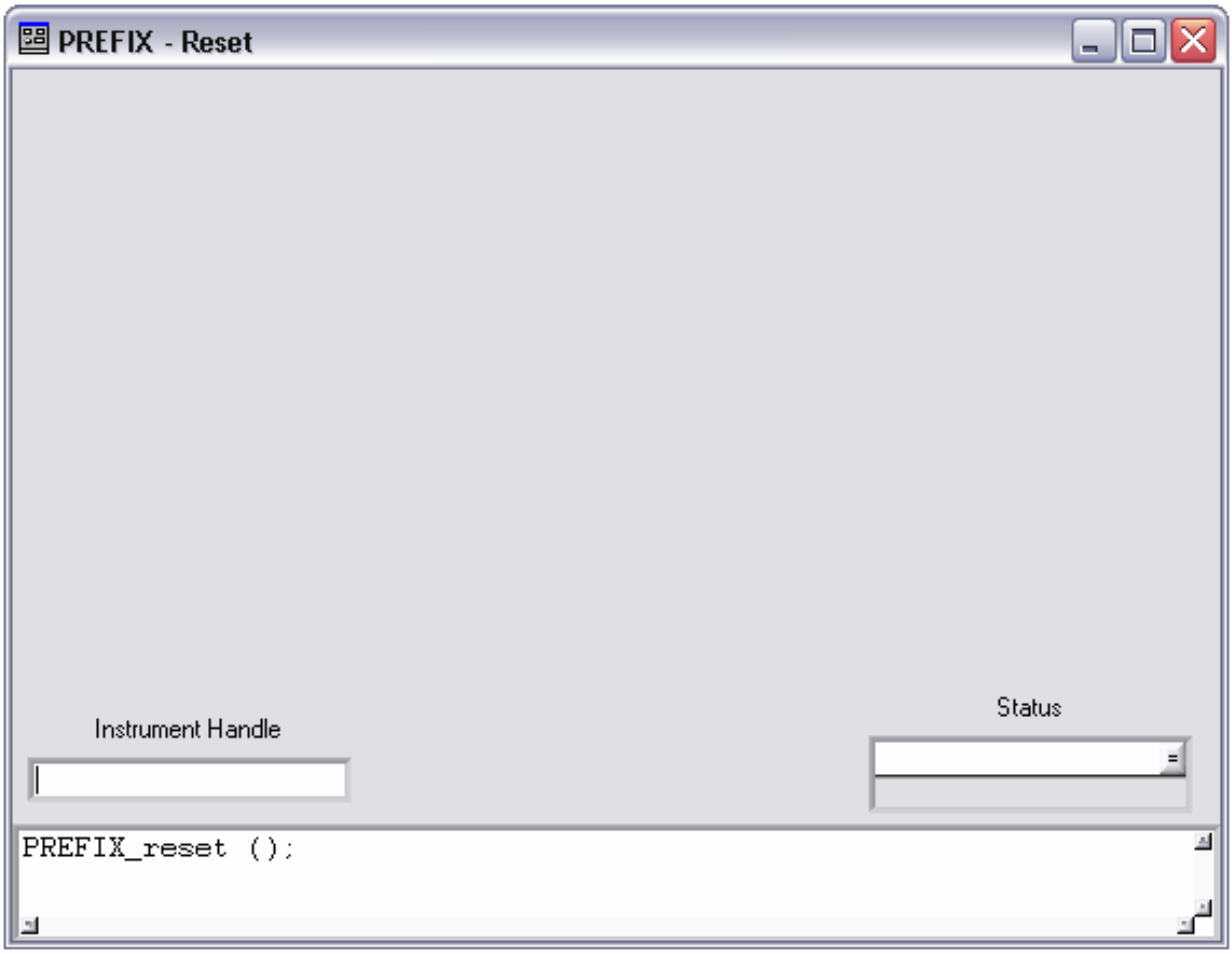

Figure 4-2 PREFIX_reset Function Panel Layout

4.3 Self Test Function

RULE 4.3

The function panel for the PREFIX_self_test template function **SHALL** be implemented as follows:

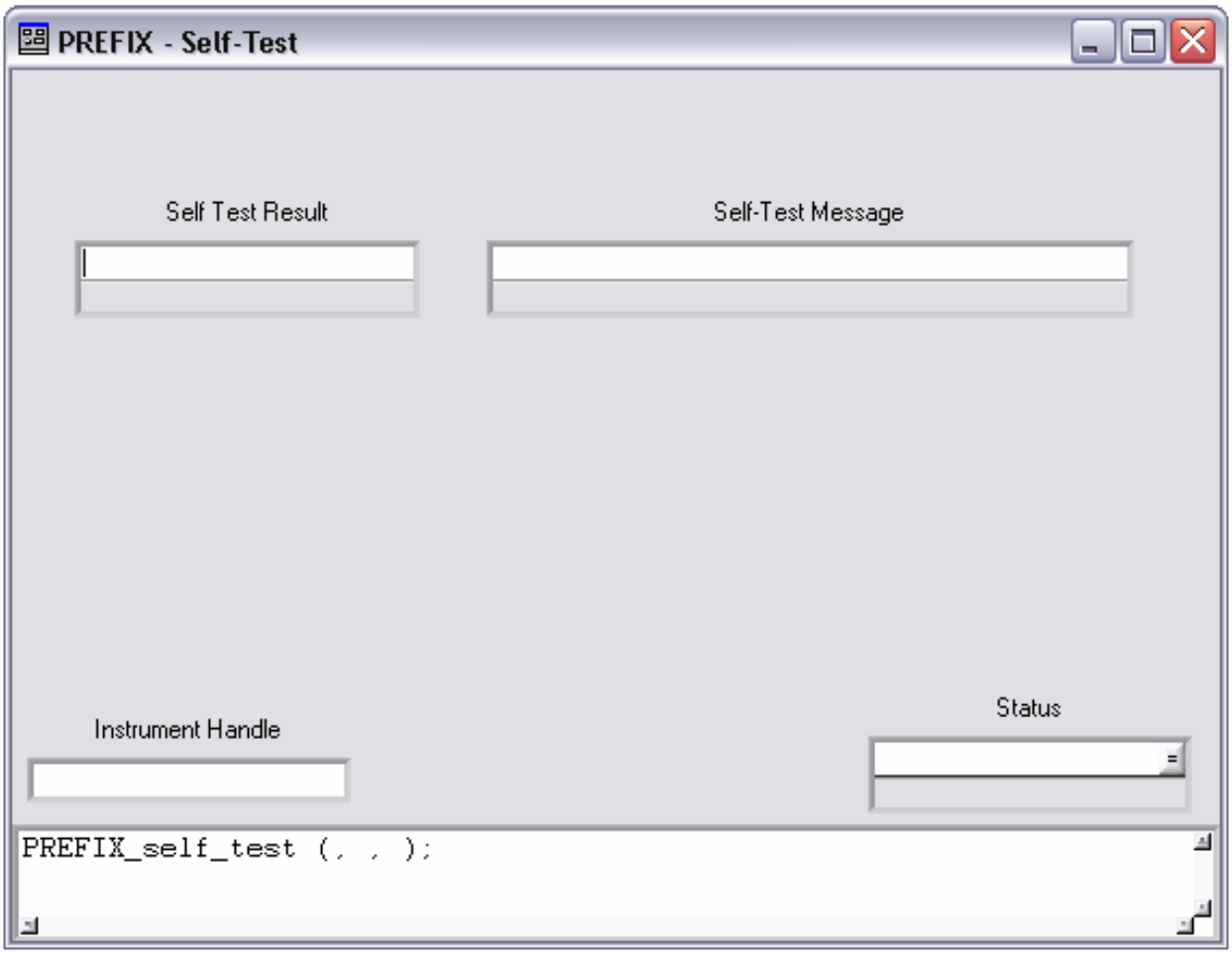

Figure 4-3 PREFIX_self_test Function Panel Layout

4.4 Error Query Function

RULE 4.4

The function panel for the PREFIX_error_query template function **SHALL** be implemented as follows:

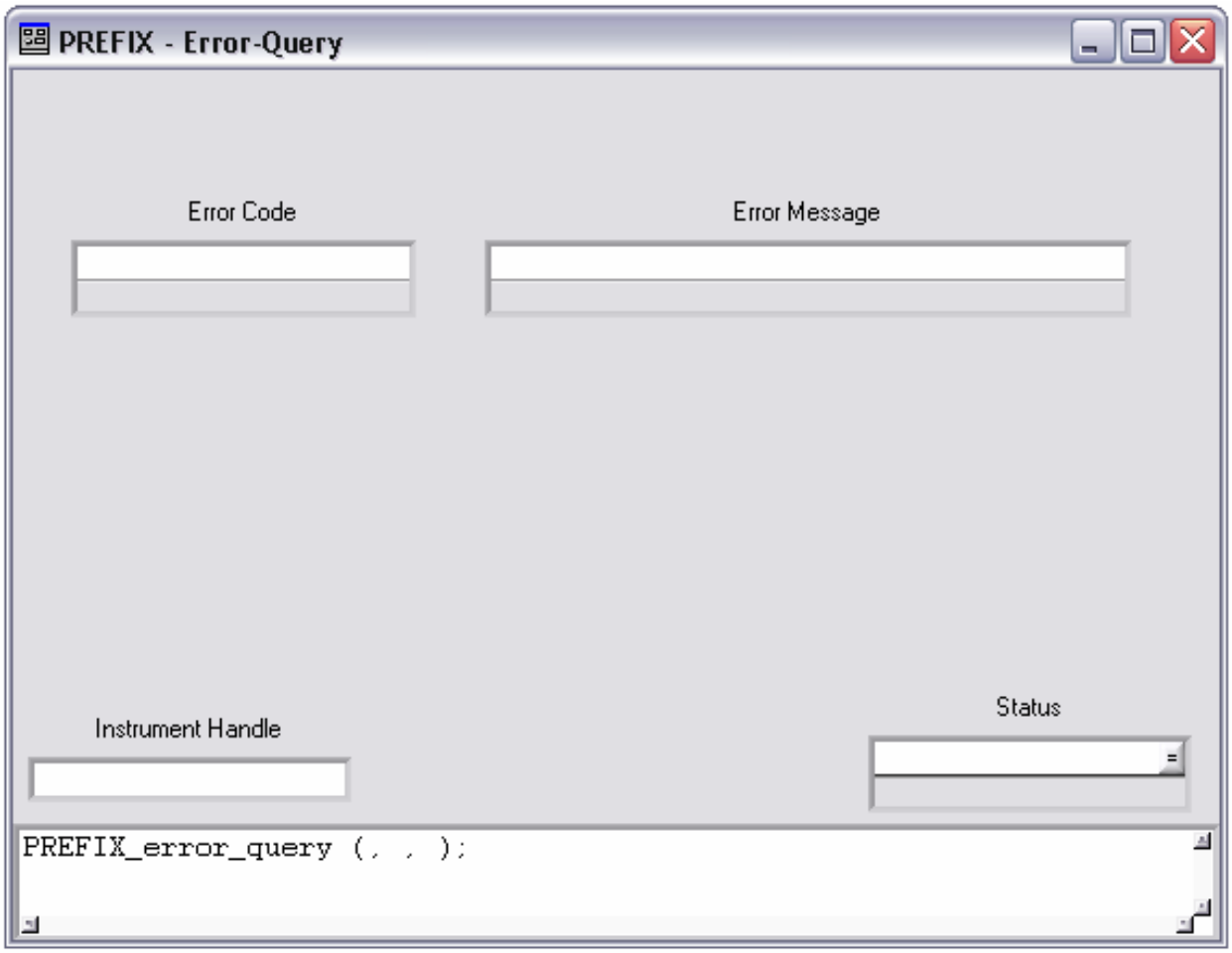

Figure 4-4 PREFIX_error_query Function Panel Layout

4.5 Error Message Function

RULE 4.5

The function panel for the PREFIX_error_message template function **SHALL** be implemented as follows:

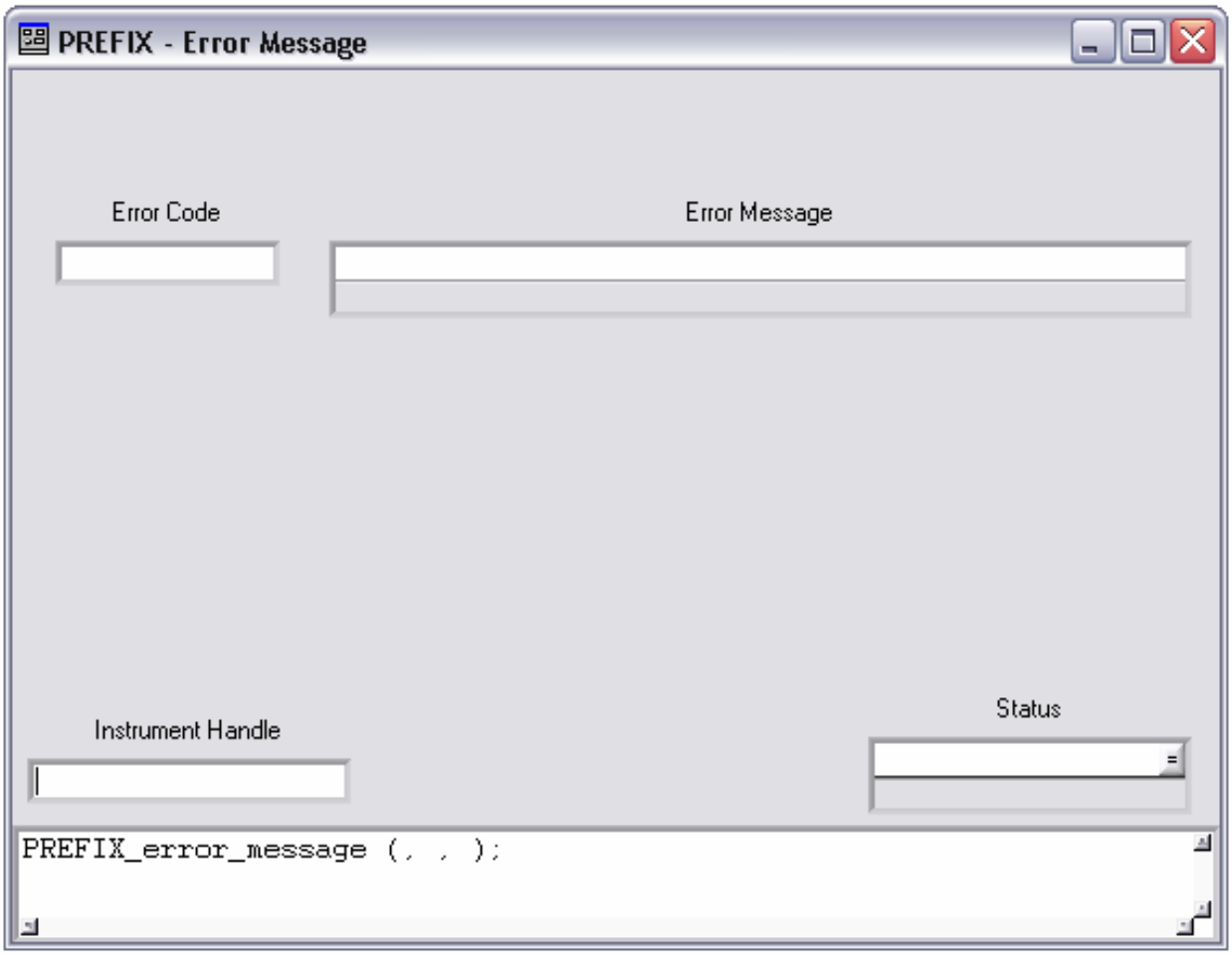

Figure 4-5 PREFIX_error_message Function Panel Layout

4.6 Revision Query Function

RULE 4.6

The function panel for the PREFIX_revision_query template function **SHALL** be implemented as follows:

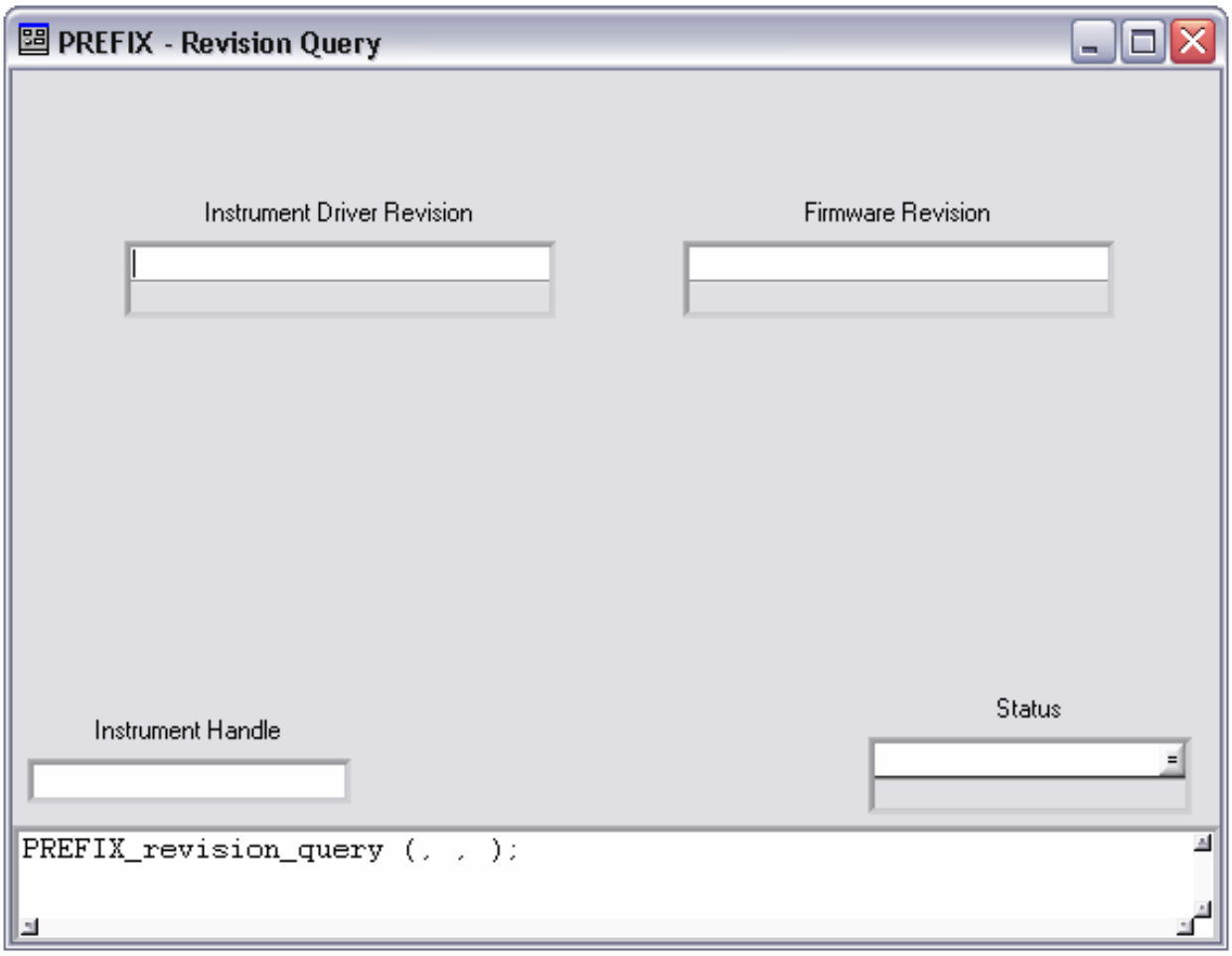

Figure 4-6 PREFIX_revision_query Function Panel Layout

4.7 Close Function

RULE 4.7

The function panel for the PREFIX_close template function **SHALL** be implemented as follows:

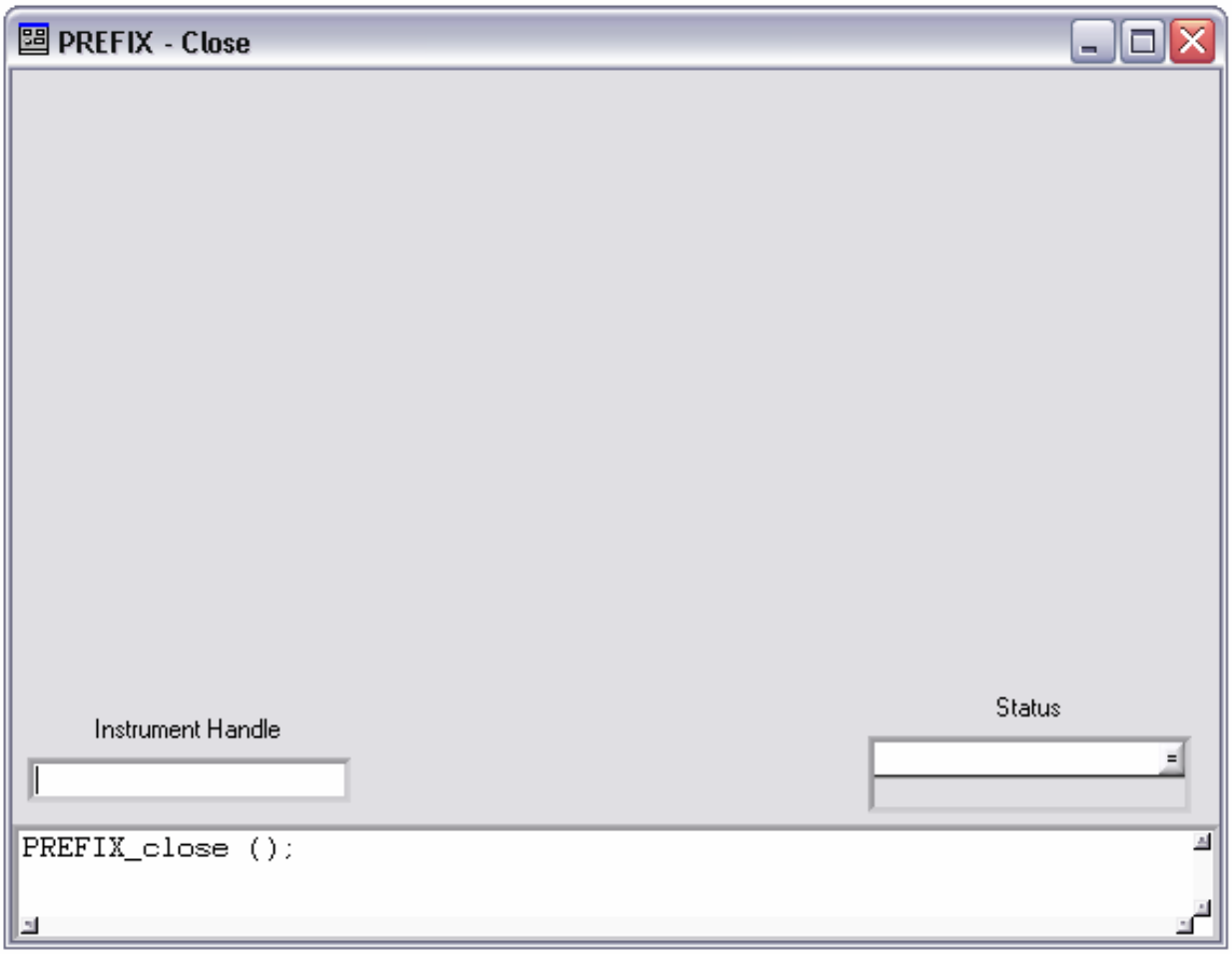

Figure 4-7 PREFIX_close Function Panel Layout

Section 5
Layout of Function Panels for Template Functions for GWINNT Framework

This section shows the layout of each of the template functions required of all VXI*plug&play* drivers. All function panels for these functions must be laid out consistent with the definitions shown here.

5.1 Init Function

RULE 5.1

The location and labeling of controls on the function panel for the PREFIX_init template function **SHALL** appear as follows:

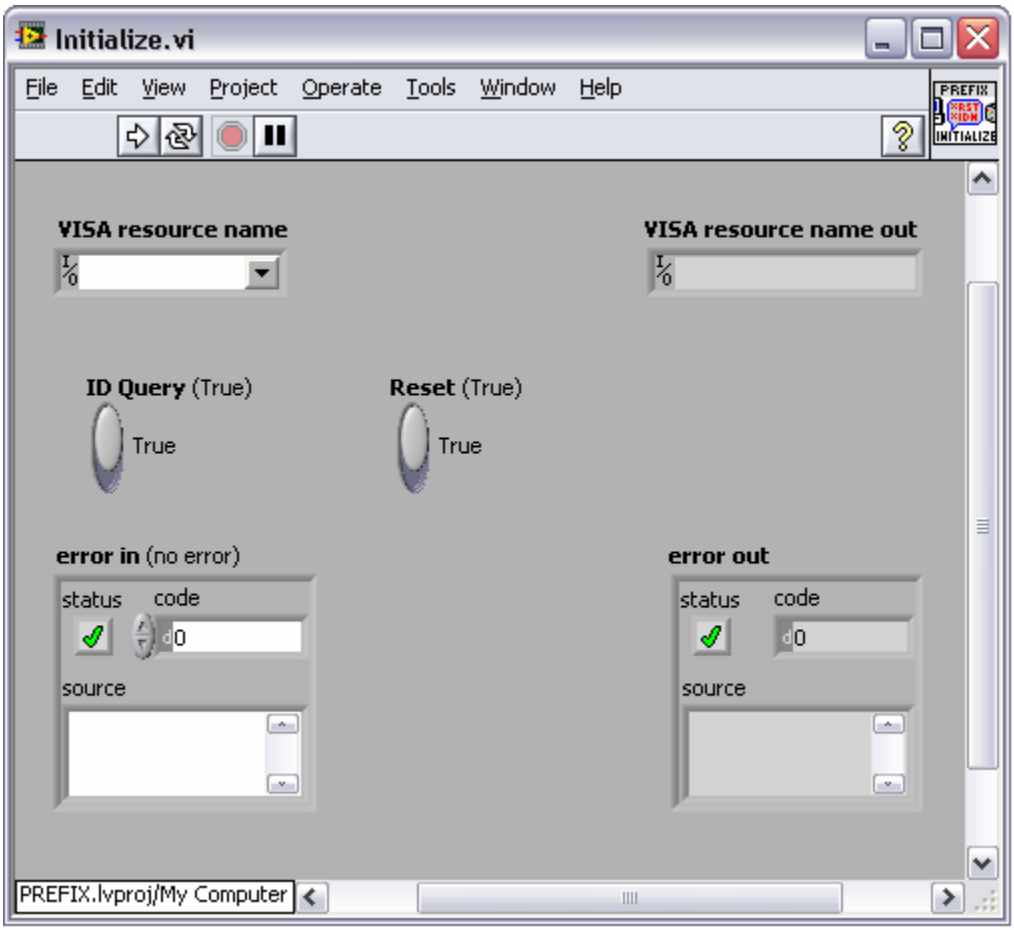

Figure 5-1 PREFIX Initialize.vi Layout

5.2 Reset Function

RULE 5.2

The location and labeling of controls on the function panel for the PREFIX_reset template function **SHALL** appear as follows:

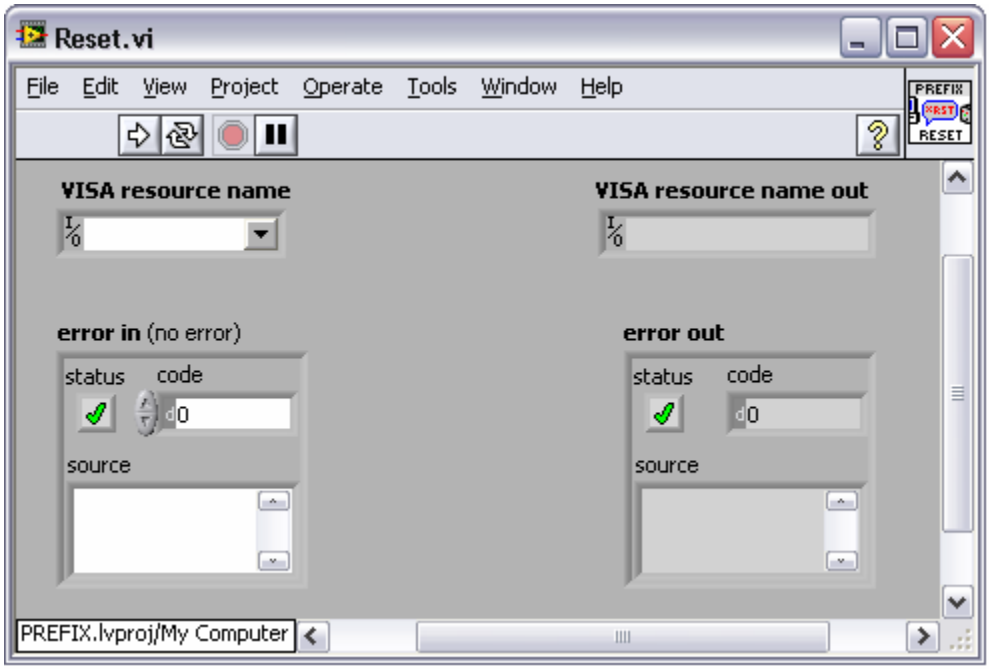

Figure 5-2 PREFIX Reset.vi Layout

5.3 Self Test Function

RULE 5.3

The location and labeling of controls on the function panel for the PREFIX_self_test template function **SHALL** appear as follows:

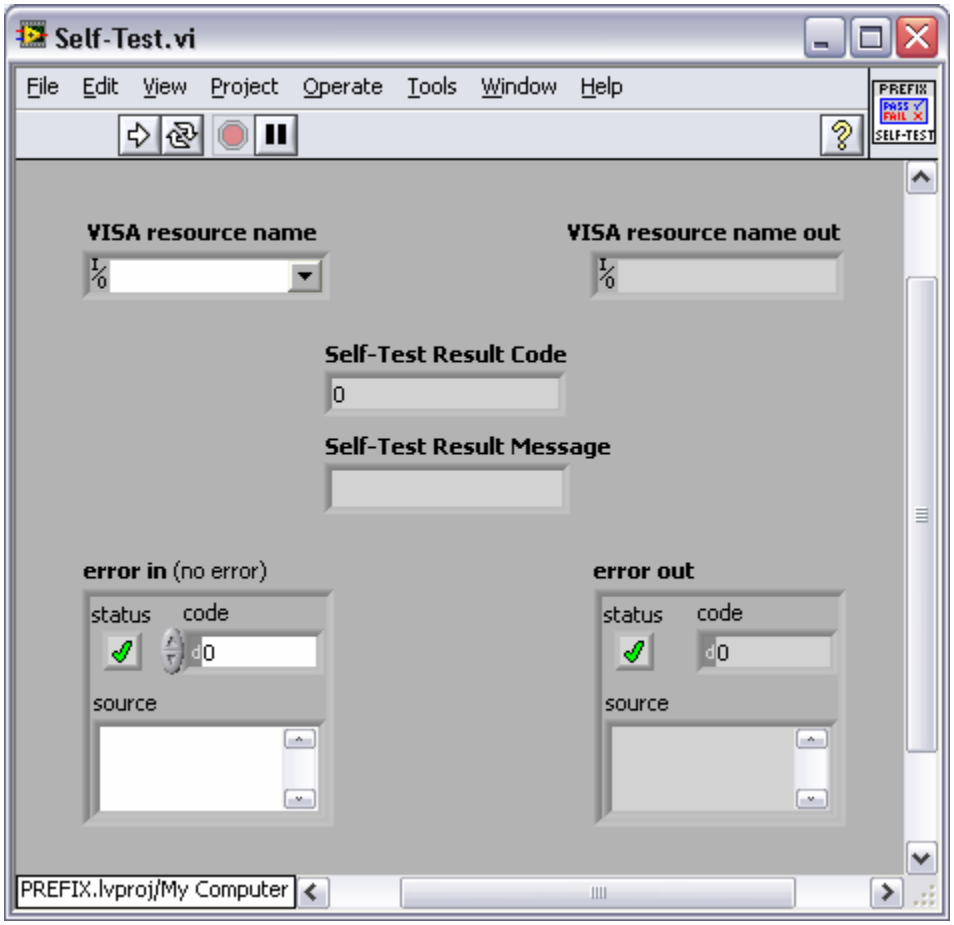

Figure 5-3 PREFIX Self-Test.vi Layout

5.4 Error Query Function

RULE 5.4

The location and labeling of controls on the function panel for the PREFIX_error_query template function **SHALL** appear as follows:

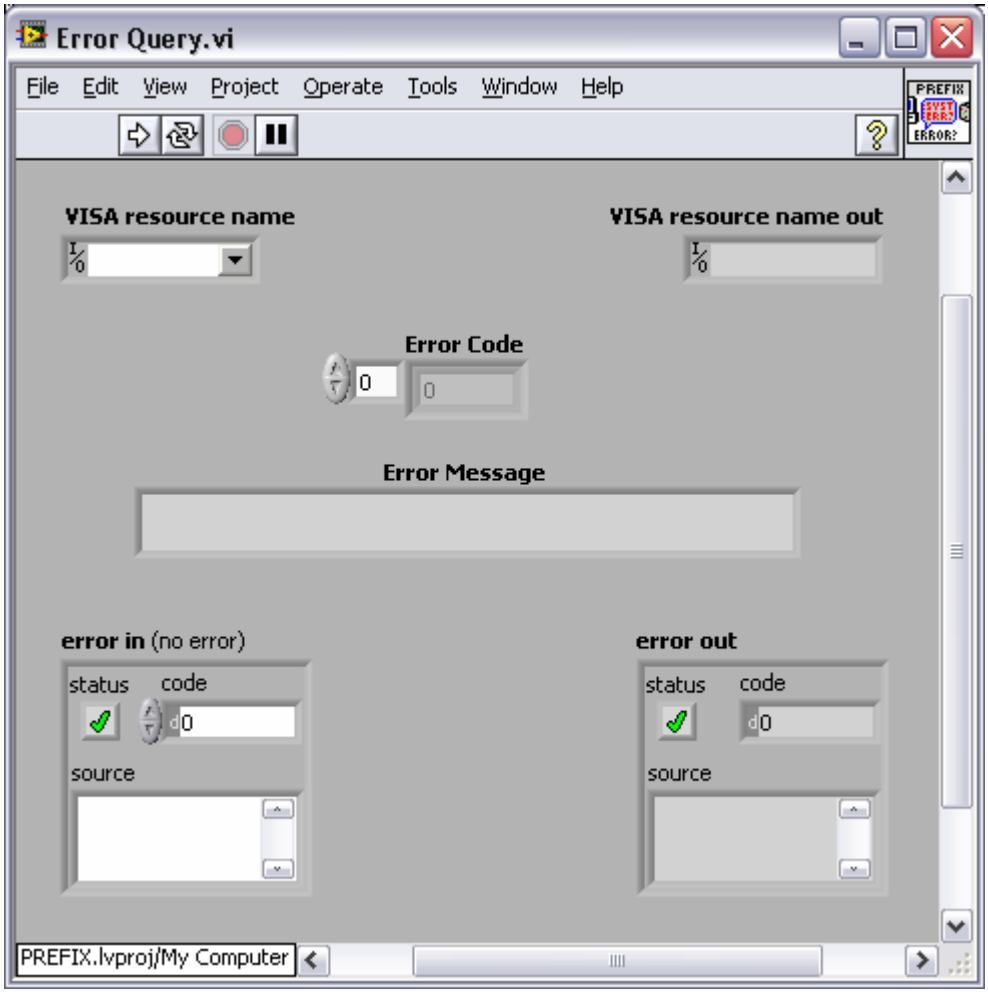

Figure 5-4 PREFIX Error-Query.vi Layout

5.5 Error Message Function

RULE 5.5

The location and labeling of controls on the function panel for the PREFIX_error_message template function **SHALL** appear as follows:

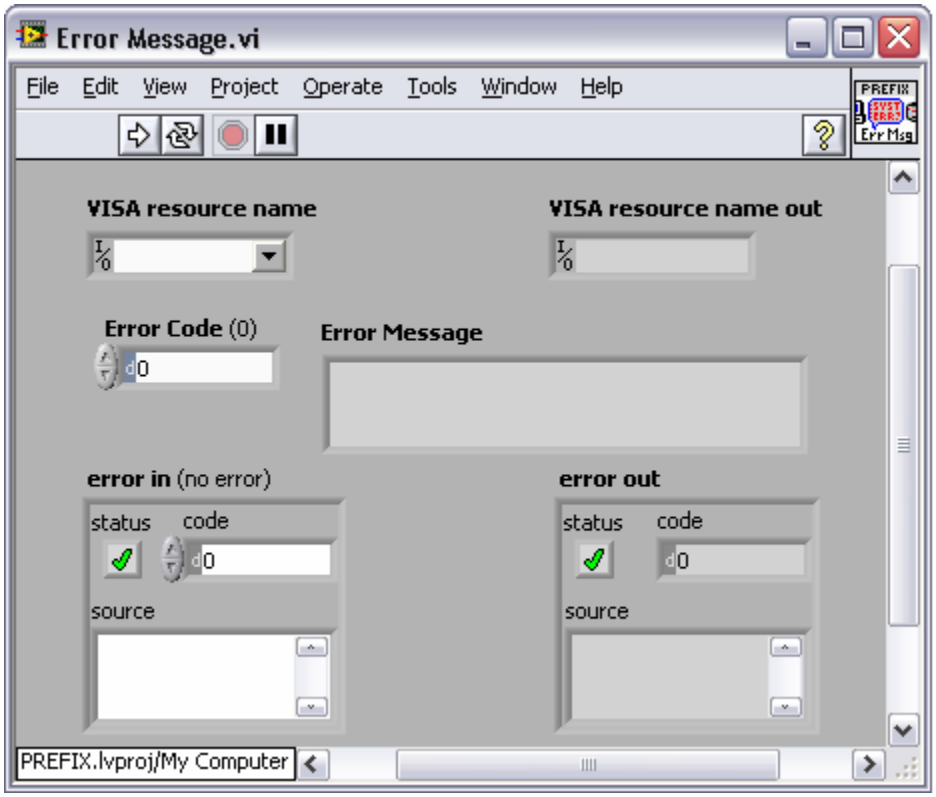

Figure 5-5 PREFIX Error-Message.vi Layout

5.6 Revision Query Function

RULE 5.6

The location and labeling of controls on the function panel for the PREFIX_revision_query template function **SHALL** appear as follows:

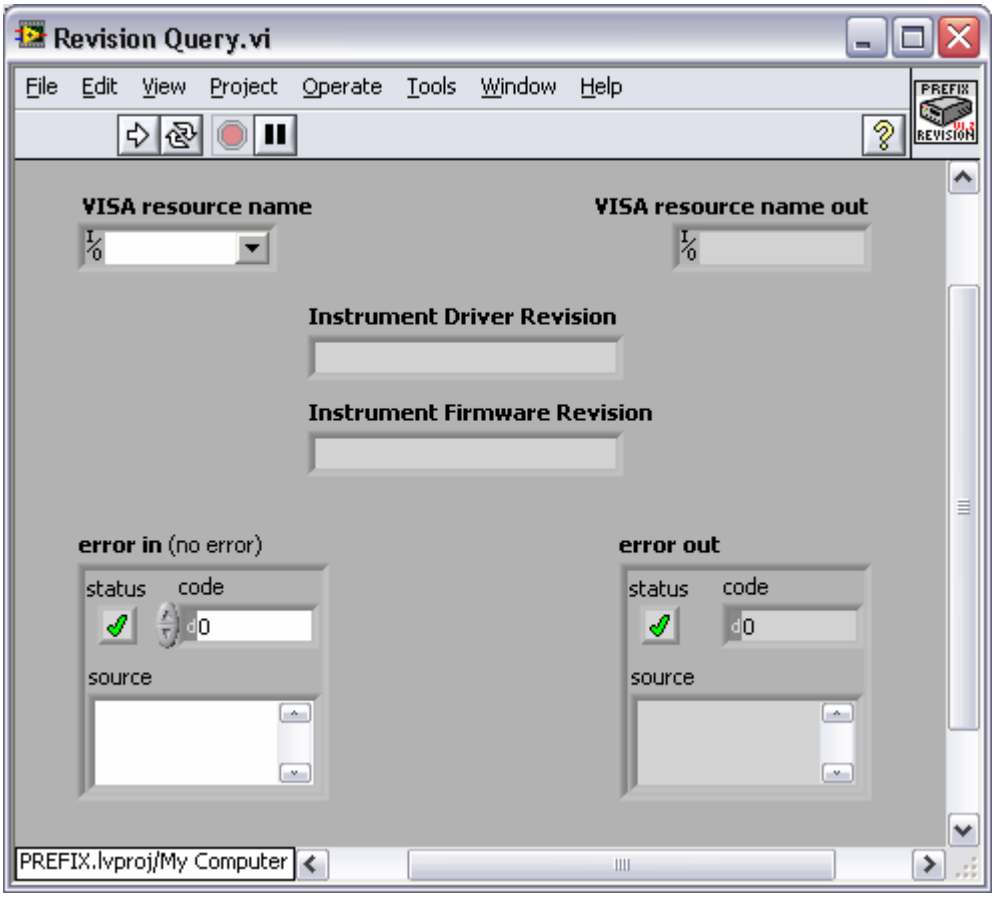

Figure 5-6 PREFIX Revision-Query.vi Layout

5.7 Close Function

RULE 5.7

The location and labeling of controls on the function panel for the PREFIX_close template function **SHALL** appear as follows:

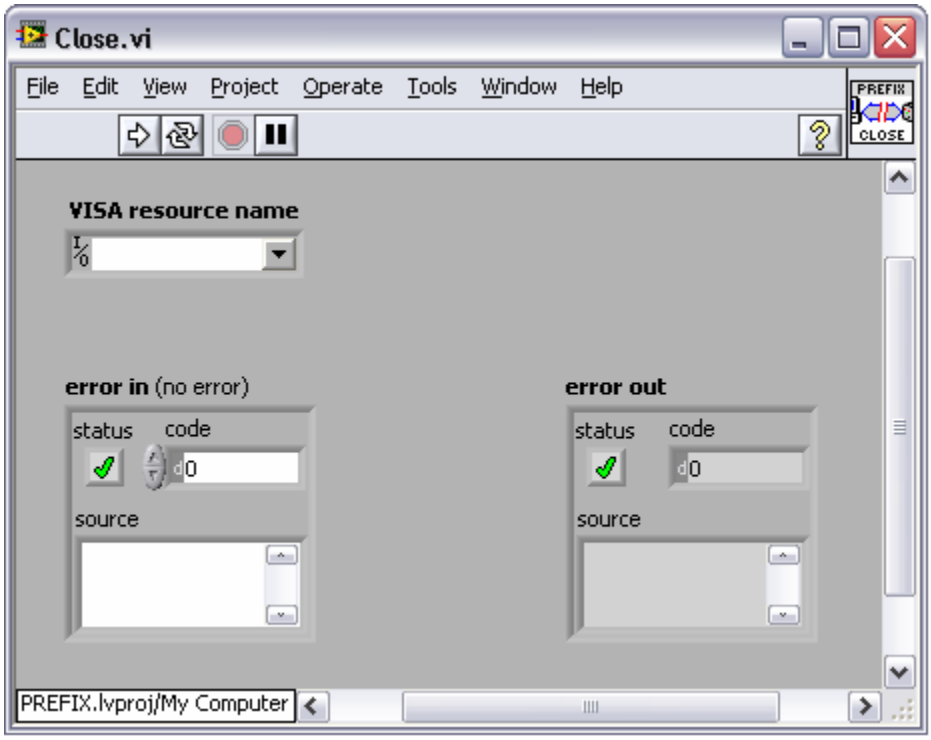

Figure 5-7 PREFIX Close.vi Layout

Section 6
Function Panel File Format

6.1 Overview

This section describes the internal binary format of a function panel file (PREFIX.fp). It is intended to be used by applications that create and/or read function panel files. Designers who are creating instrument drivers should use one of these applications to create their driver file, and are directed to the instrument designer specifications in Sections 4 and 5 of this document. After June 2000 and before June 2007, designers can choose between function panel file format version 4.1 or 5.1. After June 2007, designers can choose between function panel file formats version 4.1, 5.1, or 9.0. Version 5.1 was introduced to allow support for longer prefix and function names. Version 9.0 was introduced to allow support for 64-bit integers. A designer must comply with all changes introduced in each version to be compliant with the corresponding file format.

RULE 6.1

VXIplug&play compliant instrument drivers SHALL use function panel file format version 4.1 until June 2000. After June 2000 and through June 2007 designers must use one of the defined function panel formats, version 4.1 or version 5.1. After June 2007, designers must use one of the defined function panel formats, version 4.1, version 5.1, or version 9.0.

6.2 Data Type Notation

The following data type notation is used:

`ViInt8`	An 8-bit signed integer[1]
`ViUInt8`	An 8-bit unsigned integer[1]
`ViInt16`	A 16-bit signed integer
`ViUInt16`	A 16-bit unsigned integer[1]
`ViInt32`	A 32-bit signed integer
`ViUInt32`	A 32-bit unsigned integer[1]
`ViInt64`	A 64-bit signed integer
`ViUInt64`	A 64-bit unsigned integer[1]
`ViReal64`	A 64-bit IEEE format floating point number

[1] This type should not be used as a type for exportable functions. See VPP-3.4 for details

6.3 Byte Ordering

RULE 6.2

Values of types `ViInt16`, `ViUInt16`, `ViInt32`, `ViUInt32`, `ViInt64`, `ViUInt64`, and `ViReal64` **SHALL** be written to the file in big endian (Motorola) format.

6.4 Reserved Fields

The function panel file format that is documented in this specification was originally developed by National Instruments for the LabWindows/CVI application development environment. As LabWindows/CVI is independent of the VXI*plug&play* Systems Alliance, reserved fields are documented in the function panel format to allow National Instruments to enhance and update the LabWindows/CVI environment while maintaining backward compatibility with this specification.

RULE 6.3

When an application creates a VXI*plug&play* function panel file, it **SHALL** set all reserved fields to zero.

RULE 6.4

When an application reads or parses a VXI*plug&play* function panel file, it **SHALL NOT** require that the reserved fields be zero.

RULE 6.5

Reserved fields **SHALL** be used only as a data field or as an offset to another location in the function panel file.

6.5 Constants

The constants for the two supported function panel file formats are listed below. For constants that are the same for format version 4.1 and 5.1, "same" is written in the 5.1 column. For constants that are different, the 5.1 entry is highlighted with grey. Version 5.1 extends the lengths of prefixes as well as function, class and window name lengths.

RULE 6.6

The following constants **SHALL** be used when creating or parsing a function panel file.

Constant	Ver 4.1 Value	Ver 5.1 Value	Ver 9.0 Value	Description
`CURR_FUNTREE_MAJOR_REV`	4	5	9	Current function panel file major version number. Included in FunctionTreeHdr
`CURR_FUNTREE_MINOR_REV`	1	1	0	Current function panel file minor revision number. Included in FunctionTreeHdr
`FP_FILE_MAGIC_NUM`	same	same	same	Number used to identify a valid function panel file. Included in FunctionTreeHdr

FP_INTRINSIC_TYPE_BIT	same	same	same	For OR'ing with a predefined data type id to force it to a non-zero value. Used primarily with FPUserDataType
MAX_CTRL_LABEL_LEN	same	same	same	Maximum number of bytes in a control label, excluding the NULL byte. Used in FPCtrl.
MAX_FNODE_NAME_LEN	31	79	79	Maximum number of bytes in a class name or window name, excluding the NULL byte. Used in FunctionTreeNode.
MAX_FUNCTION_NAME_LEN	31	79	79	Maximum number of bytes in a function name, excluding the prefix and the NULL byte. Used in FPPanel.
MAX_FUNTREE_LEVEL	8	8	8	Root is level 0. Used in FunctionTreeNode.
MAX_FUNTREE_NAME_LEN	same	same	same	Maximum number of bytes in the instrument name, excluding the NULL byte. Used in FunctionTreeHdr
MAX_INSTR_PREFIX_LEN	8	31	31	Maximum number of bytes in the prefix identifying the instrument, excluding the NULL byte (the trailing underscore is not part of the prefix). Used in FunctionTreeHdr.
MAX_MESSAGE_LEN	same	same	same	Maximum number of bytes in the text in a message control, excluding the NULL byte. Used in FPMessageCtrl.
MAX_NUM_FP_CTRLS	same	same	same	Maximum number of controls per panel
MAX_NUM_FP_PANELS	same	same	same	Maximum number of function panels per window
MAX_NUM_FUNTREE_NODES	same	same	same	Maximum number of function tree nodes per file
MAX_NUM_RING_OPTS	same	same	same	Maximum number of entries in a ring control
MAX_NUM_SLIDE_OPTS	same	same	same	Maximum number of entries in a slide control
MAX_TYPE_STR_LEN	50	80	80	Used to limit entry length for the data type string control; users should use typedefs instead of very long data type strings. Used in FPUserDataType.
MAX_QUALIFIER_LEN		55	55	Used to limit entry length for function calling convention modifiers. Used in Ver 5.1 and 9.0 of FunctionTreeHdr
MAX_DFLT_TEXT_ENTRY_LEN		83	83	Used to limit the entry for default text in input and output controls. Used in FPInputCtrl and FPOutputCtrl.

MAX_GLOBAL_VAR_NAME_LEN		31	31	Used to limit the global variable name. Used in FPOutputCtrl.

6.6 Enumerations

RULE 6.7

The following enumerations **SHALL** be used when creating or parsing a function panel file.

```
Begin Enum_PredefinedDataType
    kfpInteger = 0,
    kfpLong = 1,
    kfpShort = 2,
    kfpChar = 3,
    kfpUnsignedInteger = 4,
    kfpUnsignedLong = 5,
    kfpUnsignedShort = 6,
    kfpUnsignedChar = 7,
    kfpIntegerArray = 8,
    kfpLongArray = 9,
    kfpShortArray = 10,
    kfpCharArray = 11,
    kfpUnsignedIntegerArray = 12,
    kfpUnsignedLongArray = 13,
    kfpUnsignedShortArray = 14,
    kfpUnsignedCharArray = 15,
    kfpDouble = 16,
    kfpFloat = 17,
    kfpDoubleArray = 18,
    kfpFloatArray = 19,
    kfpCharPtr = 20,
    kfpCharPtrArray = 21,
    kfpVoidPtr = 22,          /* does not work for output ctrls;    */
                              /* it would come out as 'void **';    */
                              /* use kfpAnyType instead             */
    kfpNumericArray = 23,
    kfpAnyType = 24,          /* for variably typed arguments       */
                              /* passed by reference                */
    kfpAnyArray = 25,
    kfpVarArgs = 26,          /* for variably typed arguments       */
                              /* passed by value                    */
    kfpLongLong = 27,
    kfpUnsignedLongLong = 28,
    kfpLongLongArray = 29,
    kfpUnsignedLongLongArray = 30,
    kFPUserDataTypeBase = 1000,
    kFPInvalidType = 0xFFFF   /* for termination of data type lists */
End Enum_PredefinedDataType

Begin Enum_HelpStyle
    kFPHelpStyle_New = 0,     /* the default for new function panel */
                              /* files; users define function help  */
                              /* instead of window help; the text is*/
                              /* displayed in EDITOR font instead of*/
```

```
                                /* SYSTEM font; certain information is*/
                                /* displayed automatically and so does*/
                                /* not have to be entered as help text*/

    kFPHelpStyle_Old = 1        /* from conversion of LabWindows DOS  */
                                /* function panels; used SYSTEM font, */
                                /* which is less attractive, but which*/
                                /* supports the extended characters   */
                                /* of the IBM PC character set        */
End Enum_HelpStyle

Begin Enum_NodeTypes
    kFuntreeRoot = 0,
    kFuntreeClass = 1,
    kFunctionPanelWindow = 2,
    kFuntreePlaceholder = 3
End Enum_NodeTypes

Begin Enum_CtrlTypes
    kFPCtrlType_Input = 1,
    kFPCtrlType_Output = 2,
    kFPCtrlType_Ring = 3,       /* used for both ring and numeric     */
                                /* controls                           */
    kFPCtrlType_Binary = 4,
    kFPCtrlType_Slide = 5,
    kFPCtrlType_ReturnValue = 6,
    kFPCtrlType_Global = 7,
    kFPCtrlType_Message = 8
End Enum_CtrlTypes

Begin Enum_NumericDisplayFormat
    kFPNumFmt_Decimal = 0,
    kFPNumFmt_Hex = 1,
    kFPNumFmt_Octal = 2,
    kFPNumFmt_Ascii = 3,
    kFPNumFmt_Scientific = 4,
    kFPNumFmt_Floating = 5
End Enum_NumericDisplayFormat

Begin Enum_RingType
    kFPRingType_Assoc = 1,      /* the true ring control              */
    kFPRingType_IntVal = 2,     /* integer numeric control            */
    kFPRingType_RealVal = 3     /* floating point numeric control     */
End Enum_RingType
```

RULE 6.8

The addition of new enumerations **SHALL NOT** change the value of existing enumerations.

RULE 6.9

User data types **SHALL** have values starting at `kFPUserDataTypeBase` and **SHALL** increment positively by 1.

6.7 Controls and Associated Data Types

In general, the data type of a control may be either an `Enum_PredefinedDataType` value, or a value between `kFPUserDataTypeBase` and 65534; if the latter, it must match an 'id' in one of the `FPUserDataType` records.

RULE 6.10

The data type associated with numeric controls **SHALL** only be either `kfpInteger`, `kfpShort`, `kfpLongLong`, `kfpDouble`, `kfpFloat`, or an id of an `FPUserDataType` with a valid non-zero intrinsic type.

RULE 6.11

The data type associated with return value controls **SHALL NOT** be an array type, `kfpAnyType`, or `kfpVarArgs`.

OBSERVATION 6.1

The data type for return value controls must be `ViStatus` for VXI*plug&play* drivers (see VPP-3.4)

6.8 File Record Formats

OBSERVATION 6.2

The following table summarizes the changes in the record definitions between function panel file format version 4.1 and version 5.1.

Record Name	Record definintion changes	Maximum length changes	Description
FunctionTreeHdr	Yes	Yes	• Added qualifier field. • The instrPrefix field uses MAX_INSTR_PREFIX_LEN which was changed from 8 to 31. • Removed 3 byte reserved4 field. • The funtreeVersion value is changed to reflect new format.
FPPanel	Yes	Yes	• Added qualifier field. • The functionName field uses MAX_FUNCTION_NAME_LEN which was changed from 31 to 79. • Added new reserved fields for Int32 and Int16.
FPOutputCtrl		Yes	• MAX_GLOBAL_VAR_NAME_LEN was defined to be max length for globVarName field. • Two optional fields were defined to differentiate between output control types – return, output and global.
FPRealValSet	Yes		Converted one of the reserved fields to be a precision field.

FPUserDataType		Yes	The dataTypeStr field uses MAX_TYPE_STR_LEN which was changed from 50 to 80
FunctionTreeNode		Yes	The name field uses MAX_FNODE_NAME_LEN which was changed from 31 to 79
FPWin			The fpPanel field is based on version 4.1 of record definition or version 5.1 record definition.
FPIntputCtrl		Yes	The dfltTextEntry field max length defined to be `MAX_DFLT_TEXT_ENTRY_LEN`
FPHelpText			No Change between Ver 4.1 and Ver 5.1
FPCtrl			No Change between Ver 4.1 and Ver 5.1
FPMessageCtrl			No Change between Ver 4.1 and Ver 5.1
FPBinaryCtrl			No Change between Ver 4.1 and Ver 5.1
FPAssocSet			No Change between Ver 4.1 and Ver 5.1
FPIntValSet			No Change between Ver 4.1 and Ver 5.1
FPAutoLoadList			No Change between Ver 4.1 and Ver 5.1
AutoLoadName			No Change between Ver 4.1 and Ver 5.1

OBSERVATION 6.3

The following summarizes the changes in the record definitions between function panel file format version 5.1 and version 9.0:

FPIntValSet is used with numeric controls for which the data type is `kfpInteger`, or `kfpShort`.
FPInt64ValSet is used with numeric controls for which the data type is `kfpLongLong`.

RULE 6.12

The following records are the only records that **SHALL** appear in a function panel file.

RULE 6.13

All records that appear in a function panel file **SHALL** be implemented as described in this section.

```
/* Version 4.1 FunctionTreeHdr record:                                  */
/* Only one FunctionTreeHdr record should be implemented.  Use          */
/* this record if complying with Version 4.1 of the file format.        */

Begin FunctionTreeHdr
    ViUInt32  magicNumber   /* FP_FILE_MAGIC_NUM                        */
    ViInt32   funtreeVersion/* CURR_FUNTREE_VERSION                     */
    ViInt32   reserved1
    ViInt32   treeSaveOffset/* file offset of FuntreeNodelist           */
    ViInt32   numNodes      /* number of nodes actually used            */
    ViInt32   winInfoFirstSaveOffset
                            /* file offset of first FPWin               */
    ViInt32   winInfoTotNumBytes
                            /* total number of bytes in all FPWins      */
    ViInt32   userDataTypeSaveOffset
                            /* file offset of user data type info       */
    ViInt32   numUserDataTypes
                            /* number of user data type structures      */
    ViInt32   autoLoadListOffset
                            /* offset. 0 or -1 mean no list             */
    ViInt32   reserved2[6]
    ViUInt16  minorRevisionNumber
                            /*                                          */
    ViUInt16  reserved2a    /*                                          */
    ViUInt8   helpStyle     /* Enum_HelpStyle                           */
    ViUInt8   reserved3[3]
    ViUInt8   instrPrefix[MAX_INSTR_PREFIX_LEN+1]
                            /* instrument prefix as                     */
                            /* NULL-terminated string                   */
    ViUInt8   reserved4[3]
    ViUInt8   name[MAX_FUNTREE_NAME_LEN+1]
                            /* instrument name as                       */
                            /* NULL-terminated string                   */
    ViUInt8   reserved5[3]
End FunctionTreeHdr


/* Version 5.1 FunctionTreeHdr record:                                  */
/* Only one FunctionTreeHdr record should be implemented.  Use          */
/* this record if complying with Version 5.1 of the file format.        */

Begin FunctionTreeHdr
    ViUInt32  magicNumber   /* FP_FILE_MAGIC_NUM                        */
    ViInt32   funtreeVersion/* CURR_FUNTREE_VERSION                     */
    ViInt32   reserved1
    ViInt32   treeSaveOffset/* file offset of FuntreeNodelist           */
    ViInt32   numNodes      /* number of nodes actually used            */
    ViInt32   winInfoFirstSaveOffset
```

```
                        /* file offset of first FPWin            */
    ViInt32   winInfoTotNumBytes
                        /* total number of bytes in all FPWins   */
    ViInt32   userDataTypeSaveOffset
                        /* file offset of user data type info    */
    ViInt32   numUserDataTypes
                        /* number of user data type structures   */
    ViInt32   autoLoadListOffset
                        /* offset. 0 or -1 mean no list          */
    ViInt32   reserved2[6]
    ViUInt16  minorRevisionNumber
                        /*                                       */
    ViUInt16  reserved2a    /*                                   */
    ViUInt8   helpStyle     /* Enum_HelpStyle                    */
    ViUInt8   reserved3[3]
    ViUInt8   instrPrefix[MAX_INSTR_PREFIX_LEN+1]
                        /* instrument prefix as                  */
                        /* NULL-terminated string                */
    ViUInt8   name[MAX_FUNTREE_NAME_LEN+1]
                        /* instrument name as                    */
                        /* NULL-terminated string                */
    ViUInt8   reserved5[3]
    ViUInt8   qualifier[MAX_QUALIFIER_LEN+1]
                  /* function qualifier for calling convention.  */
                  /* NULL-terminated string.                     */
                  /* If qualifier in FPPanel is non-empty, then  */
                  /* FPPanel qualifier takes precedence over     */
                  /* FunctionTreeHdr qualifier.                  */
End FunctionTreeHdr
```

```
Begin FPUserDataType        /* variable-length record                */
    ViUInt16  reserved
    ViUInt16  intrinsicType /* only for supporting numeric controls. */
                            /* must be zero, or must be one of the   */
                            /* following OR'ed with                  */
                            /* FP_INTRINSIC_TYPE_BIT: kfpInteger,    */
                            /* kfpShort, kfpLongLong, kfpDouble,     */
                            /* kfpFloat                              */
    ViInt16   typeStrLen    /* size of the data type string,         */
                            /* excluding the NULL byte               */
    ViUInt16  id            /* a unique value between                */
                            /* kFPUserDataTypeBase and 65534         */
    ViInt16   varNamePos    /* position at which to insert variable  */
                            /* name into the data type string        */
    ViInt16   dimLenPos     /* position at which to insert array     */
                            /* length in data type string; -1 if     */
                            /* not an array                          */
    ViUInt8   dataTypeStr[] /* the data type string, such as 'int'   */
                            /* or 'void (*)(int)', in variable-      */
                            /* length form; must be between 1 and    */
                            /* MAX_TYPE_STR_LEN bytes; the NULL      */
                            /* byte is not included in the file.     */
                            /* Legal values for the data type string */
                            /* are any of the data types listed in   */
                            /* VPP-3.4 Table 3-1, Compatible Data    */
                            /* Types plus ViChar and ViAddr.         */
End FPUserDataType

Begin FPHelpText            /* variable-length record                */
    ViInt32   numBytes      /* the number of bytes in (the string)   */
                            /* helpText, including the two trailing  */
                            /* NULL bytes                            */
    ViInt32   reserved
    ViUInt8   helpText[]    /* the help text; new lines are          */
                            /* represented by ASCII LF (decimal 10)  */
                            /* characters;                           */
                            /* two NULL bytes at the end of the text */
                            /* are included in the file              */

End FPHelpText
```

```
Begin FunctionTreeNode
    ViUInt8   type          /* see Enum_FuntreeNodeType               */
    ViUInt8   level         /* root is 0; deepest level is            */
                            /* MAX_FUNTREE_LEVEL                      */
    ViUInt8   reserved1
    ViUInt8   reserved2
    either:
    ViInt32   helpOffset
                            /* for a root or class node, the file     */
                            /* offset of the help text;               */
                            /* -1L if no help text                    */
    or
    ViInt32   numBytes
                            /* for a function panel window node,      */
                            /* the number of bytes in the FPWin       */
                            /* record                                 */
    ViUInt8   name[MAX_FNODE_NAME_LEN+1]
                            /* node name as NULL-terminated string    */
                            /* displayed in the function panel        */
                            /* selection dialog and on function panel */
                            /* not used for root node--too small      */
End FunctionTreeNode

Begin FPWin   /* variable-length record                                */
    ViInt32   helpOffset    /* file offset of window help text;       */
                            /* applies only to "old style" help;      */
                            /* -1L if no help text                    */
    ViInt16   reserved1
    ViInt16   reserved2
    ViInt16   numPanels     /* number of function panels in window    */
    ViInt16   reserved3
    FPPanel   fpPanel[]     /* one or more function panels.  The      */
                            /* fpPanel structure is different for     */
                            /* version 4.1 and version 5.1.           */
End FPWin
```

```
/* Version 4.1 FPPanel record:                                          */
/* Only one FPPanel record should be implemented.  Use this             */
/* record if complying with Version 4.1 of the file format.             */

Begin FPPanel
    ViInt32   helpOffset    /* file offset of function help text;       */
                            /* applies only to "new style" help;        */
                            /* -1L if no help text                      */
    ViInt32   ctrlListOffset
                            /* file offset of the first FPCtrl          */
                            /* record for the panel;                    */
                            /* -1L if no controls                       */
    ViInt16   numCtrls      /* number of controls in panel              */
    ViInt16   fnPos         /* if only one panel in window, then        */
                            /* set to 0; if multiple panels, this       */
                            /* is the zero-based position of the        */
                            /* function with respect to the other       */
                            /* functions in the generated code;         */
                            /* for common control panel, set to -1      */
    ViInt16   y             /* vertical position in pixels - see        */
                            /* note below                               */
    ViInt16   x             /* horizontal position in pixels - see      */
                            /* note below                               */
    ViInt16   height        /* client area height - see note below      */
    ViInt16   width         /* client area width - see note below       */
    ViInt8    disabledDefault
                            /* non-zero if the panel should be          */
                            /* disabled upon being loaded;              */
                            /* normally is zero                         */
    ViInt8    scrollBars    /* non-zero if the panel has scroll         */
                            /* bars; normally is zero                   */
    ViInt8    reserved1
    ViInt8    reserved2
    ViUInt8   functionName[MAX_FUNCTION_NAME_LEN+1]
                            /* the NULL-terminated name of the          */
                            /* function excluding the prefix            */
End FPPanel
```

```
/* Version 5.1 FPPanel record:                                          */
/* Only one FPPanel record should be implemented.  Use this            */
/* record if complying with Version 5.1 or 9.0 of the file format.     */

Begin FPPanel
    ViInt32   helpOffset    /* file offset of function help text;      */
                            /* applies only to "new style" help;       */
                            /* -1L if no help text                     */
    ViInt32   ctrlListOffset
                            /* file offset of the first FPCtrl         */
                            /* record for the panel;                   */
                            /* -1L if no controls                      */
    ViInt32   reserved1
    ViInt32   reserved2
    ViInt16   numCtrls      /* number of controls in panel             */
    ViInt16   fnPos         /* if only one panel in window, then       */
                            /* set to 0; if multiple panels, this      */
                            /* is the zero-based position of the       */
                            /* function with respect to the other      */
                            /* functions in the generated code;        */
                            /* for common control panel, set to -1     */
    ViInt16   y             /* vertical position in pixels - see       */
                            /* note below                              */
    ViInt16   x             /* horizontal position in pixels - see     */
                            /* note below                              */
    ViInt16   height        /* client area height - see note below     */
    ViInt16   width         /* client area width - see note below      */
    ViInt16   reserved3
    ViInt16   reserved4
    ViInt8    disabledDefault
                            /* non-zero if the panel should be         */
                            /* disabled upon being loaded;             */
                            /* normally is zero                        */
    ViInt8    scrollBars    /* non-zero if the panel has scroll        */
                            /* bars; normally is zero                  */
    ViInt8    reserved5
    ViInt8    reserved6
    ViUInt8   functionName[MAX_FUNCTION_NAME_LEN+1]
                            /* the NULL-terminated name of the         */
                            /* function excluding the prefix           */
    ViUInt8   qualifier[MAX_QUALIFIER_LEN+1]
                            /* the NULL-terminated name of the         */
                            /* calling convension.                     */
      /* If this qualifier is non-empty, then it    */
      /* takea precedence over the  FunctionTreeHdr */
               /* qualifier.              */
End FPPanel
```

```
/************************************************************/
/*       Note on FPPanel 'y', 'x', 'height' and 'width':    */
/*                                                          */
/*       This discussion is to give insight as to how function */
/*       panels are parsed and displayed in CVI. Other      */
/*       applications are not required to display function  */
/*       panels in the same manner.                         */
/*                                                          */
/*       The area within a function panel window that contains */
/*       the function panel is called the "container area." */
/*       The default size of the container area is defined by */
/*       the following constants:                           */
/*                                                          */
/*       DFLT_FPCONTAINER_HEIGHT = 349                      */
/*       DFLT_FPCONTAINER_WIDTH  = 560                      */
/*                                                          */
/*       Each function panel has a title bar and frame, the */
/*       sizes of which are defined by the following constants: */
/*                                                          */
/*       FPPANEL_FRAME_THICKNESS    =  1                    */
/*       FPPANEL_TITLEBAR_THICKNESS = 16                    */
/*                                                          */
/*       The 'height' and 'width' members of FPPanel define the */
/*       size of the client area (which excludes the frame and */
/*       titlebar) of the function panel. Thus, a function  */
/*       panel that exactly fills up the entire container area */
/*       would have:                                        */
/*                                                          */
/*       height        = DFLT_FPCONTAINER_HEIGHT            */
/*       - (2 * FPPANEL_FRAME_THICKNESS)                    */
/*       - FPPANEL_TITLEBAR_THICKNESS                       */
/*                     = 331                                */
/*                                                          */
/*       width         = DFLT_FPCONTAINER_WIDTH             */
/*       - (2 * FPPANEL_FRAME_THICKNESS)                    */
/*                     = 558                                */
/*                                                          */
/*       The 'y' and 'x' members of the FPPanel define the  */
/*       location of the top left corner of the function panel's */
/*       client area with respect to the top left corner of the */
/*       container area. Thus, a function panel that exactly */
/*       fills up the entire container area would have:     */
/*                                                          */
/*       y             = FPPANEL_FRAME_THICKNESS            */
/*       + FPPANEL_TITLEBAR_THICKNESS                       */
/*                     = 17                                 */
/*                                                          */
/*       x             = FPPANEL_FRAME_THICKNESS            */
/*                     = 1                                  */
/************************************************************/
```

```
Begin FPCtrl
    ViInt32   helpOffset    /* file offset of control help text;      */
                            /* -1L if no help text                    */
    ViInt16   y             /* vertical position in pixels - see      */
                            /* note below                             */
    ViInt16   x             /* horizontal position in pixels - see    */
                            /* note below                             */
    ViInt16   parmPos       /* zero-based parameter position of the   */
/* control within the function call                                   */
/* after any common controls; for global                              */
/* variable controls and return value                                 */
/* controls, set to -1                                                */
    ViUInt16  dataTypeId    /* either an Enum_PredefinedDataType      */
                            /* value, or a value between              */
                            /* kFPUserDataTypeBase and 65534;         */
                            /* if the latter, it must match an id     */
                            /* in one of the FPUserDataType records.  */
                            /* For numeric controls, must be either   */
                            /* kfpInteger, kfpShort, kfpLongLong,     */
                            /* kfpDouble, kfpFloat, or an id of an    */
                            /* FPUserDataType with a valid non-zero   */
                            /* intrinsic type. For return value       */
                            /* controls, must not be an array type,   */
                            /* kfpAnyType, or kfpVarArgs.             */
    ViUInt8   ctrlType      /* see Enum_CtrlType                      */
    ViUInt8   ringType      /* see Enum_RingType                      */
    ViUInt16  ctrlWidth     /* Only applies to input, output, return  */
                            /* values and global controls. Does not   */
                            /* apply to rings, slides, messages, and  */
                            /* binaries. max=2048, min = 24,          */
                            /* default = 96, 0 means use default      */
    ViUInt8   reserved1
    ViUInt8   reserved2
    ViUInt8   reserved3
    ViUInt8   reserved4
    ViUInt8   label[MAX_CTRL_LABEL_LEN+1]
                            /* control label as NULL-terminated       */
                            /* string                                 */
End FPCtrl
```

```
    /************************************************************/
    /* Note on FPCtrl 'y', 'x':                                 */
    /*                                                          */
    /*       These represent the pixel coordinates of the top left  */
    /*       corner of the control with respect to the top left     */
    /*       corner of the client area of the function panel.       */
    /*                                                          */
    /*       The 'top left corner' of the control does NOT include  */
    /*       the control label. To get a good idea of where the top */
    /*       left corner of a control is, create controls in the    */
    /*       LabWindows/CVI User Interface Editor Window, move them */
    /*       around, and use the Control Coordinates command to look*/
    /*       at their coordinates.                              */
    /*                                                          */
    /************************************************************/

Begin FPMessageCtrl         /* variable-length record                */
    ViInt32   numBytes      /* number of bytes in msgTxt, including  */
                            /* the NULL byte                         */
    ViUInt8   msgText[]     /* the message text as a NULL-terminated */
                            /* string; size (including NULL byte)    */
                            /* must not exceed MAX_MESSAGE_LEN+1     */
End FPMessageCtrl

Begin FPInputCtrl           /* variable-length record                */
    ViInt32   numBytes      /* number of bytes in dfltTextEntry,     */
                            /* including the NULL byte               */
    ViUInt8   dfltTextEntry[]
                            /* the default entry in the control,     */
                            /* as a NULL-terminated string; size     */
                            /* (including NULL byte) must not        */
                            /* exceed MAX_DFLT_TEXT_ENTRY_LEN+1      */
End FPInputCtrl
```

```
Begin FPOutputCtrl          /* variable-length record; used for       */
                            /* output, return, & global controls      */
    ViInt32   numBytes      /* 8 + number of bytes in globVarName,    */
                            /* or dfltTextEntry, including the        */
                            /* NULL byte*/
    ViInt32   reserved1
    ViInt8    dfltFormat    /* default numeric display format;        */
                            /* see Enum_NumericDisplayFormat          */
    ViInt8    reserved2[3]
Either:
    ViUInt8   globVarName[] /* the name of the global variable        */
                            /* (NULL-terminated) size                 */
                            /* (including NULL byte) must not         */
                            /*  exceed MAX_GLOBAL_VAR_NAME_LEN+1      */
or
    ViUInt8   dfltTextEntry[]
                          /* the default entry in the output control, */
                            /* as a NULL-terminated string; size      */
                            /* (including NULL byte) must not         */
                            /* exceed MAX_DFLT_TEXT_ENTRY_LEN+1       */
or
    ViUInt8   EmptyString[1]  /* Set to NULL if used for a return     */
                            /* control. No default entries.           */
End FPOutputCtrl

Begin FPBinaryCtrl          /* variable-length record                 */
    ViInt16   numBytes      /* number of bytes in labelValPairs       */
                            /* including NULL bytes                   */
    ViInt8    reserved
    ViInt8    dfltVal       /* 1 for 'on', 0 for 'off' */
    ViUInt8   labelValPairs[]
                            /* variable-length flattened list of      */
                            /* NULL-terminated label and value        */
                            /* strings; the order is:                 */
                            /*           'on' label, 'on' value,      */
                            /*           'off' label, 'off' value     */
End FPBinaryCtrl

Begin FPAssocSet            /* variable-length record                 */
                            /* slide and ring controls                */
    ViInt32   reserved
    ViInt32   dfltIndex     /* zero-based index of the label          */
                            /* value pair that contains the           */
                            /* default value                          */
    ViInt32   numPairs      /* number of label-value pairs            */
    ViInt32   numBytes      /* number of bytes in labelValPairs       */
                            /* including NULL bytes                   */
    ViUInt8   labelValPairs[]
                            /* variable-length flattened list of      */
                            /* NULL-terminated label and value        */
                            /* strings; the order is:                 */
                            /*            label1, value1,             */
                            /*            label2, value2, and so on.  */
End FPAssocSet
```

```
Begin FPIntValSet           /* integer numeric control (when data     */
                            /* type is kfpInteger or kfpShort)        */
    ViInt32   incr          /* increment from one value to next       */
    ViInt32   maxVal        /* largest value in set                   */
    ViInt32   minVal        /* smallest value in set                  */
    ViInt32   dfltVal       /* must be in the set                     */
    ViInt32   reserved1
    ViInt8    displayFormat /* numeric format in which value is       */
                            /* displayed; see                         */
                            /* Enum_NumericDisplayFormat              */
    ViInt8    reserved2[3]
End FPIntValSet


Begin FPInt64ValSet         /* integer numeric control (when data     */
                            /* type is kfpLongLong)                   */
    ViInt64   incr          /* increment from one value to next       */
    ViInt64   maxVal        /* largest value in set                   */
    ViInt64   minVal        /* smallest value in set                  */
    ViInt64   dfltVal       /* must be in the set                     */
    ViInt64   reserved1
    ViInt8    displayFormat /* numeric format in which value is       */
                            /* displayed; see                         */
                            /* Enum_NumericDisplayFormat              */
    ViInt8    reserved2[3]
End FPIntVal64Set


Begin FPRealValSet          /* floating point numeric control         */
    ViReal64  incr          /* amount to increment from one           */
                            /* value to the next                      */
    ViReal64  maxVal        /* largest value in set                   */
    ViReal64  minVal        /* smallest value in set                  */
    ViReal64  dfltVal       /* must be in the set                     */
    ViReal64  reserved1
    ViInt8    displayFormat
                            /* numeric format in which value is       */
                            /* displayed                              */
    ViInt8     precision    /* decimal digits, min 1, max 15          */
    ViInt8    reserved2[2]
End FPRealValSet

Begin FPAutoLoadList        /* List of other function panels which    */
    ViInt32   numNames      /* are automatically loaded when this     */
    AutoLoadName autoLoadName[]
                            /* this panel is loaded.                  */
End FPAutoLoadList

Begin AutoLoadName          /* Variable length record including       */
                            /* null-terminator                        */
    ViInt32   size          /* size of name, including null terminator*/
    ViInt8    name[]        /* null-terminated name of *.fp file. No  */
                            /* directory information. Example         */
                            /* "scope.fp"                             */
End AutoLoadName
```

OBSERVATION 6.4

The sizes of the records above are exactly as shown. If you declare these records as `'structs'` in C, the `sizeof()` operator may evaluate the `'struct'` sizes differently depending on the platform. This is particularly likely in the case of `FPRealValSet`.

6.9 Order of Records in the File

RULE 6.14

The records in a function panel file **SHALL** be laid out in the order shown in this section.

```
One FunctionTreeHdr record.

Zero or more FPUserDataType records.

For each root and class node in the function tree,
    If it has non-empty help text,
        An FPHelpText record.

For each function panel window node in the function tree,
    For each function panel,
        If the function help is non-empty,
            An FPHelpText record.
        For each control,
            If the control help is non-empty,
                An FPHelpText record.
        For each control,
            An FPCtrl record.
        For each control,
            One of the following records:
                FPMessageCtrl
                FPInputCtrl
                FPOutputCtrl
                FPBinaryCtrl
                FPAssocSet    /* slide and ring controls        */
                FPRealValSet  /* floating point numeric control */
                FPIntValSet   /* integer numeric control        */
                FPInt64ValSet /* integer numeric control        */

For each function panel window node in the function tree,
    An FPWin record.

For each node of any type in the function tree
    A FunctionTreeNode record.

Zero or one FPAutoLoadList records /* List of function panel file*/
                                   /* names to be auto loaded   */
```

OBSERVATION 6.5

There is no padding between the elements in records.

OBSERVATION 6.6

An implementation that reads a function panel file should not assume that there is nothing beyond the end of the last record.

6.10 Implementation Details for Reading Function Panel Files

This section is designed to provide guidance to those Application Development Environments that parse function panel files.

An instrument designer may choose to include functions that are language dependent as defined in VPP-3.4 as part of the function panel definition. In addition, some ADEs may choose to track the instrument handle automatically, providing the init and close functions in the background.

PERMISSION 6.1

An application that reads a function panel file may choose to exclude certain panels from being presented to the user.

RULE 6.15

An application that reads a function panel file **SHALL NOT** generate an error unless that file does not conform to the format presented in this document. In particular, an implementation **SHALL NOT** generate an error if it encounters a function it chooses to exclude.

Often there is not sufficient information in the function panel file itself to make the determination whether to exclude a file or not. In these cases, the implementation may need to read the include file (prefix.h) in order to make a decision. Specifically, the include file contains information on language-specific functions.

The following rules, recommendations, and permissions are designed to provide guidance for both Function Panel designers who choose to use the FPAutoLoadList feature, and the Application Development Environments and users who use these function panel files. It is assumed that a "parent.fp" is auto-loading a "child.fp".

An auto-loaded child instrument driver is one that appears in the FPAutoLoadList of another function panel.

RULE 6.16

An application that reads a function panel file and automatically loads files pertaining to an FPAutoLoadList specified in the function panel file **SHALL** search according to the following rules (exact filenames and directories are specified by the framework):

- **IF** the application automatically searches for the child DLL or shared library **THEN** it **SHALL** first search in < VXIPNPPATH > \ < FRAMEWORK > \ BIN \ child.dll, after which it **SHALL** optionally search in other ways.
- **IF** the application automatically searches for the child include file **THEN** it **SHALL** first search in < VXIPNPPATH > \ < FRAMEWORK > \ INCLUDE \ child.h, after which it **SHALL** optionally search in other ways.
- **IF** the application automatically searches for the child function panel file **THEN** it **SHALL** search in the following locations and order:
  1. Optionally search for child.fp file in a list of files that it already knows about (such as a project list)
  2. Look for < VXIPNPPATH > \ < FRAMEWORK > \ SUPPORT \ child \ child.fp
  3. Look for < VXIPNPPATH > \ < FRAMEWORK > \ child \ child.fp
  4. Optionally search in other ways

RULE 6.17

IF the application automatically searches for other child files (knowledge base, etc) **THEN** it **SHALL** first search for them in the location specified by installation VPP-6 after which it **SHALL** optionally search in other ways.

RULE 6.18

An application that reads a function panel file and automatically loads files pertaining to an FPAutoLoadList specified in the function panel file **SHALL NOT** terminate when not finding a file it attempts to automatically load and **SHALL** continue processing the remaining entries in the FPAutoLoadList.

RULE 6.19

All functions in the auto-loaded child instrument driver **SHALL** conform to the Instrument Driver Naming Conventions in VPP-3.2 and all other VXI*plug&play* rules except as specified in this section.

RULE 6.20

IF the auto-loaded child instrument driver contains any exported user-callable functions **THEN** it **SHALL** also contain and export at least the required instrument driver functions Error Query and Error Message.

PERMISSION 6.2

An auto-loaded child instrument driver may choose not to implement or export other required instrument driver functions.

RECOMMENDATION 6.1

It is recommended that init, close and autoConnect functions not be implemented in auto-loaded child instrument drivers to avoid confusion.

RULE 6.21

IF an auto-loaded child instrument driver implements init and close, **THEN** it **SHALL** be an autonomous instrument driver and **SHALL** implement all of the required instrument driver functions and all modules required of a complete instrument driver

RULE 6.22

IF an auto-loaded child instrument driver implements init and close functions **THEN** functions which specify a first parameter of type ViSession **SHALL** behave correctly when sent the parent's ViSession handle from the parent's init or autoConnect function.

RULE 6.23

IF an auto-loaded child instrument driver does not implement init and close, **THEN** functions which specify a first parameter of type ViSession **SHALL** accept the ViSession handle from the parent's init or autoConnect function.

OBSERVATION 6.7

If an auto-loaded child instrument driver does not implement init and close, it is not required to implement other components required of instrument drivers such as soft front panels.

OBSERVATION 6.8

ADE's which choose to automatically call init and close functions of the parent instrument driver are not required to call the child init and close functions even if they exist.

PERMISSION 6.3

An auto-loaded child instrument driver Function Tree may have no user-callable functions whatsoever (ie the driver just contains functions called directly by the parent and therefore contains no functions in the Function Tree).

RULE 6.24

IF an auto-loaded child instrument driver contains exported functions **THEN** it **SHALL** follow the Function Panel Tree Structure as closely as possible.

OBSERVATION 6.9

An application or user of the child instrument driver which is returned an error or warning from calling a child function, should call the child's Error Query and/or Error Message functions regarding that error or warning, not the parent's error functions.

OBSERVATION 6.10

A function panel file that is loaded as part of an FPAutoLoadList specification is a normal function panel file with no restrictions and may have an FPAutoLoadList specification that references other function panel files. However, recursion is not supported.

Section 7
Function Panel Sub File Format

7.1 Overview

Function Panel Sub files (PREFIX.sub) are useful for describing instrument attribute information, which an ADE can display to the user. For those instrument drivers which rely on an attribute-based architecture for low level communication to an instrument, such as IVI drivers, there tends to be many more attributes than high-level functions. To make a separate function panel and/or a separate function call for each attribute is time consuming for the developer and is cumbersome to the user. The solution to this problem is to implement a set of functions that are used to get and set attribute values.

A function that is used to get or set the value of attributes of a particular datatype is referred to as an attribute accessor function. To provide for type-safety, an instrument driver contains separate attribute accessor functions for getting and setting attributes of a particular datatype. An attribute accessor function takes as arguments a reference to the object whose attribute is to be set or obtained, an identifier indicating which attribute to get or set, and the value of the attribute. Some attributes have a discrete set of valid values. The set of valid values typically differs for each attribute. There are attributes, however, that share a common set of valid values. Boolean attributes, for example, share TRUE and FALSE as valid values.

The current function panel file format allows a developer to present a static help string for each parameter to a function. This is insufficient for attribute accessor functions. It is possible for a single attribute accessor function to be used to set or get the values of many attributes. However, a developer would have to include, in a single static help string, all of the help for all of the attributes of a particular datatype. This problem is compounded for the value parameter. The same value parameter is used to set the value of attributes which have different sets of valid values. A developer would therefore have to include, in a single static help string, all of the help for all of the valid values of all of the attributes that can are accessed through a single function. Thus, an end-user would have to sift through many lines of help that apply to attributes other than the one he or she is trying to get or set. In addition to being cumbersome to the user, this approach is error-prone for the developer. A different function is used to get an attribute than is used to set it. This means that all attribute-specific help would have to be specified twice, once for the get function and once for the set function.

The sub file provides a means for the instrument driver developer to specify help on a per-attribute and a per-value basis. This eliminates the need to specify help more than once for a single attribute or value. The sub file also provides a means for specifying classes of attributes. This allows the developer to organize attributes in a logical, hierarchical manner. The sub file allows the developer to specify help strings for each class of attributes. The sub file also allows the developer to associate sets of valid values with particular attributes. A development environment that implements function panels can parse the sub file and present the information to the user in a user-friendly manner. Instead of presenting a single static help string listing all of the help for all of the attributes, a development environment can show a dialog box containing the hierarchical set of attributes as specified in the sub file. When the user chooses a particular attribute, the function panel can then present help for only the valid values of that attribute.

This section describes the format of a function panel sub file (`PREFIX.sub`). It is intended to be used by applications that create and/or read function panel files and sub files. Designers who are creating instrument drivers should use one of these applications to create their instrument driver files.

RULE 7.1
VXI*plug&play* compliant instrument drivers **SHALL** not be required to have a `.sub` file.

RULE 7.2
VXI*plug&play* compliant ADE's **SHALL** not be required to create or parse `.sub` files.

RULE 7.3
The `PREFIX` of the .sub file **SHALL** be the same as the `PREFIX` of the corresponding instrument driver function panel file.

RULE 7.4
A VXI*plug&play* compliant instrument driver which has a `.sub` file **SHALL** implement the .sub file as defined in this chapter.

7.2 Order of Items in the Sub File

A sub file has a header, value sets, function identifiers, classes and attributes.

RULE 7.5
The items in a sub file **SHALL** be laid out in the following order:

- The sub file header.
- Zero or more Value Sets
- One or more Function Identifiers
- Zero or more Class Identifiers and one or more Attribute Identifiers.

The remainder of this chapter describes the items in a sub file.

7.3 Header

RULE 7.6
A VXI*plug&play* compliant .sub file **SHALL** begin with the following header information.

```
FPAttributeValueFile
n  SubType="IVI"
n  SubVersion="1"
```

7.4 Miscellaneous Sub File Items

The text after the header in the sub file is relatively free form. If the first column is non-whitespace and non-null, then the line starts a new item. The table below lists the characters that are allowed in the first column and the sub file item to which they correspond.

First Column Character	Sub File Item
v	Value Set
0	Function Identifier

1 through 7 inclusive	Class or Attribute Identifier
Whitespace	Continuation of the previous item

RULE 7.7

The character in the first column of each line **SHALL** be either a whitespace, v, or a single digit value between 0 and 7, inclusive.

The remainder of this section describes the format of the above sub file items

7.4.1 Value Sets

A 'v' in the first column indicates a value set.

RULE 7.8

All values sets **SHALL** be implemented as described in this section.

A value set has the following format.

```
v <valueSetTagName> [DataType="<dataTypeIdentifier>"] <constantValueList>
```

The valueSetTagName is the name of the value set. Items later in the file reference a particular value set by the valueSetTagName.

The DataType field specifies the datatype of the value set. The following table lists the possible values for the dataTypeIdentifier.

DataTypeIdentifier	Corresponding VISA Datatype
i	ViInt64, ViInt32, ViInt16, ViBoolean
d	ViReal64
s	ViString, ViConstString

The DataType field is optional. If the DataType field is not used, 'i' is used by default.

A constantValueList follows the DataType field. A constantValueList is a list that specifies all of the values in the value set. Each element in the constantValueList has the following format.

```
<constantName> (<constantValue>) ["<valueHelp>"]
```

The constantName is either a literal value or is a macro that the instrument driver header file defines. The constantValue must be the actual value of the constantName. The datatype of the constantValue is indicated by the DataType field in the value set. No whitespace is allowed within the parentheses. The valueHelp is optional and is a description of the value.

If the valueSetTagName begins with a "X_", then the constantValue's are displayed in hex by ADEs.

The constantName is a simple constant name or a literal value. The constantValue is a literal value. The constantName and constantValue cannot be complex expressions and cannot contain binary or arithmetic operators.

Examples of valid value sets are shown below.

```
v attrFunctionRangeTable DataType="i"
 SUBEXAMP_VAL_DC_VOLTS   (1) "DC Volts"
 SUBEXAMP_VAL_AC_VOLTS   (2) "AC Volts"
 SUBEXAMP_VAL_DC_CURRENT (3) "DC Current"
 SUBEXAMP_VAL_AC_CURRENT (4) "AC Current"
 SUBEXAMP_VAL_2_WIRE_RES (5) "2 Wire Resistance"

v X_attrHexExampleRangeTable DataType="i"
 0      (0x0)
 0x11   (0x11)
 0x18   (0x18)
```

OBSERVATION 7.1

Attributes that have a set of valid values that can be represented by discrete entries often use value sets. On the other hand, attributes that have a continuous range of valid values generally do not have value sets associated with them. The value sets are separated from the attributes in the sub file since one value set can represent valid values for multiple attributes. The value set tag name is used to associate the attribute with the value set. The value sets not only contain a listing of the valid values for a given attribute, but they also provide help for each value in the set.

7.4.2 Function Identifiers

A '0' (zero) in the first column indicates a function identifier.

RULE 7.9

All function identifiers **SHALL** be implemented as described in this section.

A function identifier identifies a particular function in the corresponding function panel file that can access a set of attributes. It also defines the positions of attribute ID and value parameters in the function prototype. A function identifier has the following format.

```
0 <functionName> <attrIDPos> <attrValPos> false <accessMode>
    DataType="<VISAType>"
```

The `functionName` is the name of a particular function including the prefix in the function panel (`PREFIX.fp`) file. The `attrIDPos` is the 1-based position of the attribute ID parameter. The `attrValPos` is the 1-based position of the attribute value parameter.

The `accessMode` specifies the operation that the function performs. The following table lists the possible values for the `accessMode`. The `accessMode` shall be `s` or `g` as defined below.

AccessMode	Function Operation
s	Sets an attribute
g	Gets an attribute

The `VISAType` is the VISA datatype of the attribute value parameter. Possible values for `VISAType` are `ViInt32`, `ViInt64`, `ViReal64`, `ViBoolean`, `ViSession`, and `ViString`.

Examples of valid function identifiers are shown below.

```
0 SubExamp_SetAttributeViInt32 3 4 false s DataType="ViInt32"
0 SubExamp_GetAttributeViInt32 3 4 false g DataType="ViInt32"
0 SubExamp_SetAttributeViReal64 3 4 false s DataType="ViReal64"
0 SubExamp_GetAttributeViReal64 3 4 false g DataType="ViReal64"
```

OBSERVATION 7.2

Function identifiers include the list of functions that use attribute and value set information. In addition to the function name, parameter positions and the attribute datatype are also specified.

RULE 7.10

The PREFIX.sub file SHALL only contain function names that have a matching function in both the PREFIX.fp and PREFIX.h files

RULE 7.11

Function prototypes SHALL include parameters for both an attribute ID and an attribute value. The attribute ID which is the parameter found at the <attrIDPos> SHALL be of type ViAttr. The attribute value, which is the parameter found at the <attrValpos> SHALL have a attribute datatype that matches the type specified by the datatype parameter specified in the function identifier.

RECOMMENDATION 7.1

The access mode and datatype are used to identify the subset of attributes that should be used with a particular attribute accessor function. Only those attributes whose access indicator includes the access mode specified by the function indentifier should be processed with the specified function. For example, a function tagged for setting attributes should only be used with attributes that can be set. Similarly, only those attributes whose datatype matches that of the function identifier should be processed with the specified function.

RECOMMENDATION 7.2

For implementing attribute accessor functions that get and set attributes, the following function identifiers should be used:

0 PREFIX_SetAttributeViInt32 3 4 false s DataType="ViInt32"
0 PREFIX_GetAttributeViInt32 3 4 false g DataType="ViInt32"
0 PREFIX_SetAttributeViInt64 3 4 false s DataType="ViInt64"
0 PREFIX_GetAttributeViInt64 3 4 false g DataType="ViInt64"
0 PREFIX_SetAttributeViReal64 3 4 false s DataType="ViReal64"
0 PREFIX_GetAttributeViReal64 3 4 false g DataType="ViReal64"
0 PREFIX_SetAttributeViSession 3 4 false s DataType="ViSession"
0 PREFIX_GetAttributeViSession 3 4 false g DataType="ViSession"
0 PREFIX_SetAttributeViBoolean 3 4 false s DataType="ViBoolean"
0 PREFIX_GetAttributeViBoolean 3 4 false g DataType="ViBoolean"
0 PREFIX_SetAttributeViString 3 4 false s DataType="ViString"
0 PREFIX_GetAttributeViString 3 5 false g DataType="ViString"

RECOMMENDATION 7.3

In addition to the set and get accessor functions, some developers might choose to implement attribute functions to check the attributes. These functions check the valid ranges for attributes with the “s” access modifier. It is recommended that the following function identifiers be used for this purpose:

0 PREFIX_CheckAttributeViInt32 3 4 false s DataType="ViInt32"
0 PREFIX_CheckAttributeViInt64 3 4 false s DataType="ViInt64"
0 PREFIX_CheckAttributeViReal64 3 4 false s DataType="ViReal64"
0 PREFIX_CheckAttributeViSession 3 4 false s DataType="ViSession"
0 PREFIX_CheckAttributeViBoolean 3 4 false s DataType="ViBoolean"
0 PREFIX_CheckAttributeViString 3 4 false s DataType="ViString"

OBSERVATION 7.3

The above check attribute functions are used in National Instruments IVI drivers.

7.4.3 Class and Attribute Identifiers

A single digit between ‘1’ and ‘7’, inclusive, indicates a class or attribute identifier. The value indicates the level of the class or attribute in the hierarchy of classes and attributes

RULE 7.12

All class and attribute identifiers **SHALL** be implemented as described in this section.

A class identifier has the following format.

```
<classLevel> all "<className>" ["<classHelp>"]
```

The `classLevel` is the level of the class within the hierarchy of classes and attributes. Valid values are a single digit between '1' and '6'., inclusive The `className` is the name of the class of attributes. The `classHelp` is optional help text for the class.

Some examples of valid class identifiers are shown below.

```
1 all "Inherent IVI Attributes"
 "Attributes common to all IVI instrument drivers."

2 all "User Options"
 "Attributes you can set to affect the operation of this instrument driver.\n"
 "    "
```

An attribute identifier has the following format.

```
<attrLevel> all "<attrName>" <attrConstName> <VISAType> <accessIndicator>
       [valueSetTagName] ["<attrHelp>"]
```

The `attrLevel` is the level of the attribute within the hierarchy of classes and attributes. Valid values are a single digit between '1', and '7', inclsuive. The `attrName` is a descriptive name of the attribute. The `attrConstName` is the macro name for the attribute ID as defined in the instrument driver header file.

The The `VISAType` is the datatype of the attribute. Possible values for `VISAType` are `ViInt32`, `ViInt64`, `ViReal64`, `ViBoolean`, `ViSession`, `ViString`, and `ViAddr`. If the `VISAType` is `ViAddr`, then the `accessIndicator` must be `hidden`.

The `accessIndicator` specify whether the user can set or get the attribute or whether the attribute is hidden. The `accessIndicator` are listed below.

AccessIndicator	**Descripion**
s	The attribute is settable
g	The attribute is gettable
sg	The attribute is settable and gettable
gs	The attribute is settable and gettable
hidden	The attribute is hidden and not accessible by the user

The `valueSetTagName` is the name of a particular value set that is defined earlier in the file. If appropriate, the `valueSetTagName` includes the "`X_`" hexadecimal display prefix. The `attrHelp` is optional help text for the attribute.

Some examples of valid attribute identifiers are shown below.

```
2 all "Function" SUBEXAMP_ATTR_FUNCTION ViInt32 gs attrFunctionRangeTable
 "Specifies the measurement function.\n"
 "    "

2 all "Range" SUBEXAMP_ATTR_RANGE ViReal64 gs attrRangeTable
 "Specifies the measurement range."

2 all "Auto Range Value" SUBEXAMP_ATTR_AUTO_RANGE_VALUE ViReal64 g
 "Always returns the actual range the DMM is currently using, even when the "
 "DMM is auto-ranging.\n"
 "    "

2 all "OPC Callback Timeout" SUBEXAMP_ATTR_OPC_TIMEOUT ViInt32 hidden
 "This attribute is hidden.  The driver uses this attribute internally to "
 "set the timeout for the OPC callback.\n"
 "    "
```

OBSERVATION 7.4

Class identifiers provide hierarchical information for separating classes of attributes. Because of this, class identifiers only specify hierarchy level information and a name. The help text associated with the class identifier is optional but often beneficial to the end user. Attribute identifiers, on the other hand, specify more information than the class identifiers. In addition to the attribute name and help description, the attribute identifiers also specify the attribute datatype, whether the attribute is settable and/or gettable, placement in the hierarchy and (optionally) a value set.

RULE 7.13

If a value set tag name is specified in the attribute identifier, then the associated value set SHALL be used to provide the set of available values for the specified attribute.

RECOMMENDATION 7.4

Attributes with a hidden access modifier should NOT be displayed to the end user from within an ADE. Generally, hidden attributes are used for development purposes only and should NOT be exposed to the end-user.

7.5 General Rules and Observations

This section describes general rules regarding sub files.

RULE 7.14

A backslash **SHALL** be used as an escape character.

Examples using a backslash as an escape character are shown below.

`"ab\cd"` is `abcd`
`"ab\"cd"` is `ab"cd`
`"ab\\cd"` is `ab\cd`
`"ab\nd"` is `ab<LF>d` where `<LF>` is ASCII 10

OBSERVATION 7.5

Help text may include the concatenation of string literals. In other words, help text can be continued across lines, but the double quote must be closed on one line and begun again on the next. Help text on a new line without an opening quote (also without a closing quote on the previous line) is interpreted as an embedded new line and indicates the beginning of a new paragraph. The new paragraph should be on a new line and indented when displayed to the user.

RULE 7.15

Double quotes **SHALL** be used only for class description, attribute descriptions, help text, and attribute data types. Only these fields can contain embedded white space.

RULE 7.16

All string comparisons **SHALL** be case sensitive except for the following:

Value set tag names
The Access modifiers for functions and attributes
The "x_" denoting hex display for value sets

7.6 Implementation Details for using sub files in conjunction with fp files

This section is designed to provide guidance to those Application Development Environments that parse and use both sub files and function panel files for end users of instrument drivers. The application should parse and display the attribute list with their corresponding values for a given function. It should be noted that a tool which parses the files for instrument driver developers might have different functionality than what is described below.

RECOMMENDATION 7.5

Any application which uses both sub files and fp files should verify that the fp and sub file correctly reference the same functions and datatypes.

1. An application that reads a function panel sub file should search for function names listed in the function identifier section of the sub files.
2. If the sub file and fp file were developed correctly, then the function names from the sub file have a corresponding function name match in the associated .fp file.
3. If there is a match between function names found in the .fp and sub files, then there are attribute and value controls that match those specified in the .sub file.

RECOMMENDATION 7.6

An ADE application which parses and displays attributes with corresponding values to an end-user in an interactive function panel should take the following guidelines into consideration.

1. An application which provides a list of available attributes in one control should dynamically update the displayed help according to the currently selected attribute.
2. An application which provides attributes in one control and values in a separate control should dynamically update the displayed values and value help according to the currently selected attribute.

7.7 Example .Sub File

```
FPAttributeValueFile
n  SubType="IVI"
n  SubVersion="1"

v attrFunctionRangeTable DataType="i"
```

```
 SUBEXAMP_VAL_DC_VOLTS   (1) "DC Volts"
 SUBEXAMP_VAL_AC_VOLTS   (2) "AC Volts"
 SUBEXAMP_VAL_DC_CURRENT (3) "DC Current"
 SUBEXAMP_VAL_AC_CURRENT (4) "AC Current"
 SUBEXAMP_VAL_2_WIRE_RES (5) "2 Wire Resistance"

v attrRangeTable DataType="d"
 0.1    (1.00000000000000E-1)
 1.0    (1.00000000000000E+0)
 10.0   (1.00000000000000E+1)
 100.0  (1.00000000000000E+2)
 1000.0 (1.00000000000000E+3)

v attrResolutionRangeTable DataType="d"
 SUBEXAMP_VAL_3_5_DIGITS (3.50000000000000E+0) "3-1/2 digits"
 SUBEXAMP_VAL_4_5_DIGITS (4.00000000000000E+0) "4-1/2 digits"
 SUBEXAMP_VAL_5_5_DIGITS (4.50000000000000E+0) "5-1/2 digits"

v Boolean_values DataType="i"
 VI_TRUE  (1) "True"
 VI_FALSE (0) "False"

v attrTriggerSourceTable DataType="s"
 SUBEXAMP_VAL_VAL_CH1       ("\"Ch1\"")           "Channel 1"
 SUBEXAMP_VAL_VAL_CH2       ("\"Ch2\"")           "Channel 2"
 SUBEXAMP_VAL_VAL_IMMEDIATE ("\"VAL_IMMEDIATE\"") "Immediate"
 SUBEXAMP_VAL_EXTERNAL      ("\"VAL_EXTERNAL\"")  "External"

0 SubExamp_SetAttributeViInt32 3 4 false s DataType="ViInt32"
0 SubExamp_GetAttributeViInt32 3 4 false g DataType="ViInt32"
0 SubExamp_SetAttributeViInt64 3 4 false s DataType="ViInt64"
0 SubExamp_GetAttributeViInt64 3 4 false g DataType="ViInt64"
0 SubExamp_SetAttributeViReal64 3 4 false s DataType="ViReal64"
0 SubExamp_GetAttributeViReal64 3 4 false g DataType="ViReal64"
0 SubExamp_SetAttributeViSession 3 4 false s DataType="ViSession"
0 SubExamp_GetAttributeViSession 3 4 false g DataType="ViSession"
0 SubExamp_SetAttributeViBoolean 3 4 false s DataType="ViBoolean"
0 SubExamp_GetAttributeViBoolean 3 4 false g DataType="ViBoolean"
0 SubExamp_SetAttributeViString 3 4 false s DataType="ViString"
0 SubExamp_GetAttributeViString 3 5 false g DataType="ViString"

1 all "Inherent IVI Attributes"
 "Attributes common to all IVI instrument drivers." \n"
 "    "

2 all "User Options"
 "Attributes you can set to affect the operation of this instrument driver.\n"
 "    "

3 all "Cache" SUBEXAMP_ATTR_CACHE ViBoolean gs Boolean_values
 "Specifies whether to cache the value of attributes.  \n"
 "    The instrument driver can choose always to cache or never to cache "
 "particular attributes regardless of the setting of this attribute.\n"
 "    "

2 all "Instrument Capabilities"
 "Attributes that provide information about this instrument driver and the "
 "physical resource it is using.\n"
 "    "

3 all "VISA Resource Manager Session" SUBEXAMP_ATTR_VISA_RM_SESSION ViSession g
 "If a specific driver uses VISA instrument I/O, it passes the value of this "
 "attribute to the viOpen function during initialization.\n"
 "    "

1 all "Basic Operation"
 "Attributes that control the basic features of the DMM.\n"
"    "
```

```
2 all "Function" SUBEXAMP_ATTR_FUNCTION ViInt32 gs attrFunctionRangeTable
 "Specifies the measurement function.\n"
 "    "

2 all "Range" SUBEXAMP_ATTR_RANGE ViReal64 gs attrRangeTable
 "Specifies the measurement range." \n"
 "    "

2 all "Auto Range Value" SUBEXAMP_ATTR_AUTO_RANGE_VALUE ViReal64 g
 "Always returns the actual range the DMM is currently using, even when the "
 "DMM is auto-ranging.\n"
 "    "

2 all "Resolution" SUBEXAMP_ATTR_RESOLUTION ViReal64 gs attrResolutionRangeTable
 "Specifies the measurement resolution in digits of precision.\n"

1 all "Hidden Attributes (not user-viewable)"
 "    "

2 all "OPC Callback Timeout" SUBEXAMP_ATTR_OPC_TIMEOUT ViInt32 hidden
 "This attribute is hidden.  The driver uses this attribute internally to "
 "set the timeout for the OPC callback.\n"
 "    "
```