API Validation Suite
APIVSXXML Specification

Version: 2.0
Save Date: 2010-12-31
Author: Thomas E. Gauger

File Name: APIVSXML.doc
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010/12/31</td>
<td>2.0</td>
<td>Accepted all changes to revision 1.4. Updated based upon actual implementation of APIVSXML.</td>
<td>Thomas E. Gauger nFocal</td>
</tr>
<tr>
<td>2010/08/18</td>
<td>1.4</td>
<td>Accepted all changes made to revision 1.3. Corrected some formatting issues, correct minor problems with examples.</td>
<td>Thomas E. Gauger nFocal</td>
</tr>
<tr>
<td>2010/08/17</td>
<td>1.3</td>
<td>Changes made during development, improved Validation Suite Diagram, added &lt;Abort&gt; element, correct minor typos, added “Hello World” example, added TOD example, added FPUI example, added supported for O_FLAG constants, added strlen() macro, added FIO example, fixed names for FIO_PORT constants, corrected understanding of RQMSG printing and processing, added #FIO_BIT_TEST() macro.</td>
<td>Thomas E. Gauger nFocal</td>
</tr>
<tr>
<td>2010/08/06</td>
<td>1.2</td>
<td>Changes following design review</td>
<td>Thomas E. Gauger nFocal</td>
</tr>
<tr>
<td>2010/08/01</td>
<td>1.1</td>
<td>Additions to constant values</td>
<td>Thomas E. Gauger nFocal</td>
</tr>
<tr>
<td>2010/07/30</td>
<td>1.0</td>
<td>Initial Released Version</td>
<td>Thomas E. Gauger nFocal</td>
</tr>
</tbody>
</table>
# Table of Contents

1. **INTRODUCTION** ............................................................................................................................................... 5
   1.1 PURPOSE ......................................................................................................................................................... 5
   1.2 INTENDED AUDIENCE ..................................................................................................................................... 5
   1.3 SCOPE ............................................................................................................................................................. 5
   1.4 CONVENTIONS ................................................................................................................................................ 5
   1.5 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS........................................................................................... 6
   1.6 REFERENCES ................................................................................................................................................... 7

2. **APIVSXML SCHEMA DEFINITION** .............................................................................................................. 8
   2.1 APIVSXML SYNTAX AND NAMING CONVENTION GUIDELINES ................................................................. 8
      2.1.1 Element Naming Convention ................................................................................................................. 8
      2.1.2 Attribute Naming Convention ................................................................................................................ 8
      2.1.3 API Defined Function Names ................................................................................................................ 8
      2.1.4 API Defined Variable Types .................................................................................................................. 8
      2.1.5 API Defined Constants and VS Defined Constants ................................................................................ 9
      2.1.6 String Constants ..................................................................................................................................... 9
      2.1.7 Numeric Constants ................................................................................................................................. 9
      2.1.8 Macro Definitions .................................................................................................................................. 9
   2.2 VALIDATION SUITE ARCHITECTURE ............................................................................................................. 10
      2.2.1 Validation Suite Description ................................................................................................................ 10
      2.2.2 Validation Suite Structure .................................................................................................................... 10
   2.3 APIVSXML SCHEMA ................................................................................................................................... 12
      2.3.1 APIVSXML Attributes .......................................................................................................................... 12
      2.3.2 Predefined API Values ........................................................................................................................ 20
      2.3.3 APIVSXML Elements ........................................................................................................................... 26

3. **APIVSXML QUICK REFERENCE GUIDE** ....................................................................................................... 41

4. **APIVSXML USAGE EXAMPLES** .................................................................................................................. 42
List of Figures

Figure 1 - Validation Test Suite Structure ................................................................................................................... 10

List of Tables

Table 1. Definitions ....................................................................................................................................................... 6
Table 2. Reference Documents ...................................................................................................................................... 7
1. Introduction

1.1 Purpose

This document describes the APIVSXML schema, syntax, semantics and usage for the API Validation Suite project. The API Validation Suite APIVSXML is the methodology by which test cases are defined for the validation suite and is used by the validation suite software to drive API validation testing.

The APIVSXML describes a programming language that is used to construct a Validation Suite (VS) for the ATC API. As such, it must contain many of the same elements of any other programming language: variables, defined constants, actions, sequences of functions, loops, if-then-else processing, subroutine calls and error handling -- to name a few.

1.2 Intended Audience

This document is of a technical nature. It is intended for members of the cross-functional project team involved in the design, implementation, maintenance, and testing of the API Validation Suite project.

1.3 Scope

This document defines the schema, syntax, semantics and usage of the API Validation Suite APIVSXML that is used to define the actual validation suite test cases and is used by the API Validation Suite VSE to drive these tests.

1.4 Conventions

- Square brackets ([ ]) indicate to-be-determined information which will be resolved before this document is finalized. For example, “The minimum hard disk space is [TBD] MB.”
- Mono-spaced type indicates functions, variables, parameters, statements, commands, and other operating system or programming language constructs. For example, “Use the dir command to list the files and directories.”
- Italicized text (substitution) indicates a substitution parameter. For example, “Use the dir path command to list the files and directories in the given path.”
- Bold type indicates user interface elements or APIVSXML syntax (windows, dialog boxes, menus, and controls). For example, “Press the OK button.” or “the print attribute …”
- Vertical bars (|) are used to indicate a path through a hierarchical user interface. For example, “Select File | Open… to open a file.”
1.5 Definitions, Acronyms, and Abbreviations

The following terms are defined in this document.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>APIVSXML</td>
<td>Application Programming Interface Validation Suite eXtensible Markup Language</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange – Character set utilized</td>
</tr>
<tr>
<td>ATC</td>
<td>Advanced Traffic Controller</td>
</tr>
<tr>
<td>FIO</td>
<td>Field I/O. A library of the ATC API.</td>
</tr>
<tr>
<td>FIOD</td>
<td>Field I/O Device</td>
</tr>
<tr>
<td>FPUI</td>
<td>Front Panel User Interface. A library of the ATC API.</td>
</tr>
<tr>
<td>I/O, IO</td>
<td>Input / Output</td>
</tr>
<tr>
<td>IPC</td>
<td>Inter-Process Communications</td>
</tr>
<tr>
<td>TOD</td>
<td>Time Of Day. A library of the ATC API.</td>
</tr>
<tr>
<td>VD</td>
<td>Virtual Display – Result of VT100 emulator processing of FPUI output.</td>
</tr>
<tr>
<td>VS</td>
<td>Validation Suite</td>
</tr>
<tr>
<td>VSE</td>
<td>Validation Suite Engine Software that utilizes the APIVSXML to validate a manufactures API.</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schema Definition</td>
</tr>
</tbody>
</table>

Table 1. Definitions
1.6 References

The following may be referenced in this document. These documents, of the issue in effect on the date of this specification, form a part of this specification to the extent specified herein. In the event of a conflict between the documents referenced herein and the contents of this specification, the contents of this specification shall be considered a superseding requirement.

<table>
<thead>
<tr>
<th>Source Document</th>
<th>ID Tag</th>
<th>Author</th>
<th>Date</th>
<th>Document Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>A simple explanation and examples of XML</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APIVSXML.xsd</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This document contains the XSD for the APIVSXML</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thomas E. Gauger</td>
<td>20101231</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>API VS Design Specification</td>
<td></td>
<td>Thomas E. Gauger</td>
<td>20101231</td>
<td></td>
</tr>
<tr>
<td>HW_Example.xml</td>
<td></td>
<td>Thomas E. Gauger</td>
<td>20101231</td>
<td>Hello World APIVSXML example, found in etc directory</td>
</tr>
<tr>
<td>TOD_Example.xml</td>
<td></td>
<td>Thomas E. Gauger</td>
<td>20101231</td>
<td>TOD library APIVSXML example, found in etc directory</td>
</tr>
<tr>
<td>FPUI_Example.xml</td>
<td></td>
<td>Thomas E. Gauger</td>
<td>20101231</td>
<td>FPUI library APIVSXML example, found in etc directory</td>
</tr>
<tr>
<td>FIO_Example.xml</td>
<td></td>
<td>Thomas E. Gauger</td>
<td>20101231</td>
<td>FIO library APIVSXML example, found in etc directory</td>
</tr>
</tbody>
</table>

Table 2. Reference Documents
2. APIVSXML Schema Definition

This section defines the schema for the APIVSXML. This schema defines all tags, elements and attributes of the APIVSXML. The document APIVSXML.xsd, defined in the references table, describes the actual XML schema used by the Validation Suite Engine (VSE).

This document does not attempt to teach XML or XSD syntax or usage. It is assumed that the reader is familiar with XML and XSD, their general syntax and rules for defining XML and XSD documents. This document rather describes the specifics of the schema for the APIVSXML and defines elements described in the XSD document that defines this schema. A good simple explanation of XML and its constructs may be found in the references table above.

2.1 APIVSXML Syntax and Naming Convention Guidelines

This section describes the syntax and naming conventions used in defining the APIVSXML. These guidelines are present to ensure consistency in the way the language is defined to improve readability, usability and maintainability. Consistency and orthogonality are important.

2.1.1 Element Naming Convention

Element names, tags, shall use the “Title” or “BumpyName” syntax. An example is:

```xml
<TestCase> … </TestCase>
<Function> … </Function>
```

2.1.2 Attribute Naming Convention

Attributes shall be named using a lower case string. If an attribute name contains more than one word, the attribute shall be named using “lowerCaseTitle” or “bumpyName” syntax. For instance:

```xml
<TestCase description="This is a test case." />
<Function name="fio_register" />
```

2.1.3 API Defined Function Names

All referenced ATC API function names will match their definition in the ATC API. For instance:

```xml
<Function name="fio_register" />
```

2.1.4 API Defined Variable Types

In the course of processing the API, it is necessary to define variables that can be initialized for a function call or where returned values can be stored and referenced by subsequent function calls or expressions. All such variables must be preceded by the ‘$’ character when referenced. The actual variable name may be in upper or lower case and is case sensitive. The variable $ABC is not the same as variable Sabc. It is highly recommended that the user be consistent in defining any such variables. A variables scope is global to a run of the VSE. There is no practical limit to the number of variables that may be defined. All variables must be defined before they are used. All referenced API variable types must match their type definition in the ATC API. If the types mismatch, the VSE will output an error message and will terminate.

For instance:

```xml
<Define type="unsigned char" var="$tmpChar" /> <!-- Defines the variable unsigned char $tmpChar -->
```
2.1.5 API Defined Constants and VS Defined Constants

The API and the Validation Suite define several enumerated values and defined values. These values must be referenced by the VS test cases. All such values must be preceded by the ‘#’ character when referenced. This syntax denotes a named numeric constant. The actual value string, for API defined constants, is consistent with what is defined by the API. For instance:

```xml
<Function p1="#FIO_SP3"/>
```

2.1.6 String Constants

The API and the Validation Suite require character strings to be defined and referenced. These values must be referenced by the VS test cases. All such values must be preceded by the ‘@’ character when referenced. This syntax denotes a character string constant. For instance:

```xml
<Function p1="@This is a character string"/>
```

2.1.7 Numeric Constants

The API and the Validation Suite require numeric constants to be defined and referenced. These values must be referenced by the VS test cases. All such values must be preceded by the ‘%’ character when referenced. This syntax denotes a numeric constant. For instance:

```xml
<Function p1="%25"/>
```

2.1.8 Macro Definitions

The APIVSXML defines several macros used in processing. These macros must be referenced by the VS test cases. All such macros must be preceded by the ‘!’ character when referenced. This syntax denotes a macro. For instance:

```xml
<If expression="!FIO_BIT_TEST($array,$bit) NE #TRUE"/>
```
2.2 Validation Suite Architecture

This section describes the overall Validation Suite (VS) Architecture that the APIVSXML satisfies.

2.2.1 Validation Suite Description

The VS is used to validate a manufactures implementation of the ATC API. XML is utilized because it is both human readable and is usable by a computer program. The APIVSXML is the mechanism by which the VS is defined in a human readable format. The APIVSXML is subsequently used by the Validation Suite Engine (VSE) to actually execute the validation tests that have been described to validate a manufacturer’s implementation of the ATC API. The APIVSXML is easily extensible to include new validation tests as they are identified.

2.2.2 Validation Suite Structure

The following diagram illustrates the structure of the Validation Suite (VS).

![Diagram of Validation Test Suite Structure]

The validation test suite is defined in an APIVSXML document. There may be multiple APIVSXML documents. Which validation test suite to execute is specified as an argument to the VSE at run-time.

Each validation test suite consists of a series of validation test cases. These validation test cases are described and defined in the APIVSXML document. Each validation test case is described by a validation test case set up, validation program and validation test case tear down sequence. A validation program consists of a sequence of APIVSXML statements (elements), arranged in such a way as to validate the ATC API feature being tested.
The “statements” are really APIVSXML elements, that like any other programming language, allows for the execution of all ATC API function interfaces and the validation of all output parameters, function return values, validation of Linux errno, events that should be “seen” or encountered, and observation of current state versus expected state. The APIVSXML is an open ended “programming” language that allows for countless combinations of processing and validation to be performed. The APIVSXML allows for easily updating the test suite as new validation tests are identified.
2.3 APIVSXML Schema

This section discusses the APIVSXML schema defined in the APIVSXML.xsd document.

2.3.1 APIVSXML Attributes

This section describes all attributes that are supported and referenced by the various APIVSXML elements described in section 0 below.

2.3.1.1 Attributes Common to All Elements

This section contains attributes that may be used in any and all APIVSXML elements.

2.3.1.1.1 Attribute: description

The description attribute may be used to add descriptive text to any element. The value associated with this attribute is a free-form text string. This descriptive text is output by the Validation Suite Engine when an element is added to the output XML, during the processing of a validation suite.

2.3.1.2 Attributes that are context specific

The attributes listed in this section are context specific and may only be used with the correct associated APIVSXML element.

2.3.1.2.1 Attribute: name

The name attribute is used to label various APIVSXML elements. In so doing, an element can be defined once and referenced and re-used multiple times like a subroutine call in a programming language. The name space is specific to the APIVSXML element being named. This means that it is possible to have a Test Suite

2.3.1.2.2 Attribute: ref

The ref attribute is used to reference various APIVSXML elements that are named using the name attribute. In so doing, an element can be defined once and referenced and re-used multiple times like a subroutine call in a programming language. All name attribute values must be unique and must be defined before being referenced.
2.3.1.2.3 Attribute: **funcName**

The **funcName** attribute is used to specify the specific ATC API function that is to be called by the VSE. Valid values for **funcName** are:

- fpui_apiver
- fpui_clear
- fpui_clear_tab
- fpui_close
- fpui_close_aux_switch
- fpui_compose_special_char
- fpui_del_keymap
- fpui_display_special_char
- fpui_get_auto_repeat
- fpui_get_auto_scroll
- fpui_get_auto_wrap
- fpui_get_backlight
- fpui_get_character_blink
- fpui_get_cursor
- fpui_get_cursor_blink
- fpui_get_cursor_pos
- fpui_get_focus
- fpui_get_keymap
- fpui_get_led
- fpui_get_reverse_video
- fpui_get_underline
- fpui_get_window_attr
- fpui_get_window_size
- fpui_home
- fpui_open
- fpui_open_aux_switch
- fpui_poll
- fpui_read
- fpui_read_aux_switch
- fpui_read_char
- fpui_read_string
- fpui_refresh
- fpui_reset_all_attributes
- fpui_reset_keymap
- fpui_set_auto_repeat
- fpui_set_auto_scroll
- fpui_set_auto_wrap
- fpui_set_backlight
- fpui_set_backlight_timeout
- fpui_set_character_blink
- fpui_set_cursor
- fpui_set_cursor_blink
- fpui_set_cursor_pos
- fpui_set_emergency
- fpui_set_keymap
- fpui_set_led
- fpui_set_reverse_video
- fpui_set_tab
- fpui_set_underline
• fpui_set_window_attr
• fpui_write
• fpui_write_at
• fpui_write_char
• fpui_write_char_at
• fpui_write_string
• fpui_write_string_at

• fio_apiver
• fio_deregister
• fio_fiod_channel_map_count
• fio_fiod_channel_map_get
• fio_fiod_channel_map_set
• fio_fiod_channel_request_get
• fio_fiod_channel_request_set
• fio_fiod_cmu_dark_channel_get
• fio_fiod_cmu_dark_channel_set
• fio_fiod_cmu_fault_get
• fio_fiod_cmu_fault_set
• fio_fiod_disable
• fio_fiod_enable
• fio_fiod_frame_notify_deregister
• fio_fiod_frame_notify_register
• fio_fiod_frame_read
• fio_fiod_frame_schedule_get
• fio_fiod_frame_schedule_set
• fio_fiod_frame_size
• fio_fiod_inputs_filter_get
• fio_fiod_inputs_filter_set
• fio_fiod_inputs_get
• fio_fiod_inputs_trans_get
• fio_fiod_inputs_trans_read
• fio_fiod_inputs_trans_set
• fio_fiod_mmu_flash_bit_get
• fio_fiod_mmu_flash_bit_set
• fio_fiod_outputs_get
• fio_fiod_outputs_reservation_get
• fio_fiod_outputs_reservation_set
• fio_fiod_outputs_set
• fio_fiod_register
• fio_fiod_status_get
• fio_fiod_status_reset
• fio_fiod_ts_fault_monitor_get
• fio_fiod_ts_fault_monitor_set
• fio_fiod_ts1_volt_monitor_get
• fio_fiod_ts1_volt_monitor_set
• fio_fiod_wd_deregister
• fio_fiod_wd_heartbeat
• fio_fiod_wd_register
• fio_fiod_wd_reservation_get
• fio_fiod_wd_reservation_set
• fio_hm_deregister
- fio_hm_fault_reset
- fio_hm_heartbeat
- fio_hm_register
- fio_query_fiod
- fio_query_frame_notify_status
- fio_register

- tod_cancel_onchange_signal
- tod_cancel_tick_signal
- tod_get
- tod_get_dst_info
- tod_get_dst_state
- tod_get_timesrc
- tod_get_timesrc_freq
- tod_request_onchange_signal
- tod_request_tick_signal
- tod_set
- tod_set_dst_info
- tod_set_dst_state
- tod_set_timesrc
2.3.1.2.4 Attribute: type

The **type** attribute is used to specify the type of variable to be created in a `<Define>` element. Valid values for **type** are:

- char
- char []
- unsigned char
- unsigned char []
- boolean
- ssize_t
- int
- unsigned int
- fpui_handle
- O_FLAGS
- FIO_APP_HANDLE
- FIO_DEV_HANDLE
- FIO_VERSION
- FIO_VIEW
- FIO_CHANNEL_MAP
- FIO_CMU_DC_MASK
- FIO_CMU_FSA
- FIO_NOTIFY
- FIO_FRAME_SCHD
- FIO_INPUT_FILTER
- FIO_INPUTS_TYPE
- FIO_TRANS_STATUS
- FIO_TRANS_BUFFER
- FIO_MMU_FLASH_BIT
- FIO_PORT
- FIO_DEVICE_TYPE
- FIO_FIOD_STATUS
- FIO_TS_FM_STATE
- FIO_TSI_VM_STATE
- FIO_NOTIFY_INFO
- struct timeval
- dst_info_t

2.3.1.2.5 Attribute: var

The **var** attribute is used to reference a variable that was created using `<Define>`. Since this is a reference, the variable must appear is $variable.

2.3.1.2.6 Attribute: value

The **value** attribute is used to set a value to a $variable in a `<Set>` element.

2.3.1.2.7 Attribute: size

The **size** attribute is used to specify the number of occurrences of **type** for this **var** in a `<Define>`.
2.3.1.2.8 Attribute: operation

The *operation* attribute is used to specify what is to be done in a `<Set>` element. Valid values for *operation* are:

- (not present) – A set var equal to value is performed.
- add – value is added to var
- subtract – value is subtracted from var

2.3.1.2.9 Attribute: member

The *member* attribute is used to reference a member of a structure that was created using `<Define>`. The member name is given as the value of the attribute.

2.3.1.2.10 Attribute: signal

The *signal* attribute is used to specify the Linux signal that is to be handled. Either a variable, constant or defined constant may be referenced.

2.3.1.2.11 Attribute: action

The *action* attribute is used to specify what is to be done with the signal in question. Valid values for *action* are:

- enable
- disable

2.3.1.2.12 Attribute: return

The *return* attribute is used to reference a variable that was created using `<Define>`, which is used to place the return value of a `<Function>`. Since this is a reference, the variable must appear is $variable.

2.3.1.2.13 Attribute: errno

The *errno* attribute is used to reference a variable that was created using `<Define>`, which is used to place the errno value of a `<Function>` into. Since this is a reference, the variable must appear is $variable.

2.3.1.2.14 Attribute: p1

The *p1* attribute is used to reference a variable, constant or defined constant that is to be passed to a `<Function>`. Since this is a reference, a variable must appear is $variable.

2.3.1.2.15 Attribute: p2

The *p2* attribute is used to reference a variable, constant or defined constant that is to be passed to a `<Function>`. Since this is a reference, a variable must appear is $variable.

2.3.1.2.16 Attribute: p3

The *p3* attribute is used to reference a variable, constant or defined constant that is to be passed to a `<Function>`. Since this is a reference, a variable must appear is $variable.

2.3.1.2.17 Attribute: p4

The *p4* attribute is used to reference a variable, constant or defined constant that is to be passed to a `<Function>`. Since this is a reference, a variable must appear is $variable.

2.3.1.2.18 Attribute: p5

The *p5* attribute is used to reference a variable, constant or defined constant that is to be passed to a `<Function>`. Since this is a reference, a variable must appear is $variable.
2.3.1.2.19  Attribute: p6
The p6 attribute is used to reference a variable, constant or defined constant that is to be passed to a <Function>. Since this is a reference, a variable must appear as $variable.

2.3.1.2.20  Attribute: expression
The expression attribute is used to define a boolean expression to be evaluated. References to variables must appear as $variable, references to numeric constants must appear as %number, references to named constants must appear as #constant, references to strings must appear as @string, and references to macros must appear as !macro().

2.3.1.2.21  Attribute: setUp
The setUp attribute is used to reference a <SetUp> element. In so doing, an element can be defined once and referenced and re-used multiple times like a subroutine call in a programming language. All <SetUp> name attribute values must be unique and must be defined before they are referenced.

2.3.1.2.22  Attribute: tearDown
The tearDown attribute is used to reference a <TearDown> element. In so doing, an element can be defined once and referenced and re-used multiple times like a subroutine call in a programming language. All <TearDown> name attribute values must be unique and must be defined before they are referenced.

2.3.1.2.23  Attribute: status
The status attribute is used to specify special PASS / FAIL notification in the log file. Valid values for status are:

- FAIL – Causes a -1 (non-conformance) to be returned to the calling environment.
- PASS – Causes a 0 (conformance) to be returned to the calling environment. This is the default status, unless “FAIL” is specified.

2.3.1.2.24  Attribute: time
The time attribute is used to specify a time in seconds. The value of the attribute must be a numeric constant, integer variable or named constant.

2.3.1.2.25  Attribute: file
The file attribute is used to specify the input file to be used.

2.3.1.2.26  Attribute: frame
The frame attribute is used to specify the request frame to apply FIO input to.

2.3.1.2.27  Attribute: level
The level attribute is used to specify the output processing level that the APIVSXML element in question is to be processed at. Only some elements contain this attribute. Valid values for level are:

- always – Always process this element regardless of output level.
- conformance – Only process this element when running at the conformance level.
- summary – Only process this element when running at the conformance or summary level. This is the default level if this attribute is not given for an element that uses this attribute.
- trace – Process this element only when running at the trace level
2.3.1.2.28  Attribute: dump
The dump attribute is used to specify the type of output for a <Dump> element. Valid values for dump are:

- VD – A dump of the Virtual Display (VD) is performed.
- CMDMSG – A dump of a FIO Command Message is performed.

2.3.1.2.29  Attribute: sequence
The sequence attribute is used to specify a sequence number to be appended to a dump file name in a <Dump> element.

2.3.1.2.30  Attribute: force
The force attribute is used to specify if a <Dump> should occur, regardless of the “capture” mode being used. This attribute is useful to ensure the capturing of <Dump> elements when a failure is present; otherwise <Dump> elements are only normally processed during “capture” mode. Valid values for force are:

- #TRUE – Always perform dump.
- #FALSE – (default) Only perform dump during capture mode.

2.3.1.2.31  Attribute: load
The load attribute is used to specify the type of input for a <Load> element. Valid values for load are:

- VD – A load of the Virtual Display (VD) is performed.
- CMDMSG – A load of a FIO Command Message is performed.

2.3.1.2.32  Attribute: flip
The flip attribute is used to specify the type of flip for a <Aux> element. Valid values for flip are:

- #ON – Flip the Auxiliary Switch On.
- #OFF – Flip the Auxiliary Switch Off.
2.3.2 Predefined API Values

This section describes all constants and macros that are supported and referenced by the various APIVSXML elements described in section 2.3.3 APIVSXML Elements below.

2.3.2.1 VSE Named Constants

This section describes named constants that are supported by the Valid Suite Engine (VSE). All named constants are referenced by the preceding a constant with a ‘#’ symbol.

2.3.2.1.1 #NULL

This constant is used to test / compare / set a pointer, returned by an ATC API or to be passed to the ATC API, the special value 0x00000000 or <null>.

2.3.2.1.2 #TRUE

This constant is used to test / compare / set a boolean to a value of 0x00000001 or <true>.

2.3.2.1.3 #FALSE

This constant is used to test / compare / set a boolean to a value of 0x00000000 or <false>.

2.3.2.1.4 #ON

This constant is used to test / compare / set a boolean to a value of 0x00000001 or <true>.

2.3.2.1.5 #OFF

This constant is used to test / compare / set a boolean to a value of 0x00000000 or <false>.

2.3.2.1.6 #ROWS

This constant is the value of the number of rows in Virtual Display set in the configuration file for this VSE run.

2.3.2.1.7 #COLUMNS

This constant is the value of the number of columns in Virtual Display set in the configuration file for this VSE run.

2.3.2.1.8 Errno Constants

The VSE supports the errno constants by referencing the decimal equivalent of the errno being utilized, utilizing the %number syntax.

2.3.2.1.9 SIGNAL Constants

The VSE supports the signal value constants:

- #FIO_SIGIO
- #SIGWINCH
- Other signals may be referenced using their decimal equivalent and the %number syntax.
2.3.2.1.10 O_FLAGS Constants

The VSE supports the Linux O_FLAGS constants. Values may be OR’ED together using `<Set operation="add" />` element. If a constant name is not supported by the VSE, the O_FLAGS value may be specified in using the decimal equivalent `%number` syntax. The constant values supported are:

- `#O_RDWR`
- `#O_RDONLY`
- `#O_WRONLY`
- `#O_DIRECT`
- `#O_NONBLOCK`

2.3.2.2 VSE Macros

This section defines macros that are supported by the VSE to allow processing of information that the VSE maintains or collects during the course of running an APIVSXML “program”. All macros are preceded by the ‘!’ character.

2.3.2.2.1 Generally Useful Macros

During the course of VSE processing, certain operations will make processing easier. The macros listed below are generally useful for VSE processing in general. The available macros are:

- `!strlen(char [])`
  This macro may be used to return the string length of a variable using type “char []”. The value returned is of type “int”.

- `!sizeof($var)`
  This macro may be used to return the number of elements of $var, as configured using the `size` attribute in a `<Define>`. The value returned is of type “int”.

2.3.2.2.2 FPUI Macros

During the course of processing FPUI API calls, it is necessary to compare the current state of the Virtual Display (VD) generated by the VT100 emulator against what the current VD is expected to contain. In order to accomplish this comparison, the following macro is defined:

- `!VDCompare(@expected_display_file)`
  This macro will return #TRUE if the current contents of the VD match the information contained in the “expected_display_file”. Otherwise, this macro returns #FALSE. The “@expected_display_file” is the path to the file to be used for comparison to the VD.

2.3.2.2.3 FIO Macros

During the course of processing FIO API calls, it is necessary to compare the last SDLC Command Message (CMDMSG) that was received on the FIO Loopback Port, for the given frame_number, against what the last CMDMSG was expected to be and contain. In order to accomplish this comparison, the following macro is defined:

- `!CMDMSGCompare(@expected_message_file,%frame_number)`
  This macro will return #TRUE if the last CMDMSG received on the FIO Loopback Port matches the information contained in “expected_message_file”. Otherwise, this macro returns #FALSE. The “@expected_message_file” is the path to the file to be used for comparison to the last CMDMSG received.

During processing of output and input points, it is necessary to test bits in a bit array. The following macro supports testing bits in a bit array:

- `!FIO_BIT_TEST($array,%bit)`
This macro will return #TRUE if the bit is set in the bit array, otherwise #FALSE is returned.

2.3.2.3 ATC API Constants

This section describes constants that are support by the VSE that are defined by the ATC API itself.

2.3.2.3.1 FPUI Constants

There are no constants that are just specific to the FPUI.

2.3.2.3.2 FIO Constants

This section describes constants that are specific to the FIO.

2.3.2.3.2.1 FIO_VERSION Constants

The constants that are valid for FIO_VERSION are:

- #FIO_VERSION_LIBRARY
- #FIO_VERSION_LKM

2.3.2.3.2.2 FIO_VIEW Constants

The constants that are valid for FIO_VIEW are:

- #FIO_VIEW_APP
- #FIO_VIEW_SYSTEM

2.3.2.3.2.3 FIO_CMU_DC_MASK Constants

The constants that are valid for FIO_CMU_DC_MASK are:

- #FIO_CMU_DC_MASK1
- #FIO_CMU_DC_MASK2
- #FIO_CMU_DC_MASK3
- #FIO_CMU_DC_MASK4

2.3.2.3.2.4 FIO_CMU_FSA Constants

The constants that are valid for FIO_CMU_FSA are:

- #FIO_CMU_FSA_NONE
- #FIO_CMU_FSA_NON_LATCHING
- #FIO_CMU_FSA_LATCHING

2.3.2.3.2.5 FIO_NOTIFY Constants

The constants that are valid for FIO_NOTIFY are:

- #FIO_NOTIFY_ONCE
- #FIO_NOTIFY_ALWAYS

2.3.2.3.2.6 FIO_INPUTS_TYPE Constants

The constants that are valid for FIO_INPUTS_TYPE are:

- #FIO_INPUTS_RAW
- #FIO_INPUTS_FILTERED
2.3.2.3.2.7  FIO_TRANS_STATUS Constants

The constants that are valid for FIO_TRANS_STATUS are:

- #FIO_TRANS_SUCCESS
- #FIO_TRANS_FIOD_OVERRUN
- #FIO_TRANS_APP_OVERRUN

2.3.2.3.2.8  FIO_MMU_FLASH_BIT Constants

The constants that are valid for FIO_MMU_FLASH_BIT are:

- #FIO_MMU_FLASH_BIT_ON
- #FIO_MMU_FLASH_BIT_OFF

2.3.2.3.2.9  FIO_PORT Constants

The constants that are valid for FIO_PORT are:

- #FIO_SP3
- #FIO_SP5
- #FIO_SP8
2.3.2.3.2.10 FIO_DEVICE_TYPE Constants

The constants that are valid for FIO_DEVICE_TYPE are:

- #FIO332
- #FIOTS1
- #FIOTS2
- #FIOMMU
- #FIODR1
- #FIODR2
- #FIODR3
- #FIODR4
- #FIODR5
- #FIODR6
- #FIODR7
- #FIODR8
- #FIOTF1
- #FIOTF2
- #FIOTF3
- #FIOTF4
- #FIOTF5
- #FIOTF6
- #FIOTF7
- #FIOTF8
- #FIOCMU
- #FIOINSIU1
- #FIOINSIU2
- #FIOINSIU3
- #FIOINSIU4
- #FIOINSIU5
- #FIOOUT6SIU1
- #FIOOUT6SIU2
- #FIOOUT6SIU3
- #FIOOUT6SIU4
- #FIOOUT14SIU1
- #FIOOUT14SIU2

2.3.2.3.2.11 FIO_TS_FM_STATE Constants

The constants that are valid for FIO_TS_FM_STATE are:

- #FIO_TS_FM_ON
- #FIO_TS_FM_OFF

2.3.2.3.2.12 FIO_TS1_VM_STATE Constants

The constants that are valid for FIO_TS1_VM_STATE are:

- #FIO_TS1_VM_ON
- #FIO_TS1_VM_OFF
2.3.2.3.2.13 FIO size Constants

The constants that are valid for FIO sizes are:

- #FIO_INPUT_POINTS_BYTES
- #FIO_OUTPUT_POINTS_BYTES
- #FIO_CHANNEL_BYTES

2.3.2.3.3 TOD Constants

This section describes constants that are specific to the TOD.

2.3.2.3.4 TOD_TIMESRC_ENUM Constants

The constants that are valid for TOD_TIMESRC_ENUM are:

- #TOD_TIMESRC_LINESYNC
- #TOD_TIMESRC_RTCSQWR
- #TOD_TIMESRC_CRYSTAL
- #TOD_TIMESRC_EXTERNAL1
- #TOD_TIMESRC_EXTERNAL2
2.3.3 APIVSXML Elements

This section describes all elements and corresponding legal attributes that are defined as part of the APIVSXML.

2.3.3.1.1 Block set up

At the start of a block, the attribute “setUp” may be used to specify a set up subroutine that may be utilized to perform initialization (C++ constructor like processing) for the block. Common sequences of ATC API functions may be processed in a set up subroutine, such as, registering for services and initialization of variables.

2.3.3.1.2 Block tear down

At the end of a block, the attribute “tearDown” may be used to specify a tear down subroutine that may be utilized to perform clean up (C++ destructor like processing) for the block. Tear down subroutines are commonly used for deregistering of services.
2.3.3.2 APIVSXML Elements

This section describes all elements of the APIVSXML.

2.3.3.3 APIVSXML Structure Elements

This section defines APIVSXML elements that define the structure of a validation test suite.

2.3.3.3.1 Element: `<APIVSXML>`

The topmost, and root, element of an APIVSXML document is `<APIVSXML>`.

2.3.3.3.2 Element: `<TestSuite>`

A `<TestSuite>` defines all of the validation `<TestCase>` elements that encompass the validation test suite. A `<TestSuite>` element defines a block. Unless `<Abort status="FAIL"/>` has been called, the test suite exits with a “PASS” status.

Valid attributes for the `<TestSuite>` element are:

- **name**: REQUIRED – This attribute is a string that is referenced when the VSE is invoked to specify a validation test suite to be executed.
- **setUp**: optional – This attribute is a string that references a set up routine (defined with the `<SetUp>` element). This routine is called at the start of the test suite.
- **tearDown**: optional – This attribute is a string that references a tear down routine (defined with the `<TearDown>` element). This routine is called at the end of the test suite.
- **description**: optional – This attribute is used to add descriptive text to a test suite.

The `<TestSuite>` element then may contain an unbounded set of `<TestCaseRef>` elements. These `<TestCaseRef>` elements are executed sequentially until either an `<Abort>` element is encountered or all of the test cases have been executed.

Example:

```xml
<TestSuite name="Validation Test Suite #1" 
    setUp="TestSuiteSetUp"
    tearDown="TestSuiteTearDown"
    description="Validation Test Suite #1" >

    <TestCaseRef ref="Test #1" />
    <TestCaseRef reg="Test #2" />
    <TestCaseRef reg="Test #N" />

</TestSuite>
```
2.3.3.3.3 Element: <TestCaseRef>

A <TestCaseRef> element references a <TestCase> element that actually tests and validates an ATC API feature under test (validation).

Valid attributes for the <TestCaseRef> element are:

- **ref**: REQUIRED – This attribute is a string that references a named <TestCase> element. The value of the ref attribute must match the value of a name attribute for one of the named <TestCase> elements in the document. <TestCase> must be defined before it is referenced.

Example:

```xml
<TestCaseRef ref="Test #1" />
<TestCaseRef ref="Test #2" />
<TestCaseRef ref="Test #N" />
```

2.3.3.3.4 Element: <TestCase>

A <TestCase> defines the sequence of steps that are executed by the VSE to perform validation of an ATC API feature. A <TestCase> element defines a block.

Valid attributes for the <TestCase> element are:

- **name**: REQUIRED – This attribute is a string that is referenced by a <TestCaseRef> element.
- **setUp**: optional – This attribute is a string that references a set up routine (defined with the <SetUp> element). This routine is called at the start of the test case.
- **tearDown**: optional – This attribute is a string that references a tear down routine (defined with the <TearDown> element). This routine is called at the end of the test case.
- **description**: optional – This attribute is used to add descriptive text to a test case. This description is output into the log file, by the VSE, at the appropriate time.

The <TestCase> element then may contain an unbounded set of <Statement> elements. These <Statement> elements are executed sequentially until either an <Abort> element is encountered or all of the statements have been executed. The default exit status of a <TestCase> is “PASS”, unless otherwise specifically stated in an <Abort> call.

Example:

```xml
<TestCase name="Test #1"
        setUp="TestCase1SetUp"
        tearDown="TestCase1TearDown"
        description="Test #1">
    <Call ref="FIORegisterDeregisterSub"
          description="Call subroutine to register and then deregister with FIO"/>
</TestCase>
```

2.3.3.3.5 Element: <Statement>

A <Statement> element is substituted with any of the APIVSXML elements listed in sections below.
2.3.3.4 APIVSXML Statement Primitive Elements

This section describes APIVSXML elements that define the primitive statement elements of the APIVSXML programming language.

2.3.3.4.1 Element: `<Call>`

The `<Call>` element is used to make a call to a `<Subroutine>` element.

Valid attributes for the `<Call>` element are:

- `ref`: REQUIRED – This attribute is a string that references a named `<Subroutine>` element. The value of the `ref` attribute must match the value of the `name` attribute for one of the named `<Subroutine>` elements in the document.
- `setUp`: optional – This attribute is a string that references a set up routine (defined with the `<SetUp>` element). This routine is called at the start of the subroutine.
- `tearDown`: optional – This attribute is a string that references a tear down routine (defined with the `<TearDown>` element). This routine is called at the end of the subroutine.
- `description`: optional – This attribute is used to add descriptive text to a call. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<Call ref="FIORegisterDeregisterSub" description="Call subroutine to register and then deregister with FIO" />
```

2.3.3.4.2 Element: `<Subroutine>`

The `<Subroutine>` element is used to create a logical collection of `<Statement>` elements. A `<Subroutine>` element defines a block.

Valid attributes for the `<Subroutine>` element are:

- `name`: REQUIRED – This attribute is a string that is referenced by a `<Call>` element.
- `description`: optional – This attribute is used to add descriptive text to a subroutine. This description is output into the log file, by the VSE, at the appropriate time.

The `<Subroutine>` element then may contain an unbounded set of `<Statement>` elements. These `<Statement>` elements are executed sequentially until either an `<Abort>` element is encountered or all of the statements have been executed.

Example:

```xml
<Subroutine name="FIORegisterDeregisterSub"
            setUp="FIORegDeregSetUp"
            tearDown="FIORegDeregTearDown"
            description="Subroutine to process FIO register / deregister validation">
    <Function name="fio_register"
              return="SFIO_HANDLE"
              errno="Serrno" />

    <Function name="fio_deregister"
              return="SreturnInt"
              errno="Serrno"
              p1="SFIO_HANDLE" />

</Subroutine>
```
2.3.3.4.3  Element: <Function>

The `<Function>` element is used to execute an ATC API function.

Valid attributes for the `<Function>` element are:

- **funcName**: REQUIRED – This attribute is a string that specifies the ATC API function to call.
- **description**: optional – This attribute is used to add descriptive text to a function call. This description is output into the log file, by the VSE, at the appropriate time.
- **return**: optional – This attribute specifies where the functions return value should be stored. If this attribute is omitted, the return value is ignored. If this attribute is specified, the variable referenced by the string value must be of the same type as the return value of the function. If a mismatch is detected, the VSE will output an error message and terminate.
- **errno**: optional – This attribute specifies where the errno value should be stored. If this attribute is omitted, the errno value is ignored. If this attribute is specified, the variable referenced by the string value must be of type integer. If a mismatch is detected, the VSE will output an error message and terminate.
- **p1 – p6**: optional – These attributes specify the parameters to be passed to the ATC API function call. Parameters are passed in the order specified by the p#. Each p# may be referenced only once. If a p# is referenced more than once or is absent, the VSE will output an error message and terminate. The variable or constant referenced by the string value must be of the same type as the ATC API function parameter. If a mismatch is detected, the VSE will output an error message and terminate.

Example:

```xml
<Function name="fio_register"
    return="SFIO_HANDLE"
    errno="$errno" />

<Function name="fio_deregister"
    return="$returnInt
    errno="$errno"
    p1="SFIO_HANDLE"/>
```
2.3.3.4 Element: <If>

The <If> element is used to perform validation of returned information (return code, errno and output variables) from a <Function> or a series of <Function> elements.

Valid attributes for the <If> element are:

- **expression**: REQUIRED – This attribute is a boolean expression of information that is to be validated. If the boolean expression evaluates to true, the <Then> element is executed, otherwise the <Else> element is executed. The <Then> and <Else> elements are optional. If they are not present, processing continues with the next <Statement>.

- **description**: optional – This attribute is used to add descriptive text to a <If> element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<If expression="$FIO_HANDLE EQ #NULL” >
  <Then>
    <Print >FIO_HANDLE is NULL ... Error</Print>
    <Abort status="FAIL”/>
  </Then>
</If>

<If expression="$errno NE %0” >
  <Then>
    <Print >Errno is not 0 ... Error</Print>
    <Abort status =” FAIL”/>
  </Then>
</If>

<If expression="!VDCompare(@cleared_screen) EQ #FALSE”>
  <Then>
    <Print >Display is not cleared, contents of display</Print>
    <Dump force="#TRUE” dump="VD” file="@cleared_display_fail” />
    <Abort status =” FAIL”/>
  </Then>
</If>

<If expression="!CMDMSGCompare(@message_received,$frame) EQ #FALSE”>
  <Then>
    <Print >The correct message was not received</Print>
    <Dump force="#TRUE” dump="CMDMSG” file="message_received_fail” />
    <Abort status =” FAIL”/>
  </Then>
</If>
```
2.3.3.4.1 Element: <Then>

The <Then> element is used to perform “true” processing, following a <If> element evaluation.

Valid attributes for the <Then> element are:

- **description**: optional – This attribute is used to add descriptive text to a <Then> element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```
<If expression="$errno NE %0" >
  <Then>
    <Print >FIO_HANDLE is NULL … Error</Print>
    <Abort status =" FAIL"/>
  </Then>
</If>
```
2.3.3.4.4.2 Element: <Else>

The <Else> element is used to perform “false” processing, following a <If> element evaluation. An <Else> element defines a block.

Valid attributes for the <Else> element are:

- **description**: optional – This attribute is used to add descriptive text to a <Else> element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<If expression="$FIO_HANDLE NE %0" >
   <Else description="FIO_HANDLE is valid" />
</If>
```

2.3.3.4.5 Element: <While>

The <While> element is used to perform a loop of <Statement> elements.

Valid attributes for the <While> element are:

- **expression**: REQUIRED – This attribute is a boolean expression of information that is to be validated. If the boolean expression evaluates to true, the <Statement> elements in the loop are executed. If the boolean expression evaluates to false, the <Statement> following the loop is executed.

- **description**: optional – This attribute is used to add descriptive text to a <While> element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<Define type="int" var="$ii" />  
<SetUp name="setUpii" >  
   <Set var="$ii" value="%10" />  
</SetUp>  
<While expression="$ii NE %0" >  
   <Set var="$ii" operation="subtract" value="%1" />  
</While>  
<Print >Loop is complete</Print>
```

2.3.3.4.6 Element: <FPUIInput>

The <FPUIInput> element is used to place input data on the FPUI Loopback Port, to be processed by FPUI read functions.

Valid attributes for the <FPUIInput> element are:

- **file**: REQUIRED – This attribute is a string that references a human-readable flat file that contains the data to be placed on the FPUI Loopback Port, that will be subsequently processed by a FPUI read call.

- **description**: optional – This attribute is used to add descriptive text. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<FPUIInput file="@FPUI_Input_Keystrokes"  
   description="Input keystrokes from the front panel" />
```
2.3.3.4.7 Element: `<FIOResponse>`

The `<FIOResponse>` element is used to pre-load SDLC response message data for a given response frame. The FIO emulator will send this response frame when the correlated command message is received for the ATC FIOM.

Valid attributes for the `<FIOResponse>` element are:

- **file**: REQUIRED – This attribute is a string that references a human-readable flat file that contains the data to be pre-loaded for a given response frame.
- **frame**: REQUIRED – This attribute indicates the response frame that this input applies to.
- **description**: optional – This attribute is used to add descriptive text. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<FIOInput input="@FIO_Response_Message" frame="%183" description="SDLC Response Message" />
```

2.3.3.4.8 Element: `<Aux>`

The `<Aux>` element is used to simulate the flipping of the Auxiliary Switch. The VT100 emulator will send the appropriate `<ESC>` sequences to the ATC, via the FP Loopback Cable, when the switch is flipped.

Valid attributes for the `<Aux>` element are:

- **flip**: REQUIRED – This attribute specifies the state of the Auxiliary Switch.
- **description**: optional – This attribute is used to add descriptive text. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<Aux flip="#ON" />
```

2.3.3.4.9 Element: `<Signal>`

The `<Signal>` element is used set up signal processing by a `<SignalHandler>` element.

Valid attributes for the `<Signal>` element are:

- **signal**: REQUIRED – This attribute indicates the signal number that should be handled.
- **action**: REQUIRED – This attribute is used to enable or disable handling of the indicated signal.
- **ref**: optional – This attribute is a string that references a named `<SignalHandler>` element. The value of the ref attribute must match the value of the name attribute for one of the named `<SignalHandler>` elements in the APIVSXML document. The ref attribute is required if the action element is “enabled”. A name must be defined before it is referenced.
- **description**: optional – This attribute is used to add descriptive text to a call. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<Signal signal="#SIGFIO" action="enable" ref="SigHandlerFIO" description="Handle the SIGFIO signal" />
<Signal signal="#SIGFIO" action="disable" />
```
2.3.3.4.10 Element: `<SignalHandler>`

The `<SignalHandler>` element is used to define a signal handler. What signals are processed by this handler is defined by using the `<Signal>` statement.

Valid attributes for the `<SignalHandler>` element are:

- **name**: REQUIRED – This attribute is a string that is referenced by a `<Signal>` element.
- **description**: optional – This attribute is used to add descriptive text to a signal handler. This description is output into the log file, by the VSE, at the appropriate time.

The `<SignalHandler>` element then may contain an unbounded set of `<Statement>` elements. These `<Statement>` elements are executed sequentially until either an `<Abort>` element is encountered or all of the statements have been executed.

Example:

```xml
<SignalHandler name="SigHandlerFIO" description="Signal Handler for handling SIGFIOs">
  <Print>fio notify has occurred</Print>
  <Set var="$sig_occurred" value="#TRUE" />
</SignalHandler>
```

In order to capture information from a `<SignalHandler>` into the VSE foreground environment `<Set>` a variable in the `<SignalHandler>` element can then be tested in the VSE foreground and take appropriate action.

2.3.3.5 APIVSXML Variable Elements

This section describes APIVSXML elements that are used to define and manipulate variables.

2.3.3.5.1 Element: `<Define>`

The `<Define>` element is used to define a variable to be used during processing. A variable must be defined before it is referenced. When a variable is referenced, it must be preceded by a `$`, to indicate a reference to a defined variable.

Valid attributes for the `<Define>` element are:

- **var**: REQUIRED – This attribute indicates the variable to initialize. The variable must first the created using `<Define>` and the type of variable must match the context in which the variable is initialized or an error will occur and the VSE will terminate.
- **type**: REQUIRED – This attribute indicates the type of variable to create. When a variable is referenced, the type of variable must match the context in which the variable is referenced or an error will occur and the VSE will terminate.
- **size**: optional – This attribute is used to define the number of elements to be contained in the variable.
- **description**: optional – This attribute is used to add descriptive text to a `<Define>` element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<Define var="$ii" type="int" />
<Define var="$FIO_HANDLE" type="FIO_APP_HANDLE" />
<Define var="$timeval" type="struct timeval" size="#FIO_INPUT_POINTS_BYTES" />
```
2.3.3.5.2 Element: `<Set>`

The `<Set>` element is used to initialize a variable. The content of a variable is undefined until it is initialized. A variable must be defined before it is referenced. When a variable is referenced, it must be preceded by a `$`, to indicate a reference to a defined variable.

Valid attributes for the `<Set>` element are:

- **var**: REQUIRED – This attribute indicates the variable to initialize. The variable must first the created using `<Define>` and the type of variable must match the context in which the variable is initialized or an error will occur and the VSE will terminate.

- **value**: optional – This attribute specifies the value to set the variable to. The type of the `value` must be consistent with the type of the variable. Either `value` or `file` must be specified, but not both. Not all variable types support the `value` attribute.

- **file**: optional – This attribute specifies a human-readable file from which the data set in variable will be input from. Either `value` or `file` must be specified, but not both. Not all variable types support the `file` attribute.

- **operation**: optional – This attribute specifies the type of operation to be performed by the set. If absent, the variable is simply set to the indicated `value` or `file`. Not all variable types support all operations.

- **member**: optional – This attribute is utilized when the variable being initialized is a structure. In this case, each structure member must be initialized separately.

- **description**: optional – This attribute is used to add descriptive text to a `<Initialize>` element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```
<Set var="$ii" value="%10" />
<Set var="$timeval" member="tv_sec" value="%100" />
<Set var="$ls_plus" file="@ls_plus_set" />
```
2.3.3.6 APIVSXML Flow Control Elements

This section describes APIVSXML elements that are used to control the flow of execution.

2.3.3.6.1 Element: <Sleep>

The <Sleep> element is used to delay processing for the indicated number of seconds.

Valid attributes for the <Sleep> element are:

- **time**: REQUIRED – This attribute indicates the number of seconds to sleep the execution of the APIVSXML program.
- **description**: optional – This attribute is used to add descriptive text to a <Sleep> element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<Sleep time="%10"/>
```

2.3.3.6.2 Element: <Abort>

The <Abort> element is used to terminate processing of a <TestSuite>. The current <TestSuite> will be marked in the output with the **status** that is specified.

Valid attributes for the <Abort> element are:

- **description**: optional – This attribute is used to add descriptive text to a <Abort> element. This description is output into the log file, by the VSE, at the appropriate time.
- **status**: required – This attribute is used to print a special tag to the output log file to indicate PASS or FAIL, if desired.

Example:

```xml
<Abort status="FAIL"/>
```
2.3.3.7 APIVSXML Supporting Block Elements

This section describes APIVSXML elements that are used for APIVSXML block processing. Several APIVSXML statements are used to enclose a block of `<Statement>` elements. These statements allow for the set up and tear down of information and processing associated with a block. This processing is the equivalent functionality to that of C++ constructors and destructors.

The APIVSXML `<Statement>` elements that define a block are: `<TestSuite>`, `<TestCase>`, and `<Call>`.

2.3.3.7.1 Element: `<SetUp>`

The `<SetUp>` element is used to define a set up processing sequence of statements.

Valid attributes for the `<SetUp>` element are:

- **name**: REQUIRED – This attribute is a string that is referenced by a `setUp` attribute.
- **description**: optional – This attribute is used to add descriptive text to a `setUp` sequence. This description is output into the log file, by the VSE, at the appropriate time.

The `<SetUp>` element then may contain an unbounded set of `<Statement>` elements. These `<Statement>` elements are executed sequentially until either an `<Abort>` element is encountered or all of the statements have been executed.

Example:

```xml
<SetUp name="setUpii" >
  <Initialize var="$ii" value="%10" />
</SetUp>
```

2.3.3.7.2 Element: `<TearDown>`

The `<TearDown>` element is used to define a tear down processing sequence of statements.

Valid attributes for the `<TearDown>` element are:

- **name**: REQUIRED – This attribute is a string that is referenced by a `tearDown` attribute.
- **description**: optional – This attribute is used to add descriptive text to a `tearDown` sequence. This description is output into the log file, by the VSE, at the appropriate time.

The `<TearDown>` element then may contain an unbounded set of `<Statement>` elements. These `<Statement>` elements are executed sequentially until either an `<Abort>` element is encountered or all of the statements have been executed.

Example:

```xml
<TearDown name="tearDownSequence" >
  <Function name="fio_deregister" />
</TearDown>
```
2.3.3.8 APIVSXML Logging Elements

This section describes APIVSXML elements that are used to support logging and the validation process.

2.3.3.8.1 Element: <Print>

The <Print> element is used to print information to the log file. <Print> is used to print a string of text to the output file.

Valid attributes for the <Print> element are:

- **description**: optional – This attribute is used to add descriptive text to a print sequence. This description is output into the log file, by the VSE, at the start of the print.

- **level**: optional – This attribute allows for the control of at what run level a print occurs. When **level** is not present, summary level is assumed.

Example:

```
<Print>This text is output to the log file</Print>
<Print description="This is the description" level="always" >*** FAILURE ***</Print>
```

2.3.3.8.2 Element: <Dump>

The <Dump> element is used to capture information to a human-readable file. <Dump> only occurs during a “capture” mode run of the VSE, unless the attribute **force** is specified.

Valid attributes for the <Dump> element are:

- **description**: optional – This attribute is used to add descriptive text to a print sequence. This description is output into the log file, by the VSE, at the start of the print.

- **level**: optional – This attribute allows for the control of at what run level a print occurs. When **level** is not present, summary level is assumed.

- **file**: REQUIRED – This attribute specifies the name of the file for the <Dump>.

- **dump**: REQUIRED – This attribute specifies the type of <Dump> to perform.

- **sequence**: optional – For VD dumps, an optional sequence number may be appended to the file name. This allows <Dump> to occur in a loop.

- **frame**: optional – For CMDMSG dumps, a frame must be specified which indicates with command message frame to dump.

- **force**: optional – This attribute may be used to force the <Dump> to occur, regardless of the “capture” mode.

Example:

```
<Dump file="@VD_dump" dump="VD" />
<Dump description="This is the description" force="#TRUE" file="@CMDMSG_42" frame="%42" />
```
2.3.3.8.3 Element: `<Load>`

The `<Load>` element is used to load human-readable file information into the VSE. This is mostly used for testing the VSE itself.

Valid attributes for the `<Load>` element are:

- **description**: optional – This attribute is used to add descriptive text to a print sequence. This description is output into the log file, by the VSE, at the start of the print.
- **file**: REQUIRED – This attribute specifies the name of the file for the `<Dump>`.
- **load**: REQUIRED – This attribute specifies the type of `<Load>` to perform.
- **frame**: optional – For CMDMSG loads, a frame must be specified which indicates with command message frame to load.

Example:

```
<Load file="@VD_dump" load="VD" />
<Load description="This is the description" file="@CMDMSG_42" frame="%42" />
```

2.3.3.8.4 Element: `<Format>`

The `<Format>` element is used print variables.

Valid attributes for the `<Format>` element are:

- **description**: optional – This attribute is used to add descriptive text to a print sequence. This description is output into the log file, by the VSE, at the start of the print.
- **level**: optional – This attribute allows for the control of at what run level a format occurs. When level is not present, summary level is assumed.
- **var**: optional – This attribute specifies a specific variable to be output. If omitted, all active variables are output.

Example:

```
<Format var="$ii" />
<Format description="Print all active variables" />
```
3. APIVSXML Quick Reference Guide

This section contains a quick APIVSXML reference guide for all APIVSXML elements and attributes; to have one place to look.

```
<TestSuite name="" setUp="" tearDown="" description="" > ... </TestSuite>

<TestCase name="" setUp="" tearDown="" description="" > ... </TestCase>

<Call ref="" description="" setUp="" tearDown="" />

<Subroutine name="" description="" > ... </Subroutine>

<Function name="" description="" return="" errno=""
  p1="" p2="" p3="" p4="" p5="" p6="" />

<If expression="" description="" > ... </If>

<Then description="" > ... </Then>

<Else description="" > ... </Else>

<While expression="" description="" > ... </While>

<FPUIInput file="" description="" />

<FIOResponse file="" frame="" description="" />

<Aux flip="" description="" />

<Signal signal="" action="" ref="" description="" />

<SignalHandler name="" description="" > ... </SignalHandler>

<Define var="" type="" description="" />

<Set var="" member="" description="" value="" file="" operation="" />

<Sleep time="" description="" />

<Abort status="" description="" />

<SetUp name="" description="" > ... </SetUp>

<TearDown name="" description="" > ... </TearDown>

<Print description="" level="" > STRING </Print>

<Dump dump="" file="" sequence="" frame="" description="" level="" />

<Load load="" file="" description="" frame="" />

<Format var="" description="" />
```
4. APIVSXML Usage Examples

For listings of APIVSXML examples please look at the APIVSXML files found in the etc directory of the VSE code base.