

Agilent E2977A System Validation Package 2.1

## **Test API Reference**



**Agilent Technologies**

## Important Notice

### Copyright

© 2001, 2002 Agilent Technologies. All rights reserved.

No part of this manual may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Agilent Technologies Inc. as governed by United States and international copyright laws.

Author: Anja Schauer, t3 medien GmbH

### Notice

The material contained in this document is subject to change without notice. Agilent Technologies makes 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. Agilent Technologies 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.

ATTENTION: Use of the Software is subject to the Agilent Software License Terms set forth below. Using the Software indicates your acceptance of these license terms. If you do not accept these license terms, you may return the Software for a full refund. If the Software is bundled with another product, you may return the entire unused product for a full refund.

### Agilent Software License Terms

The following License Terms govern your use of the accompanying Software unless you have a separate signed agreement with Agilent.

**License Grant.** Agilent grants you a license to use one copy of the Software. "Use" means storing, loading, installing, executing or displaying the Software. You may not modify the Software or disable any licensing or control features of the Software. If the Software is licensed for "concurrent use", you may not allow more than the maximum number of authorized users to Use the Software concurrently.

**Ownership.** The Software is owned and copyrighted by Agilent or its third party suppliers. Your license confers no title to, or ownership in, the Software and is not a sale of any rights in the Software. Agilent's third party suppliers may protect their rights in the event of any violation of these License Terms.

**Copies and Adaptations.** You may only make copies or adaptations of the Software for archival purposes or when copying or adaptation is an essential step in the authorized Use of the Software. You must reproduce all copyright notices in the original Software on all copies or adaptations. You may not copy the Software onto any public network.

**No Disassembly or Decryption.** You may not disassemble or decompile the Software unless Agilent's prior written consent is obtained. In some jurisdictions, Agilent's consent may not be required for limited disassembly or decompilation. Upon request, you will provide Agilent with reasonably detailed information regarding any disassembly or decompilation. You may not decrypt the Software unless decryption is a necessary part of the operation of the Software.

**Transfer.** Your license will automatically terminate upon any transfer of the Software. Upon transfer, you must deliver the Software, including any copies and related documentation, to the transferee. The transferee must accept these License Terms as a condition to the transfer.

**Termination.** Agilent may terminate your license upon notice for failure to comply with any of these License Terms. Upon termination, you must immediately destroy the Software, together with all copies, adaptations and merged portions in any form.

**Export Requirements.** You may not export or re-export the Software or any copy or adaptation in violation of any applicable laws or regulations.

**U.S. Government Restricted Rights.** The Software and any accompanying documentation have been developed entirely at private expense. They are delivered and licensed as "commercial computer software" as defined in DFARS 252.227-7013 (Oct 1988), DFARS 252.211-7015 (May 1991) or DFARS 252.227-7014 (Jun 1995), as a "commercial item" as defined in FAR 2.101(a), or as "Restricted computer software" as defined in FAR 52.227-19 (Jun 1987)(or any equivalent agency regulation or contract clause), whichever is applicable. You have only those rights provided for such Software and any accompanying documentation by the applicable FAR or DFARS clause or the Agilent standard software agreement for the product involved.

# Contents

<b>Introduction</b>	<b>7</b>
<b>Structure of the Test API Reference</b>	<b>8</b>
<b>Basic Information for Using the Test API</b>	<b>9</b>
Platform-Relevant Information	9
Conventions	10
Exception/Error Handling	10
<b>Classes of the Test API Library</b>	<b>11</b>
<b>SVPClass Class</b>	<b>14</b>
Check	18
CommandLineInit	18
CommandLineUsage	19
CreateObject	19
Default	20
FileLoad	20
FileSave	21
FileSaveAs	21
GetCardList	22
GetNewLog	22
GetSelectionObject	23
GetTestList	23
GetTotalDuration	24
InsertObject	24
OfflineMode	25
RemoveObject	25
Run	26
StaticReport	26
Stop	27
SVPClass, Constructor	27
SVPClass, Destructor	27
<b>SVPScenario Class</b>	<b>28</b>
GetTotalDuration	31

SVPTestBase Class	32
GetTotalDuration	35
SVPTestCard Class	36
AllocateBuffer	40
BandwidthSet	40
CardType	41
CardTypeFromString	41
CheckPPR	42
ConfigScan	43
CPUTargetSetup	44
Default	45
GetAddress	46
GetLocation	47
GetRule	47
GetRuleCount	48
GetRuleDescription	48
GetSystemID	49
MasterReadSetup	50
MasterWRCSetup	51
MasterWriteSetup	52
New	53
ObserverTestSetup	54
Operator ==	54
Ping	55
ResetFactoryDefault	55
Run	56
SetRule	56
StaticReport	57
Stop	58
SVPPCICard Class	59
SVPPCIXCard Class	62
SVPBase Class	65
Check	68
CheckProp	69
ClearLog	70
Default	70
GetChildList	71
GetErr	71

GetID	72
GetLog	72
GetNameByObjectType	73
GetNewLog	74
GetObjectTypeByName	74
GetObjectType	75
GetProp	75
Init	76
InsertObject	77
Log	78
Name	78
PrepareRun	78
RemoveObject	79
Run	80
SetID	80
StaticReport	81
Status	81
Stop	82
UpdateStatus	82
SVPThread Class	83
Run	85
Stop	85
SVPCPUTest Class	86
SVPFSTest Class	89
SVPPool Class	92
SVPList Class	95
Count	96
DiscardList	96
GetObjectIDList	97
GetObjectType	98
Operator [] (int index)	98
Operator [] (const BString & strID)	98
SVPList, Constructor	99
SVPList, Destructor	99
SVPCardList Class	100

<b>Classes of the Service Library</b>	<b>103</b>
BErr	103
BString	104
BAddress	104
BRandomData	104
BLog	105
GetOStream	106
BLocation	106
<b>Enumeration Definitions</b>	<b>107</b>
EAddressSpace	107
ECardType	108
EState	108
ESVPObjectType	109
<b>Setup File Reference</b>	<b>111</b>
Scenario and Test Parameter	112
Testcard Parameters	113
Testcard and Location Information	113
Card Features Settings	114
Master Settings (for PCI Testcards)	115
Target Settings (for PCI Testcards)	117
Requester Settings (for PCI-X Testcards)	118
Completer Settings (for PCI-X Testcards)	119
Protocol Checker (Rule Masking)	121
<b>Overall Example Programs</b>	<b>123</b>
Simple Command Line Executable	124
Custom Test Function	126



# Introduction

The Test API library is a C++ library that provides all functionality of the System Validation Package 2.1. The System Validation Package 2.1 is based on the application interfaces (C-API) of Agilent PCI and PCI-X testcards.

The Test API library provides a number of tests that can be run immediately or be configured for individual test needs.

This library uses an object-oriented, class-based approach. The SVP Test API Reference describes all classes and their public members that are recommended for direct use.

# Structure of the Test API Reference

The SVP Test API Reference is divided into the following sections:

- **Classes of the Test API Library**  
Describes the classes of the SVPBase, SVPList and the SVPPropBase packages and all their public members that are recommended for direct use.  
Every class description starts with an overview of the class members in logical order, followed by the full description of the characteristic members in alphabetical order.
- **Classes of the Service Library**  
Describes the classes of the Service library and all their public members that are recommended for direct use.  
In addition to the Test API library, the Service library `servlib` provides a number of classes for general purposes, such as error handling.
- **Enumeration Definitions**  
Describes all defined enumerations that are used in the methods of Test API library and Service library classes.
- **Setup File Reference**  
Describes all properties, along with their names and value ranges that are needed when you modify the SVP settings files.
- **Overall Example Programs**  
Shows a simple command line executable program and a custom test function.



# Basic Information for Using the Test API

You can find basic information on using the SVP Test API library in the following sections:

## Platform-Relevant Information

**Platform Independence** Portability of the Test API software is ensured by avoiding platform-dependent code. Most parts are implemented using plain ANSI C++, which can be compiled by any standard C++ compiler.

Systems that do not have a stable operating system or no C/C++ compiler can be tested using an external host and the Front Side Interface (FSI).

**Platform-Dependent Features** Communication with E2920 series PCI testcards uses the E2920 series C-API. Communication with PCI-X testcards uses the PCI-X C-API. Both contain the necessary drivers to communicate with the testcards.

PCI C-API and PCI-X C-API are available in binary form for a number of operating systems, and as compilable source code for other systems.

The platform and operating system determine, which drivers are necessary for internal communication with the testcard and for the memory.

Drivers for Windows NT are available and shipped with the product, other operating systems may need to be ported and compiled by the user.

## Conventions

In the libraries, the following conventions are used for naming classes, methods, enumerations, variables and constants.

**Naming of Classes** Classes are written with lowercase and uppercase letters. The prefixes of the classes indicate to which library they belong:

- Classes of the Test API library begin with `SVPxxx`.

**Example:** `SVPBase`

- Classes of the Service library begin with `Bxxx`.

**Example:** `BErr`

**Naming of Methods** Methods are written with lowercase and uppercase letters and indicate the action.

**Example:** `RemoveObject`

**Naming of Enumerations** Enumerations are written with uppercase and lowercase letters. Each enumeration name begins with `Exxx`.

**Example:** `ECardType`

**Naming of Variables** Variables are written with uppercase and lowercase letters, but the prefix is always lowercase.

**Example:** `strID`

**Naming of Constants** Constants are written with uppercase letters.

**Example:** `S_WAIT`

## Exception/Error Handling

The Test API uses the C++ mechanism for handling exceptions. All methods of the various objects will handle some C++ exceptions internally and throw `BErr` error objects in case of unrecoverable errors.

**Example:**

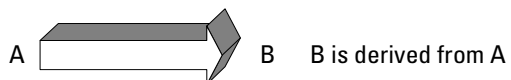
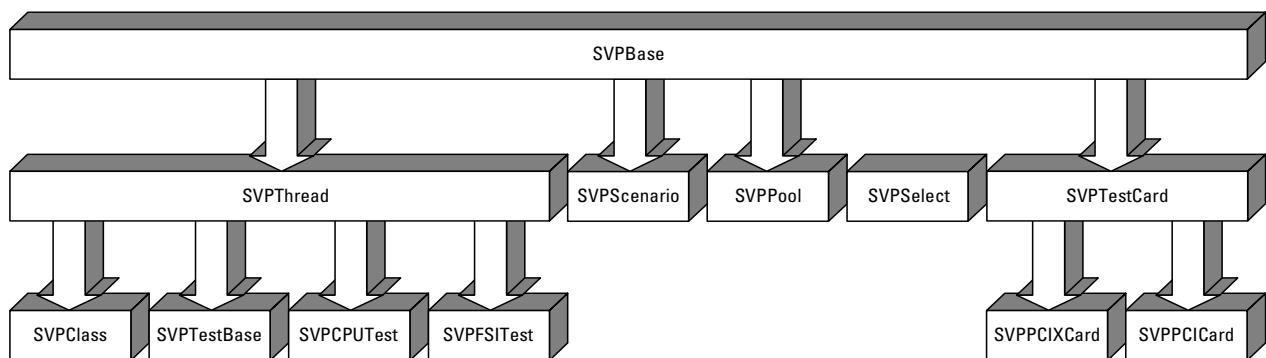
When trying to open a file that does not exist, a method would throw a `BErr::FILENOTFOUND` error along with a descriptive error message. The `BErr` object can be used to analyze the error, or may simply be printed as an error message to the console.

A user program should therefore encapsulate all calls to Test API functions and class methods in `try` blocks; see “*Overall Example Programs*” on page 123.

# Classes of the Test API Library

The Test API Library includes three independent packages of classes: the SVPBase package, the SVPList package and the SVPPropBase package.

**SVPBase Package Overview** The following figure shows the class architecture of the SVPBase package:



**SVPBase Package Classes** SVPBase Class is a base class that implements all fundamental features of an object and provides a standard means for accessing objects. In this context, objects are the whole SVP application (SVPClass), a scenario (SVPScenario), a test (SVPTestBase) or a testcard (SVPTestCard and derived from it SVPPCICard and SVPPCIXCard).

From this class, the following classes are derived:

- SVPThread Class

Provides basic threading functionality for running several tests in parallel. From this class, the following classes are derived:

- SVPClass Class

Represents the whole SVP application.

- SVPTestBase Class

Provides all testing functionality of the SVP application.

- SVPCPUtest Class

Is used to perform tests with CPU interaction.

- SVPFSITest Class

Is used to perform tests with FSI interaction.

- SVPScenario Class

Provides the features of a scenario. A scenario is used to put several tests together for running these tests in parallel.

- SVPPOOL Class

Is used to manage the available tests and testcards.

- SVPTestCard Class

From this class, the following classes are derived:

- SVPPCICard Class

Covers all E2926/7/8 and E2940 testcards.

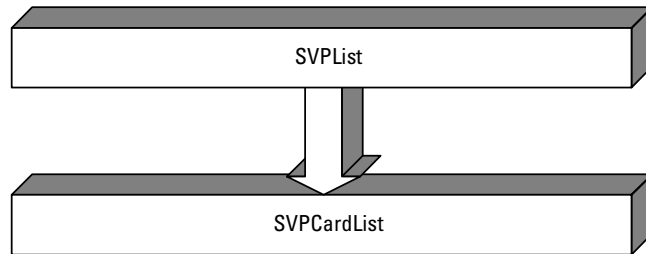
- SVPPCIXCard Class

Covers all E2922/9 testcards.

**SVPSelect Class** The Test API library includes also the SVPSelect class, which is independent from all other classes in the SVPBase package. The SVPSelect class is normally not needed in the user program. It is mainly used by the GUI.

Each class of this package has its own SVPPropBase package.

**SVPList Package Overview** The following figure shows the class architecture of the SVPList package:



**SVPList Package Classes** The SVPList package provides the following classes for administration of SVP objects (scenarios, tests and testcards):

- **SVPList Class**

This class provides a container class for SVP objects. It provides methods:

- for handling the deletion and creation from settings files
- for walking through the lists

- **SVPCardList Class**

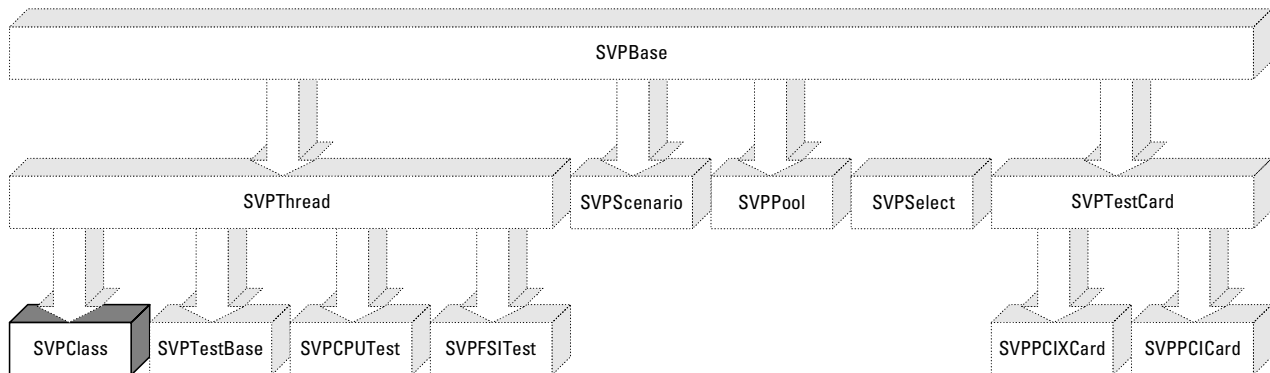
This class is derived from SVPList. This class provides methods for scanning and listing available testcards.

**SVPPropBase Class** The SVPPropBase class provides functionality for handling properties and settings.

This class includes the CPropEl class that allows you to assign different types of values (for example, integer, string or boolean values) to the same class.

# SVPClass Class

**Description** The SVPClass class provides the main interface to the user programs.

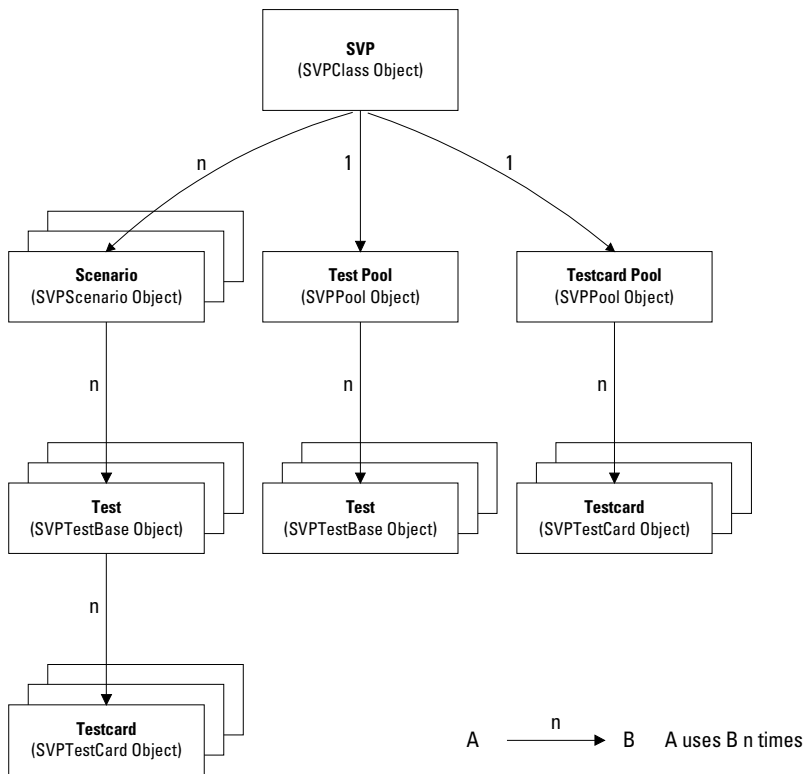


SVPClass provides methods for the following purposes:

- Accessing objects for direct modification
- Initialization and file handling
- Test execution
- Reporting

- Creation and deletion of objects

The objects can be scenarios, tests and testcards. The hierarchy of the objects is shown below.



#### Characteristic Members

The following tables list all public members of the SVPClass class that are recommended for direct use.

Constructor	
public	SVPClass (void)

Destructor	
public virtual ~	SVPClass (void)

Public Methods	
SVPBase *	CreateObject (const BString & name)
SVPSelect	GetSelectionObject (SVPBase * theObject)

Initialization and File Handling	
const char *	CommandLineUsage (void) const
void	CommandLineInit (int argc, char * const argv[])
void	FileLoad (const char * pLoadFileName, bool isOfflineMode=B_TRUE)
void	FileSave (void)
void	FileSaveAs (const char * savefile)
virtual bool	OfflineMode (bool goOffline)

Other Public Methods Overriding SVPBase	
void	Run (void)
void	Stop (void)
void	Check (BErr & errLog)
void	Default (void)
virtual void	RemoveObject (const ESVPObject objectType, const BString & strID)
virtual void	InsertObject (SVPBase * pSvpObject)
virtual ostream &	GetNewLog (ostream & o)

Access to Lists of Enumerations/Direct Object Access	
virtual const SVPList *	GetTestList ( )
virtual const SVPList *	GetCardList ( ) const

Reporting	
virtual ostream &	StaticReport (ostream & o)
time_t	GetTotalDuration (void) const

### Inherited Members from SVPThread

The following table lists the members inherited from the *SVPThread* class:

Public Methods	
virtual void	Run (void)
virtual void	Stop (void)

For detailed description of the inherited members, refer to “*SVPThread Class*” on page 83.



**Inherited Members from SVPBase** The following tables list the members inherited from the SVPBase class:

Various Public Methods	
virtual void	PrepareRun (SVPPropSection * propsection)
virtual EState	Status (void) const
virtual EState	UpdateStatus (void)

Access to Lists of Enumerations/Direct Object Access	
virtual const SVPList *	GetChildList (void)

Static Member Methods	
static const ESVPObjType	GetObjectTypeByName (const BString & name)
static const BString	GetNameByObjectType (const ESVPObjType objectType)

Identification Routines	
virtual const ESVPObjType	GetObjectType (void) const
virtual const BString	Name (void) const
virtual void	SetID (const BString & theID)
virtual const BString &	GetID (void) const

Property Publication	
virtual CPropEl &	GetProp (const BString & key) const
virtual bool	CheckProp (const BString & key) const

Reporting	
virtual BLog &	Log (void)
virtual ostream &	GetLog (ostream & o)
virtual void	ClearLog (void)
virtual const BErr &	GetErr (void) const

For detailed description of the inherited members, refer to “SVPBase Class” on page 65.

## Check

**Include Files** `#include <svpclass.h>`

**Call** `public virtual void Check( BErr & errLog );`

**Description** Checks if the SVP application has been correctly initialized. This method is called by `Run()` as well.

**Return Value** No return value.

**Output Parameters** **errLog** Reference to a BErr object. The object contains:

- a list of all error messages that occurred – if the run is not permitted.
- `BErr::OK` – if the run is permitted.

**See Also** *“Run” on page 26*

## CommandLineInit

**Include Files** `#include <svpclass.h>`

**Call** `void * CommandLineInit(  
                                int     argc,  
                                char    * const argv[] );`

**Description** Passes command line arguments to the SVPClass object (as passed from `main()`).

**Return Value** No return value.

**Input Parameters** **argc** Number of arguments on command line.

**argv** Array of arguments.

**See Also** *“CommandLineUsage” on page 19*

## CommandLineUsage

**Include Files** `#include <svpclass.h>`

**Call** `const char * CommandLineUsage( void ) const;`

**Description** Returns the string that contains the information for using the command line.

**Return Value** Pointer to the char array that contains the text string.

**See Also** *“CommandLineInit” on page 18*

## CreateObject

**Include Files** `#include <svpclass.h>`

**Call** `SVPBase * CreateObject( const ESVPObjectType objectType );`

**Description** Creates a new object specified by its type. Use this function to create:

- scenarios (SVPScenario objects)
- tests (SVPTestBase objects)
- testcards (SVPPCICard or SVPPCIXCard objects)

CreateObject inserts the objects into the corresponding pool. To add tests to a scenario or testcards to a test, use InsertObject.

**Return Value** The return values are:

- Pointer to the new object.
- NULL – if object creation was not successful.

**Input Parameters** **objectType** Type of the object that is created. See *“ESVPObjectType” on page 109*.

**See Also** *“InsertObject” on page 24*

**Example**

```
// create a new test object
SVPTestBase * newTest = (SVPTestBase *) svp.CreateObject(T_TEST);
SVPAAssert(newTest);
```

## Default

**Include Files** `#include <svpclass.h>`

**Call** `virtual void Default( void );`

**Description** Sets this object to default values. That means:

- One empty scenario is configured.
- All available test functions are defined as test objects.
- In online mode, the available testcards found in the system are defined as objects of the type testcard. In offline mode, no testcards will be configured.

**Return Value** No return value.

**See Also** –

## FileLoad

**Include Files** `#include <svpclass.h>`

**Call** `void FileLoad(  
          const char * pLoadFileName,  
          bool isOfflineMode = B_FALSE );`

**Description** Loads SVP settings from the specific file.

**Return Value** No return value.

**Input Parameters** **pLoadFileName** Name of the file from which the settings are to be loaded.

**isOfflineMode** Forces the offline mode, regardless of whether or not it is specified in the settings file.

**See Also** *“FileSave” on page 21*  
*“FileSaveAs” on page 21*

## FileSave

**Include Files** `#include <svpclass.h>`

**Call** `void FileSave( void );`

**Description** Saves the settings to the file from which the settings were loaded with the FileLoad call.

**Return Value** No return value.

**See Also** *"FileLoad" on page 20*  
*"FileSaveAs" on page 21*

## FileSaveAs

**Include Files** `#include <svpclass.h>`

**Call** `void FileSaveAs( const char * savefile );`

**Description** Saves the SVP settings to the specified file.

**Return Value** No return value.

**Input Parameters** **savefile** Name of the file to which the settings are to be saved.

**See Also** *"FileSave" on page 21*  
*"FileLoad" on page 20*

## GetCardList

**Include Files** `#include <svpclass.h>`  
`#include <svplist.h>`

**Call** `virtual const SVPList * GetCardList() const;`

**Description** Returns a pointer to the testcard pool list. This method is used for list enumeration.

**Return Value** Pointer to the testcard pool list.

**See Also** *“InsertObject” on page 24*

**Example** `// gets the first testcard in the list`  
`SVPBase * theCard = (*(svp.GetCardList()))[0];`

## GetNewLog

**Include Files** `#include <svpclass.h>`

**Call** `virtual ostream & GetNewLog( ostream & o );`

**Description** Writes to the selected ostream object whatever has been added to the report since the last GetLog or GetNewLog call.

**Return Value** Reference to the ostream object input parameter.

**Input Parameters** **o** ostream object. This object can be, for example, cout, cerr, or an ofstream file. To get the pointer to the ostream object, use the GetOStream call.

**See Also** *“GetLog” on page 72*  
*“Log” on page 78*  
*“GetOStream” on page 106*

**Example** `while (svp.Status() == SVPBase::S_RUN || svp.Status() ==  
SVPBase::S_WAIT)`  
`{`  
`// wait for ten seconds`  
`Sleep (10000);`  
`// additional test code`  
`...`  
`// writes latest additions to the test report to stdout`  
`svp.GetNewLog(cout);`  
`}`

## GetSelectionObject

**Include Files** `#include <svpclass.h>`

**Call** `SVPSelect GetSelectionObject( SVPBase * theObject );`

**Description** Creates a SVPSelect object for selecting this object's children.

**NOTE** You must remove the created object when you have finished selecting the object's children.

**Return Value** The return values are:

- Pointer to the SVPSelection object.
- NULL – if creation was not successful.

**Input Parameters** **theObject** Pointer to the object for which the selection takes place.

**See Also** *"GetID" on page 72*  
*"SetID" on page 80*

## GetTestList

**Include Files** `#include <svpclass.h>`  
`#include <svplist.h>`

**Call** `virtual const SVPList * GetTestList();`

**Description** Returns a pointer to the test pool list. This method is used for list enumeration.

**Return Value** Pointer to the test pool list.

**See Also** *"InsertObject" on page 24*

**Example** `// gets the first test in the list`  
`SVPBase * theTest = (*(svp.GetTestList()))[0];`

## GetTotalDuration

**Include Files** `#include <svpclass.h>`

**Call** `time_t GetTotalDuration( void ) const;`

**Description** Returns the total duration of the whole testing.

**Return Value** Total duration as `time_t` value. For more information about `time_t`, see the C-standard library.

**See Also** –

## InsertObject

**Include Files** `#include <svpclass.h>`

**Call** `virtual void InsertObject( SVPBase * pSvpObject );`

**Description** Inserts the specified object into this object's pool. Use this function to insert:

- a test into the test pool
- a testcard into the testcard pool
- a scenario into the SVP object's child list

**Return Value** No return value.

**Input Parameters** **pSvpObject** Pointer to an object. The object can be:

- a test (defined by `SVPTestBase`)  
To get the pointer to a specific test, use the `CreateObject` call.
- a testcard (defined by `SVPPICard` or `SVPPICIXCard`)  
To get the pointer to a specific testcard, use the `GetCardList` call.

**See Also** *“CreateObject” on page 19*  
*“GetCardList” on page 22*

**Example**

```
// assign one testcard to the test (first testcard that was found)
SVPBase * theCard = (*(svp.GetCardList()))[0];
newTest->InsertObject(theCard);
```



## OfflineMode

**Include Files** `#include <svpclass.h>`

**Call** `void OfflineMode(bool goOffline);`

**Description** Switches between offline and online mode. The offline mode allows you to configure tests without actually having a testcard plugged into the system under test. The online mode allows you to run tests and to generate a test report.

**Return Value** No return value.

**Input Parameters** **goOffline** The values are:

- `B_TRUE` – offline mode is switched on.
- `B_FALSE` – offline mode is switched off.

**See Also** *“Run” on page 26*  
*“GetLog” on page 72*

## RemoveObject

**Include Files** `#include <svpclass.h>`  
`#include <bstring.h>`

**Call** `virtual void RemoveObject(  
                                    const ESVPObjectType objectType,  
                                    const BString              & strID );`

**Description** Removes an object completely (from pool and all references). The object is specified by its string ID and type.

To get the pointer of the string ID, use the `GetID` call. To set the string ID, use the `SetID` call.

**Return Value** No return value.

**Input Parameters** **objectType** Type of the object that is removed; see *“ESVPObjectType” on page 109*.

**strID** String that identifies the object.

**See Also** *“GetID” on page 72*  
*“SetID” on page 80*

## Run

**Include Files** `#include <svpclass.h>`

**Call** `virtual void Run( void );`

**Description** Runs the SVP testing within a thread.

**Return Value** No return value.

**See Also** *“Stop” on page 27*

## StaticReport

**Include Files** `#include <svpclass.h>`

**Call** `virtual ostream & StaticReport( ostream & o );`

**Description** Generates the static report for this object. The static report is generated after the setup has been finished and before any actual testing takes place.

This report gives information about:

- Software  
Version, build number or DLL versions
- System configuration  
Operating system, number of processors, number of busses
- Testcard configuration  
Number of cards, types of cards, firmware version(s), possibly serial numbers to uniquely identify cards
- Test setup and the number of scheduled tests

**Return Value** Reference to the `ostream` object input parameter.

**Input Parameters** **o** `ostream` object. The `ostream` object can be, for example, `cout`, `cerr`, or an `ofstream` file.

**See Also** –

## Stop

**Include Files** `#include <svpclass.h>`

**Call** `virtual void Stop( void );`

**Description** Requests a thread and stops it.

**Return Value** No return value.

**See Also** *“Run” on page 26*

## SVPClass, Constructor

**Include Files** `#include <svpclass.h>`

**Call** `public SVPClass( void );`

**Description** Default constructor.

Initializes this object and allocates a sufficient memory space.

**Return Value** No return value.

**See Also** *“SVPClass, Destructor” on page 27*

## SVPClass, Destructor

**Include Files** `#include <svpclass.h>`

**Call** `public virtual ~SVPClass( void );`

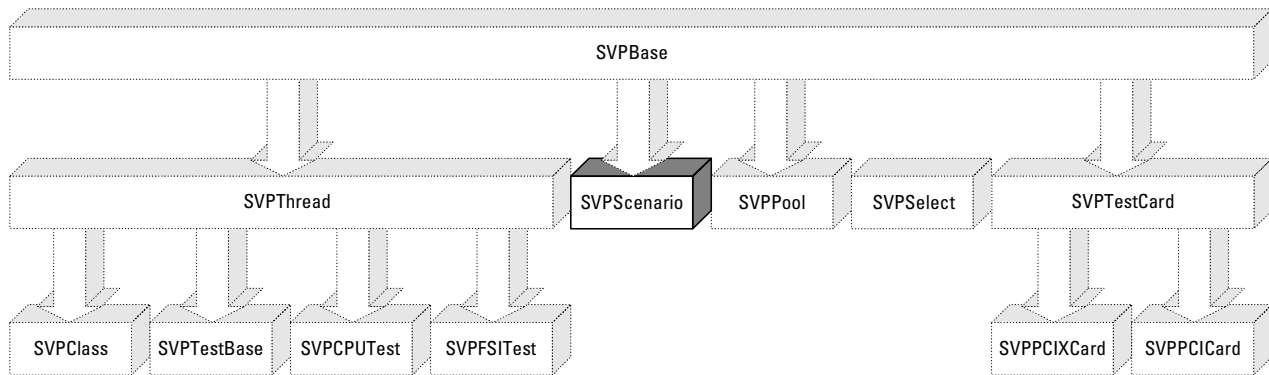
**Description** Frees the memory space that has been allocated for this object.

**Return Value** No return value.

**See Also** *“SVPClass, Constructor” on page 27*

# SVPSenario Class

**Description** The SVPSenario class provides the features of a scenario.



**Characteristic Members** The following tables list all public members of the SVPSenario class that are recommended for direct use.

## Constructor

No public constructor available. Use SVPClass::CreateObject instead.

## Destructor

No public destructor available. Use SVPClass::RemoveObject instead.

## Public Method

Public Method	
time_t	GetTotalDuration (void) const

**Inherited Members** The following tables list the members inherited from the SVPBase class.

Various Public Methods	
virtual void	Init (void)
virtual void	Run (void)
virtual void	Stop (void)
virtual void	Check (const BErr & errLog)
virtual void	PrepareRun (SVPPropSection * propsection)
virtual EState	Status (void) const
virtual EState	UpdateStatus (void)
virtual void	RemoveObject (const ESVPObjType objectType, const BString & strID)
virtual void	InsertObject (SVPBase & pSvpObject)

Access to Lists of Enumerations/Direct Object Access	
virtual const SVPList *	GetChildList (void)

Static Member Methods	
static const ESVPObjType	GetObjectTypeByName (const BString & name)
static const BString	GetNameByObjectType (const ESVPObjType objectType)

Identification Routines	
virtual const ESVPObjType	GetObjectType (void) const
virtual const BString	Name (void) const
virtual void	SetID (const BString & theID)
virtual const BString &	GetID (void) const

Default Settings	
virtual void	Default (void)

Property Publication	
virtual CPropEl &	GetProp (const BString & key) const
virtual bool	CheckProp (const BString & key) const

Reporting	
virtual BLog &	Log (void)
virtual ostream &	GetLog (ostream & o)
virtual ostream &	GetNewLog (ostream & o)
virtual void	ClearLog (void)
virtual ostream &	StaticReport (ostream & o)
virtual const BErr &	GetErr (void) const

For detailed description of the inherited members, refer to “*SVPBase Class*” on page 65.

## GetTotalDuration

**Include Files** `#include <svps scena.h>`

**Call** `time_t GetTotalDuration( void ) const;`

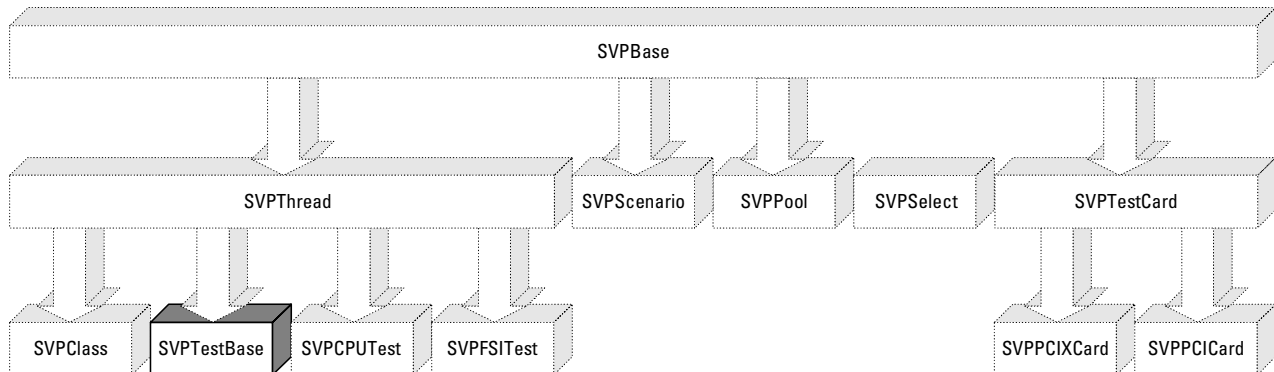
**Description** Returns the total duration of testing for the specified scenario. The total duration of a scenario is the maximum period of time that results from the sum of start delay and test duration.

**Return Value** Total duration as `time_t` value. For reference to `time_t`, see the C-standard library.

**See Also** –

# SVPTTestBase Class

**Description** The SVPTTestBase class provides all testing functionality of the SVP application. The most useful methods are overrides from SVPBase and SVPTThread, and provide the same functionality.



**Characteristic Members** The following tables list all public members of the SVPTTestBase class that are recommended for direct use.

Constructor	
No public constructor available. Use SVPClass::CreateObject instead.	

Destructor	
No public destructor available. Use SVPClass::RemoveObject instead.	

Public Method	
time_t	GetTotalDuration (void) const

**Inherited Members from SVPTThread** The following table lists the members inherited from the SVPTThread class.

Public Methods	
virtual void	Run (void)
virtual void	Stop (void)

For detailed description of these inherited members, refer to “SVPTThread Class” on page 83.



**Inherited Members from SVPBase** The following tables list the members inherited from the SVPBase class.

Constructor	
No public constructor available. Use SVPClass::CreateObject instead.	

Destructor	
No public destructor available. Use SVPClass::RemoveObject instead.	

Various Public Methods	
virtual void	Init (void)
virtual void	Check (const BErr & errLog)
virtual void	PrepareRun (SVPPropSection * propsection)
virtual EState	Status (void) const
virtual EState	UpdateStatus (void)
virtual void	RemoveObject (const ESVPObjectType objectType, const BString & strID)
virtual void	InsertObject (SVPBase & pSvpObject)

Access to Lists of Enumerations/Direct Object Access	
virtual const SVPList *	GetChildList (void)

Identification Routines	
virtual const ESVPObjectType	GetObjectType (void) const
virtual const BString	Name (void) const
virtual void	SetID (const BString & theID)
virtual const BString &	GetID (void) const

Default Settings	
virtual void	Default (void)

Property Publication	
virtual CPropEl &	GetProp (const BString & key) const
virtual bool	CheckProp (const BString & key) const

Reporting	
virtual BLog &	Log (void)
virtual ostream &	GetLog (ostream & o)
virtual ostream &	GetNewLog (ostream & o)
virtual void	ClearLog (void)
virtual ostream &	StaticReport (ostream & o)
virtual const BErr &	GetErr (void) const

Static Member Methods	
static const ESVPObjType	GetObjectTypeByName (const BString & name)
static const BString	GetNameByObjectType (const ESVPObjType objectType)

For detailed description of these inherited members, refer to “*SVPBase Class*” on page 65.

## GetTotalDuration

**Include Files** `#include <testbase.h>`

**Call** `time_t GetTotalDuration( void ) const;`

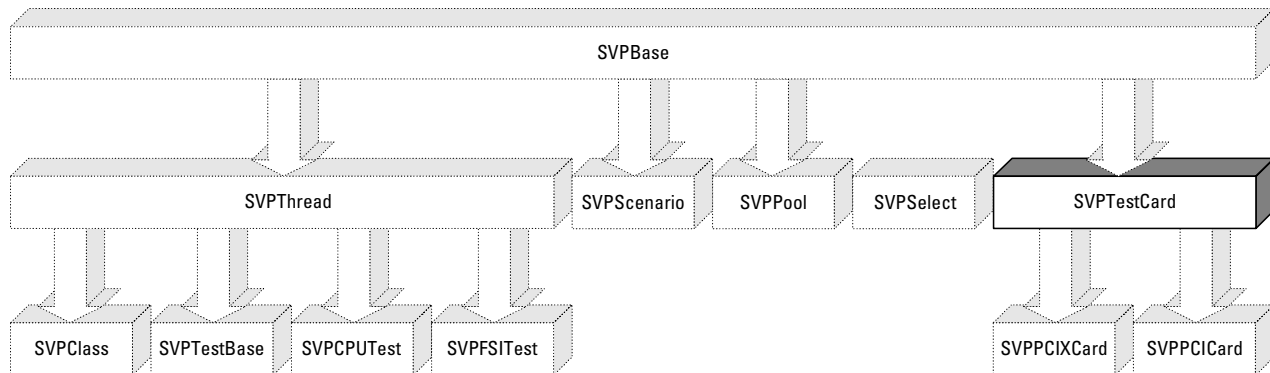
**Description** Returns the total duration of the specified test. The total duration of a test is the sum of start delay and test duration.

**Return Value** Total duration as `time_t` value. For reference to `time_t`, see the C-standard library.

**See Also** –

# SVPTeStCard Class

**Description** The SVPTeStCard class includes the members of the SVPBase class.



**Characteristic Members** The following tables list all public members of the SVPTeStCard class that are recommended for direct use.

Defined Operator	
public virtual bool	operator == (const SVPBase & other) const

Constructor
No public constructor available. Use SVPClass::CreateObject instead.

Destructor
No public destructor available. Use SVPClass::RemoveObject instead.

Default Settings	
virtual void	Default (void)
virtual void	ResetFactoryDefault (void)
virtual const BString	Ping (void)

Reporting	
virtual ostream &	StaticReport (ostream & o)

Public Methods Overriding Base Class's Methods	
void	Run (void)
void	Stop (void)

Various Public Methods	
virtual bool	CheckPPR (DWORD fmFlags, BString * pprReport)
virtual void	BandwidthSet (double theBandwidth)
virtual ECardType	CardType (void) const
virtual const BLocation &	GetLocation (void)
virtual DWORD	GetSystemID (void)
virtual bool	GetAddress (BAddress::EAddressSpace space, int isPrefetch, BAddress & address)
virtual void	AllocateBuffer (BAddress & Address

Testcard Setup Methods	
virtual void	MasterWRCSetup (const BAddress & address)
virtual void	MasterReadSetup (const BAddress & address)
virtual void	MasterWriteSetup (const BAddress & address)
virtual const BString	ConfigScan (void)
virtual void	CPUTargetSetup (const BAddress::EAddressSpace space, int prefetch, LPRUN_FCT runFct)
virtual void	ObserverTestSetup (void)

Protocol Rules	
virtual DWORD	GetRuleCount (void) const
virtual const SSvpPropRule	GetRule (DWORD dwIndex) const
virtual const BString	GetRuleDescription (DWORD dwIndex) const
virtual void	SetRule (DWORD dwIndex, bool bState)

Static Members	
static ECardType	CardTypeFromString (const BString & cardStr)
static SVPTestCard *	New (SVPBase * parent, const BString & modelStr="E2928A")

**Inherited Members** The following tables list the members inherited from the SVPBase class.

Various Public Methods	
virtual void	Init (void)
virtual void	Run (void)
virtual void	Stop (void)
virtual void	Check (const BErr & errLog)
virtual void	PrepareRun (SVPPropSection * propsection)
virtual EState	Status (void) const
virtual EState	UpdateStatus (void)
virtual void	RemoveObject (const ESVPObjectType objectType, const BString & strID)
virtual void	InsertObject (SVPBase & pSvpObject)

Access to Lists of Enumerations/Direct Object Access	
virtual const SVPList *	GetChildList (void)

Static Member Methods	
static const ESVPObjectType	GetObjectTypeByName (const BString & name)
static const BString	GetNameByObjectType (const ESVPObjectType objectType)

Identification Routines	
virtual const ESVPObjectType	GetObjectType (void) const
virtual const BString	Name (void) const
virtual void	SetID (const BString & theID)
virtual const BString &	GetID (void) const

Property Publication	
virtual CPropEl &	GetProp (const BString & key) const
virtual bool	CheckProp (const BString & key) const

Reporting	
virtual BLog &	Log (void)
virtual ostream &	GetLog (ostream & o)
virtual ostream &	GetNewLog (ostream & o)
virtual void	ClearLog (void)
virtual const BErr &	GetErr (void) const

For detailed description of the inherited members, refer to “*SVPBase Class*” on page 65.

## AllocateBuffer

**Include Files** `#include <testcard.h>`

**Call** `virtual void AllocateBuffer( BAddress & address );`

**Description** Allocates buffer on the system under test for this testcard object.

**Return Value** No return value.

**Output Parameters** **address** Reference to the BAddress object.

**See Also** –

**Example** `DWORD size = 1024; // must be power of two  
const BAddress & address = theCard->AllocateBuffer(size);`

## BandwidthSet

**Include Files** `#include <testcard.h>`

**Call** `virtual void BandwidthSet (double theBandwidth);`

**Description** Sets the bus bandwidth that is to be occupied by the test in percent. The test tries to occupy the bus with this bandwidth. Please be aware that the selected bus bandwidth cannot always be achieved.

**Return Value** No return value.

**Input Parameters** **theBandwidth** Desired value for the bandwidth. Valid values are in the range of 0.0 ... 1.0, where 1.0 means 100%.

**See Also** –



## CardType

**Include Files** `#include <testcard.h>`

**Call** `virtual ECardType CardType( void ) const;`

**Description** Returns the type of the testcard.

**Return Value** Type of the card; see “*ECardType*” on page 108.

**See Also** “*CardTypeFromString*” on page 41

## CardTypeFromString

**Include Files** `#include <bstring.h>`  
`#include <testcard.h>`

**Call** `virtual ECardType CardTypeFromString( const BString & cardStr );`

**Description** Returns the type of the testcard, according to the model number.

**Return Value** Type of the card; see “*ECardType*” on page 108.

**Input Parameters** **cardStr** String that contains the model number. Valid strings are:

- “E2925B”
- “E2926A”
- “E2926B”
- “E2927A”
- “E2928A”
- “E2940A”
- “E2929A”
- “E2922A”

**See Also** “*New*” on page 53  
“*CardType*” on page 41

## CheckPPR

**Include Files** `#include <testcard.h>`

**Call** `virtual bool CheckPPR(  
          DWORD      mFlags;  
          BString     * pprReport);`

**Description** Checks the PPR software and generates a report.

**Return Value** The return values are:

- B\_TRUE – if no errors occurred.
- B\_FALSE – if an error occurred.

**Input Parameters** **mFlags** Flag mask that indicates which report is to be generated. Relevant bits are:

- fm\_USE\_MASTER – if the testcard is using the master.
- fm\_USE\_TARGET – if the testcard is using the target.

**Output Parameters** **pprReport** Report as a string.

**See Also** –

**Example**

```
BString reportStr;  
DWORD mFlags = fm_USE_MASTER | fm_USE_TARGET;  
  
// report message  
if (testCard->CheckPPR(mFlags, &reportStr) == B_TRUE)  
{  
    cout << "Check OK:" << (const char *)reportStr << endl;  
}  
else  
{  
    cerr << "Check failed:" << (const char *)reportStr << endl;  
}
```

## ConfigScan

**Include Files**    `#include <bstring.h>`  
                  `#include <testcard.h>`

**Call**            `virtual const BString ConfigScan( void );`

**Description**    Scans the whole configuration space of the PCI-X bus system.

**Return Value**    The results of the configuration space scan as a string.

**See Also**        *“MasterReadSetup” on page 50*  
                  *“MasterWRCSetup” on page 51*  
                  *“MasterWriteSetup” on page 52*  
                  *“CPUTargetSetup” on page 44*  
                  *“ObserverTestSetup” on page 54*

## CPUTargetSetup

**Include Files** `#include <testcard.h>`

**Call**

```
virtual void CPUTargetSetup(
    const BAddress::EAddressSpace & space,
    int prefetch,
    LPRUN_FCT runFct;
```

**Description** Sets up the target and the CPU test. You need this method also to set up the FSI test.

**Return Value** No return value.

**Input Parameters** **space** Address space. See “*EAddressSpace*” on page 107.

**prefetch** Valid values are:

- 0 Use decoder that has prefetch “on”.
- 1 Use decoder that has prefetch “off” (don’t care).

**runFct** Pointer to the run function for internal testing. The run function is a part of the `STestFct` structure. See the example.

**See Also** “*GetAddress*” on page 46  
 “*MasterReadSetup*” on page 50  
 “*MasterWRCSetup*” on page 51  
 “*MasterWriteSetup*” on page 52  
 “*ConfigScan*” on page 43  
 “*ObserverTestSetup*” on page 54

**Example** // The following struct contains the actual test definitions including the run function

```

STestFct g_customFct =
{
    "Custom Memory Read", // descriptive name
    "custommemread", // short name
    custommemread_init, // init function
    custommemread_run, // run function
    0, // no stop function
    1, 1, // one card only
    "Reads from main memory and does other custom testing", // long
    description
    tpf_FULL_ADDRESS | tpf_BANDWIDTH // test flags
};

```

For a detailed example of a custom test function, refer to “*Custom Test Function*” on page 126.

## Default

**Include Files** #include <testcard.h>

**Call** virtual void Default( void );

**Description** Sets all properties of this testcard object to default values.

**See Also** “*GetProp*” on page 75  
 “*ResetFactoryDefault*” on page 55  
 “*Ping*” on page 55

**Example** SVPTestCard card;  
 ...  
 // Sets all properties of the testcard object to default values  
 card.Default();

## GetAddress

**Include Files** `#include <testcard.h>`

**Call** `virtual int GetAddress(  
                                 BAddress::EAddressSpace  space,  
                                 bool                      isPrefetch,  
                                 BAddress                  & address);`

**Description** Gets the base address of the desired address space. For the address space, you can select between configuration space, memory space and I/O space. The obtained base address can be used as input parameter in the following methods:

- MasterWRCSetup
- MasterReadSetup
- MasterWriteSetup

**Return Value** The return values are:

0 ... 5	BAR 0 ... 5
-1	no BAR is found

**Input Parameters** **space** Address space. See “*EAddressSpace*” on page 107.

**isPrefetch** The valid values are:

-1	don't care
0	no
1	yes

**Output Parameters** **address** Reference to the BAddress object that contains the base address.

**See Also** “*MasterReadSetup*” on page 50  
 “*MasterWRCSetup*” on page 51  
 “*MasterWriteSetup*” on page 52

**Example**

```
BAddress::EAddressSpace space = BAddress::SPACE_MEM;

int prefetch = -1;

BAddress address;

// get address of second card to program first one
secondCard->GetAddress(space, prefetch, address);

// setup test for first card
firstCard->MasterWRCSetup(address);
```

## GetLocation

**Include Files** `#include <testcard.h>`

**Call** `virtual const BLocation & GetLocation( void );`

**Description** Gets the location of this testcard object in the system under test. For external connections, FSI is needed.

**Return Value** BLocation object with bus number, device number and function number of this testcard object.

**See Also** –

## GetRule

**Include Files** `#include <testcard.h>`

**Call** `virtual const SSvpProtRule GetRule( DWORD dwIndex ) const;`

**Description** Gets the rule with the specified index.

**Return Value** SsvpProtRule object with name and state.

**Input Parameters** **dwIndex** Index of the rule.

**See Also** *“GetRuleDescription” on page 48*  
*“GetRuleCount” on page 48*  
*“SetRule” on page 56*

## GetRuleCount

**Include Files** `#include <testcard.h>`

**Call** `virtual DWORD GetRuleCount( void ) const;`

**Description** Returns the number of rules that are observed by this testcard object.

**Return Value** Number of rules as DWORD.

**See Also** *“GetRule” on page 47*  
*“GetRuleDescription” on page 48*  
*“SetRule” on page 56*

## GetRuleDescription

**Include Files** `#include <testcard.h>`

**Call** `virtual const BString GetRuleDescription( DWORD dwIndex ) const;`

**Description** Gets a detailed description of the rule with the specified index.

**Return Value** String that contains the description of the rule.

**Input Parameters** **dwIndex** Index of the rule.

**See Also** *“GetRule” on page 47*  
*“GetRuleCount” on page 48*  
*“SetRule” on page 56*



## GetSystemID

**Include Files** `#include <testcard.h>`

**Call** `virtual DWORD GetSystemID( void );`

**Description** Gets the unique system ID.

**Return Value** System ID. The values are:

0 Unknown

1 Controlling host

A unique random number All other systems that are controlled by FSI

**See Also** –

## MasterReadSetup

**Include Files** `#include <testcard.h>`

**Call** `virtual void MasterReadSetup( const BAddress & address );`

**Description** Sets up a *Read* test.

When you want to access this testcard from another testcard or from the CPU, you can use the `GetAddress` call to define its address and to get the reference to the defined `BAddress` object.

**Return Value** No return value.

**Input Parameters** **address** `BAddress` object that defines the address, the size of the address space and the kind of address space (mem or I/O). See *"GetAddress" on page 46*.

**See Also** *"MasterWRCSetup" on page 51*  
*"MasterWriteSetup" on page 52*  
*"ConfigScan" on page 43*  
*"CPUTargetSetup" on page 44*  
*"ObserverTestSetup" on page 54*  
*"GetAddress" on page 46*

**Example**

```
BAddress address (0xb8000, 0x0, BAddress::SPACE_MEM, 1024);  
// part of video memory  
  
theCard->MasterReadSetup(address);  
// read from the specified address
```

## MasterWRCSetup

**Include Files** `#include <testcard.h>`

**Call** `public virtual void MasterWRCSetup( const BAddress & address );`

**Description** Sets up a *Write*, *Read* and *Compare* test.

When you want to access this testcard from another testcard or from the CPU, you can use the `GetAddress` call to define its address and to get the reference to the defined `BAddress` object.

**Return Value** No return value.

**Input Parameters** **address** `BAddress` object that defines the address, the size of the address space and the kind of address space (mem or I/O). See *"GetAddress" on page 46*.

**See Also** *"MasterReadSetup" on page 50*  
*"MasterWriteSetup" on page 52*  
*"ConfigScan" on page 43*  
*"CPUTargetSetup" on page 44*  
*"ObserverTestSetup" on page 54*  
*"GetAddress" on page 46*

**Example**

```
BAddress address (0xb8000, 0x0, BAddress::SPACE_MEM, 1024);  
// part of video memory  
  
theCard->MasterWRCSetup(address);  
// write/read/compare from the specified address
```

## MasterWriteSetup

**Include Files** `#include <testcard.h>`

**Call** `public virtual void MasterWriteSetup( const BAddress & address );`

**Description** Sets up a *Write* test.

When you want to access this testcard from another testcard or from the CPU, you can use the `GetAddress` call to define its address and to get the reference to the defined `BAddress` object.

**Return Value** No return value.

**Input Parameters** **address** `BAddress` object that defines the address, the size of the address space and the kind of address space (mem or I/O). See *"GetAddress" on page 46*.

**See Also** *"MasterReadSetup" on page 50*  
*"MasterWRCSetup" on page 51*  
*"ConfigScan" on page 43*  
*"CPUTargetSetup" on page 44*  
*"ObserverTestSetup" on page 54*  
*"GetAddress" on page 46*

**Example**

```
BAddress address (0xb8000, 0x0, BAddress::SPACE_MEM, 1024);  
// part of video memory  
  
theCard->MasterWriteSetup(address);  
// write to the specified address
```

## New

**Include Files** `#include <testcard.h>`

**Call** `static SVPTestCard * New(  
                    SVPBase * parent,  
                    const BString & modelStr = "E2928A");`

**Description** Defines a new testcard object, which is specified by the parent object and the model number.

New is only used in **offline** mode to create a testcard of a specific type. In **online** mode, New is not needed: testcards are created by the SVPClass object.

**NOTE** To insert the testcard into the pool, you **MUST** use `SVPClass::InsertObject`.

**Return Value** Pointer to the new testcard object.

**Input Parameters** **parent** Pointer to the parent object.

**modelStr** String that contains the model number. Valid strings are:

- "E2925B"
- "E2926A"
- "E2926B"
- "E2927A"
- "E2928A"
- "E2940A"
- "E2929A"
- "E2922A"

**See Also** *"CardTypeFromString" on page 41*

## ObserverTestSetup

**Include Files** `#include <testcard.h>`

**Call** `virtual void ObserverTestSetup( void );`

**Description** Sets up this testcard object for an observer-only test. That means that master and target are not used.

**Return Value** No return value.

**See Also** *“MasterReadSetup” on page 50*  
*“MasterWRCSetup” on page 51*  
*“MasterWriteSetup” on page 52*  
*“ConfigScan” on page 43*  
*“CPUTargetSetup” on page 44*

## Operator ==

**Include Files** `#include <testcard.h>`

**Call** `virtual bool operator == ( const SVPBase & other ) const;`

**Description** Checks if two testcard objects refer to the same testcard. For example, this method can be used to check if one testcard object is connected to two ports.

**Return Value** The return values are:

- B\_TRUE – if the testcard objects refer to the same testcard.
- B\_FALSE – if the testcard objects refer to different testcards.

**Input Parameter** **other** Reference to the second testcard.

**See Also** –

**Example**

```
if (firstCard == secondCard)
{
    // if firstCard and secondCard are identical (connected by
    // different ports, maybe), peer-to-peer testing is not possible
}
else
{
    myPeertoPeer(firstCard, secondCard);
}
```

## Ping

**Include Files** `#include <bstring.h>`  
`#include <testcard.h>`

**Call** `virtual void BString Ping( void );`

**Description** Checks the connection to this testcard object. The green and the red LEDs on the card flash. An error message will report any connection errors.

**Return Value** The method returns:

- An empty string – if the testcard connection is correct.
- An error message – if connection errors occurred.

**See Also** *“Default” on page 45*  
*“ResetFactoryDefault” on page 55*

## ResetFactoryDefault

**Include Files** `#include <testcard.h>`

**Call** `virtual void ResetFactoryDefault( void );`

**Description** Resets testcard settings for this testcard object to default values. For the changes to become effective, you must reboot the testcard.

**Return Value** No return value.

**See Also** *“Default” on page 45*  
*“Ping” on page 55*

## Run

**Include Files** `#include <testcard.h>`

**Call** `public virtual void Run( void );`

**Description** Runs this object. The object can be the entire SVP application, a scenario or a test. This method also checks if a test or a testcard has been locked. If the object has been locked, an error is thrown.

Before you use this function, call `PrepareRun` to verify the settings.

**NOTE** For running single tests and scenarios, use `UpdateStatus` to query the current object status.

**Return Value** No return value.

**See Also** *“Stop” on page 58*

## SetRule

**Include Files** `#include <testcard.h>`

**Call** `virtual void SetRule(  
          DWORD    dwIndex,  
          bool     bState );`

**Description** Enables or disables the rule with the specified index.

**Return Value** No return value.

**Input Parameters** **dwIndex** Index of the rule.

**bState** The values are:

- `B_TRUE` – the rule is disabled.
- `B_FALSE` – the rule is enabled.

**See Also** *“GetRuleDescription” on page 48*  
*“GetRule” on page 47*  
*“GetRuleCount” on page 48*



## StaticReport

**Include Files** `#include <testcard.h>`

**Call** `public virtual ostream StaticReport( ostream & o );`

**Description** Generates the static report for this object. This method overrides the SVPBase class StaticReport method.

The static report is generated after the setup has been finished and before any actual testing takes place.

This report gives information about:

- Software  
Version, build number or DLL versions
- System configuration  
Operating system, number of processors, number of busses
- Testcard configuration  
Number of cards, types of cards, firmware version(s), possibly serial numbers to uniquely identify cards
- Test setup and the number of scheduled tests

**Return Value** Reference to the ostream object input parameter.

**Input Parameters**

- o ostream object. The ostream object can be, for example, cout, cerr, or an ofstream file.

**See Also** –

## Stop

**Include Files** `#include <testcard.h>`

**Call** `virtual void Stop( void );`

**Description** Stops this running object. The object can be the entire SVP application, a scenario or a test.

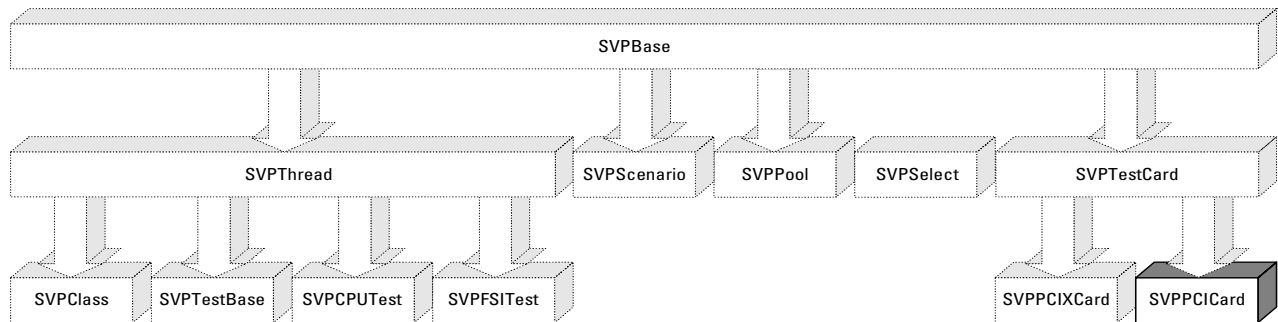
**Return Value** No return value.

**See Also** *“Run” on page 56*

# SVPPCICard Class

**Description** The SVPPCICard class is used for PCI series cards and provides all functionality of the SVPTestCard class. The following testcards are supported:

- E2925B
- E2926A/B
- E2927A
- E2928A
- E2940A CompactPCI



**Characteristic Members** This class contains no additional members that can be directly called by the user.

**Inherited Members** The SVPPCICard class includes the members of the SVPTTestCard class. The following tables list the members inherited from the SVPTTestCard class.

Defined Operator	
public virtual bool	operator == (const SVPPCICard & other) const

Constructor
No public constructor available. Use SVPPCICard::CreateObject instead.

Destructor
No public destructor available. Use SVPPCICard::RemoveObject instead.

Default Settings	
virtual void	Default (void)
virtual void	ResetFactoryDefault (void)
virtual const BString	Ping (void)

Reporting	
virtual ostream &	StaticReport (ostream & o)

Public Methods Overriding Base Class's Methods	
void	Run (void)
void	Stop (void)

Various Public Methods	
virtual bool	CheckPPR (DWORD fmFlags, BString * pprReport)
virtual void	BandwidthSet (double theBandwidth)
virtual ECardType	CardType (void) const
virtual const BLocation &	GetLocation (void)
virtual DWORD	GetSystemID (void)
virtual bool	GetAddress (BAddress::EAddressSpace space, int isPrefetch, BAddress & address)
virtual void	AllocateBuffer (BAddress & Address

Testcard Setup Methods	
virtual void	MasterWRCSetup (const BAddress & address)
virtual void	MasterReadSetup (const BAddress & address)
virtual void	MasterWriteSetup (const BAddress & address)
virtual const BString	ConfigScan (void)
virtual void	CPUTargetSetup (const BAddress::EAddressSpace space, int prefetch, LPRUN_FCT runFct)
virtual void	ObserverTestSetup (void)

Protocol Rules	
virtual DWORD	GetRuleCount (void) const
virtual const SSvpPropRule	GetRule (DWORD dwIndex) const
virtual const BString	GetRuleDescription (DWORD dwIndex) const
virtual void	SetRule (DWORD dwIndex, bool bState)

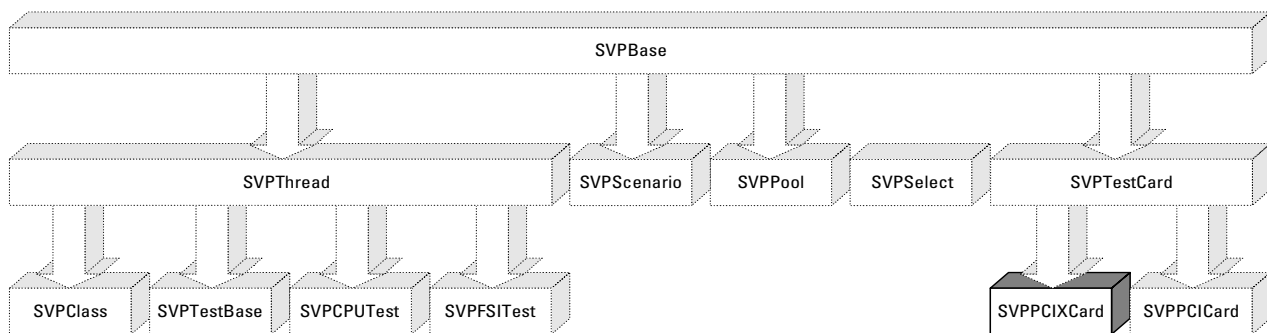
Static Members	
static ECardType	CardTypeFromString (const BString & cardStr)
static SVPTestCard *	New (SVPBase * parent, const BString & modelStr="E2928A")

For detailed description of the inherited members, refer to “*SVPTestCard Class*” on page 36.

# SVPPCIXCard Class

**Description** The SVPPCIXCard class is used for PCI-X series cards and provides all functionality of the SVPTTestCard class. The following testcards are supported:

- E2929A
- E2922A



**Characteristic Members** This class contains no additional members that can be directly called by the user.

**Inherited Members** The SVPPCIXCard class includes the members of the SVPTTestCard class. The following tables list the members inherited from the SVPTTestCard class.

Defined Operator	
public virtual bool	operator == (const SVPBase & other) const

Constructor
No public constructor available. Use SVPClass::CreateObject instead.

Destructor
No public destructor available. Use SVPClass::RemoveObject instead.

Default Settings	
virtual void	Default (void)
virtual void	ResetFactoryDefault (void)
virtual const BString	Ping (void)

Reporting	
virtual ostream &	StaticReport (ostream & o)

Public Methods Overriding Base Class's Methods	
void	Run (void)
void	Stop (void)

Various Public Methods	
virtual bool	CheckPPR (DWORD fmFlags, BString * pprReport)
virtual void	BandwidthSet (double theBandwidth)
virtual ECardType	CardType (void) const
virtual const BLocation &	GetLocation (void)
virtual DWORD	GetSystemID (void)
virtual bool	GetAddress (BAddress::EAddressSpace space, int isPrefetch, BAddress & address)
virtual void	AllocateBuffer (BAddress & Address

Testcard Setup Methods	
virtual void	MasterWRCSetup (const BAddress & address)
virtual void	MasterReadSetup (const BAddress & address)
virtual void	MasterWriteSetup (const BAddress & address)
virtual const BString	ConfigScan (void)
virtual void	CPUTargetSetup (const BAddress::EAddressSpace space, int prefetch, LPRUN_FCT runFct)
virtual void	ObserverTestSetup (void)

Protocol Rules	
virtual DWORD	GetRuleCount (void) const
virtual const SSvpPropRule	GetRule (DWORD dwIndex) const
virtual const BString	GetRuleDescription (DWORD dwIndex) const
virtual void	SetRule (DWORD dwIndex, bool bState)

Static Members	
static ECardType	CardTypeFromString (const BString & cardStr)
static SVPTestCard *	New (SVPBase * parent., const BString & modelStr="E2928A")

For detailed description of the inherited members, refer to “*SVPTestCard Class*” on page 36.

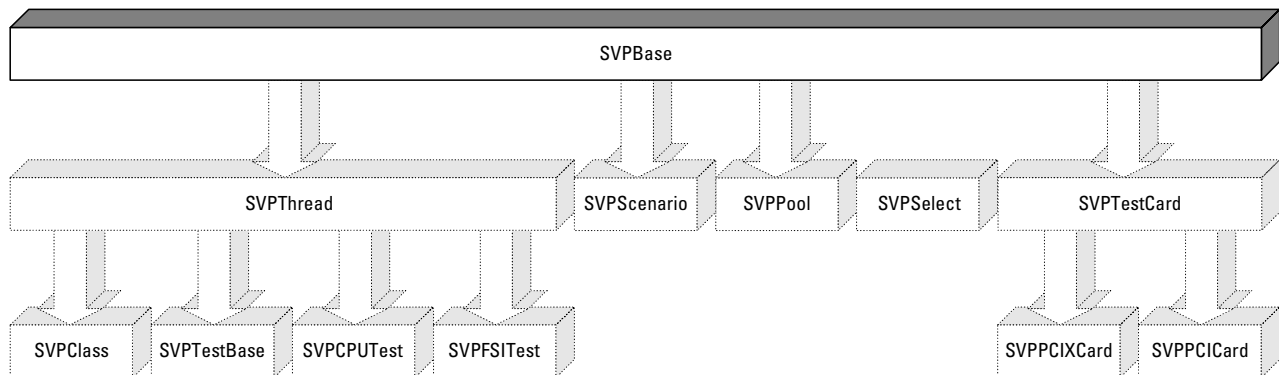


# SVPBase Class

**Description** SVPBase is the base class for all SVP objects. This class includes fundamental features and provides a standard means for accessing objects. Objects can be:

- the whole SVP application (SVPClass)
- a scenario (SVPSscenario)
- a test (SVPTTestBase)
- a testcard (SVPTTestCard)

**NOTE** SVPBase objects must not be created directly. SVPClass creates and deletes objects.



**Characteristic Members** The following tables list all public members of the SVPBase class that are recommended for direct use.

Constructor
No public constructor available. Use SVPClass::CreateObject instead.

Destructor
No public destructor available. Use SVPClass::RemoveObject instead.

Various Public Methods	
virtual void	Init (void)
virtual void	Run (void)
virtual void	Stop (void)
virtual void	Check (const BErr & errLog)
virtual void	PrepareRun (SVPPropSection * propsection)
virtual EState	Status (void) const
virtual EState	UpdateStatus (void)
virtual void	RemoveObject (const ESVPObjectType objectType, const BString & strID)
virtual void	InsertObject (SVPBase & pSvpObject)

Access to Lists of Enumerations/Direct Object Access	
virtual const SVPList *	GetChildList (void)

Static Member Methods	
static const ESVPObjectType	GetObjectTypeByName (const BString & name)
static const BString	GetNameByObjectType (const ESVPObjectType objectType)

Identification Routines	
virtual const ESVPObjectType	GetObjectType (void) const
virtual const BString	Name (void) const
virtual void	SetID (const BString & theID)
virtual const BString &	GetID (void) const

Default Settings	
virtual void	Default (void)

Property Publication	
virtual CPropEl &	GetProp (const BString & key) const
virtual bool	CheckProp (const BString & key) const

Reporting	
virtual BLog &	Log (void)
virtual ostream &	GetLog (ostream & o)
virtual ostream &	GetNewLog (ostream & o)
virtual void	ClearLog (void)
virtual ostream &	StaticReport (ostream & o)
virtual const BErr &	GetErr (void) const

## Check

**Include Files** `#include <svpbases.h>`

**Call** `public virtual void Check( BErr & errLog );`

**Description** Checks if an object has been correctly initialized. The object can be:

- the whole SVP application (SVPClass object)
- a scenario (SVPScenario object)
- a test (SVPTestBase object)
- a testcard (SVPPCICard or SVPPCIXCard object)

**Return Value** No return value.

**Output Parameters** **errLog** Reference to an BErr object that contains all error messages that occurred when checking the initialization of the regarding object. If initialization is correct, it contains BErr::OK.

**See Also** –

**Example**

```
BErr errLog;
theObject->Check(errLog);
if (errLog)
{
    cerr << "Error checking object: " << errLog << endl;
}
```

## CheckProp

**Include Files** `#include <bstring.h>`  
`#include <svpbases.h>`

**Call** `public virtual bool CheckProp( const BString & key ) const;`

**Description** Returns the status of a boolean property.

**Return Value** The return values are:

- `B_TRUE` – if the property has been activated.
- `B_FALSE` – if the property has not been activated.

**Input Parameters** **key** Specifies the boolean property to be checked, such as "use.master", which determines if the master of the testcard is used. For the available properties, see *"Setup File Reference" on page 111*.

**See Also** *"GetProp" on page 75*

**Example**

```
int prefetch;

if (svpTest->CheckProp("address.prefetch"))
{
    prefetch =
        (bool) (svpTest->GetProp(pAddressPrefetch)) == B_TRUE ? 1 : 0;
}
else
{
    prefetch = -1;
}
```

## ClearLog

**Include Files** `#include <svpbases.h>`

**Call** `virtual void ClearLog( void );`

**Description** Clears the log (test report) of the entire SVP application.

**See Also** *“GetLog” on page 72*  
*“Log” on page 78*  
*“GetNewLog” on page 74*

## Default

**Include Files** `#include <svpbases.h>`

**Call** `virtual void Default( void );`

**Description** Sets all properties of an object to default values. The object can be:

- the whole SVP application (defined by SVPClass)
- a scenario (defined by SVPScenario)
- a test (defined by SVPTestBase)
- a testcard (defined by SVPPCICard or SVPPCIXCard)

**See Also** *“GetProp” on page 75*

**Example** `SVPClass svp;`

`...`

`// Sets all properties of the SVP application object to default  
values`

`svp.Default();`

## GetChildList

**Include Files** `#include <svpbased.h>`  
`#include <svplist.h>`

**Call** `virtual const SVPList * GetChildList( void ) const;`

**Description** Returns a pointer to an object's child list.

**Return Value** The return values are:

- Pointer to the object's child list.
- NULL – if no pointer is available.

**See Also** *"RemoveObject" on page 79*

**Example**

```
SVPScenario * theSc;  
SVPTTestBase * theTest;  
  
// get pointer to first scenario  
theSc = (SVPScenario *) (*svp.GetChildList())[0];  
  
// get pointer to first test  
theTest = (SVPTTestBase *) svp.GetTestList()[0];  
  
// insert test into scenario  
theSc.InsertObject (theTest);
```

## GetErr

**Include Files** `#include <svpbased.h>`

**Call** `virtual const BErr & GetErr( void ) const;`

**Description** Returns the latest error that occurred during the SVP application.

**Return Value** Reference to the error object that contains the latest error.

**See Also** –

## GetID

**Include Files** `#include <bstring.h>`  
`#include <svpbased.h>`

**Call** `virtual const BString & GetID( void ) const;`

**Description** Returns the user-defined name of the regarding object.

**Return Value** Name of the object.

**See Also** *“SetID” on page 80*

## GetLog

**Include Files** `#include <svpbased.h>`

**Call** `virtual ostream & GetLog( ostream & o );`

**Description** Writes the log (test report) of this object to the specified ostream object.

**Return Value** Reference to the ostream object input parameter.

**Input Parameters** **o** ostream object. This object can be, for example, cout, cerr, or an ofstream file. To get the pointer to the ostream object, use the GetOStream call.

**See Also** *“Log” on page 78*  
*“GetNewLog” on page 74*  
*“GetOStream” on page 106*

**Example** `// print log (test report) to std out`  
`svp.GetLog(cout);`



## GetNameByObjectType

**Include Files** `#include <bstring.h>`  
`#include <svpbase.h>`

**Call** `static const BString GetNameByObjectType(  
const ESVPObjType objectType);`

**Description** Reads the name that identifies the object with the specified object type. For example, if you select the object type T\_TESTCARD, “Testcard” will be read.

**Return Value** Name of the object.

**Input Parameters** **objectType** Type of the object of which the name is read; see “*ESVPObjType*” on page 109.

**See Also** “*GetObjectTypeByName*” on page 74

**Example**

```
SVPBase * theObject;  
  
// assign valid object to theObject...  
// ...  
  
BString theString = SVPBase::GetNameByObjectType( theObject);  
cout << "Object is a " << theString << endl;
```

## GetNewLog

**Include Files** `#include <svpbases.h>`

**Call** `virtual ostream & GetNewLog( ostream & o );`

**Description** Writes an additional log (test report) of an object to the selected ostream object.

**Return Value** Reference to the ostream object input parameter.

**Input Parameters** **o** ostream object. This object can be, for example, cout, cerr, or an ofstream file. To get the pointer to the ostream object, use the GetOStream call.

**See Also** *“GetLog” on page 72*  
*“Log” on page 78*  
*“GetOStream” on page 106*

**Example**

```
while (svp.Status() == SVPBase::S_RUN || svp.Status() ==
SVPBase::S_WAIT)
{
    ...
    // report additions to test report to stdout
    svp.GetNewLog(cout);
}
```

## GetObjectTypeName

**Include Files** `#include <bstring.h>`  
`#include <svpbases.h>`

**Call** `static const ESVPObjType GetObjectTypeName(
const BString & name );`

**Description** Returns the object type for the selected name.

**Return Value** Object type; see *“ESVPObjType” on page 109*.

**Input Parameters** **name** Name of the object.

**See Also** *“GetNameByObjectType” on page 73*

## GetObjectType

**Include Files** `#include <svpbases.h>`

**Call** `virtual const ESVPObjectType GetObjectType( void );`

**Description** Returns the type of an object.

**Return Value** Object type; see “*ESVPObjectType*” on page 109.

**See Also** “*GetObjectTypeByName*” on page 74

## GetProp

**Include Files** `#include <bstring.h>`  
`#include <svpbases.h>`

**Call** `virtual CPropEl & GetProp( const BString & key );`

**Description** Used to access a property by its name. You can then set this property to the desired value. For properties and values, see “*Setup File Reference*” on page 111.

**Return Value** Reference to an CPropEl object. The class CPropEl is needed to assign different types of values (for example, integer, string or boolean) to the same class.

**Input Parameters** **key** String that specifies the property. For all available properties, see “*Setup File Reference*” on page 111.

**See Also** “*CheckProp*” on page 69

**Example**

```
// set the test duration to 120 seconds
newTest->GetProp("duration") = 120UL;
// set the address property of the test
newTest->GetProp("address") = testAddress;
```

## Init

**Include Files** `#include <svpbases.h>`

**Call** `public virtual void Init( void );`

**Description** Initializes an object. The object can be:

- the whole SVP application (SVPClass)
- a scenario (SVPScenario)
- a test (SVPTestBase)
- a testcard (SVPPCICard and SVPPCIXCard)

**Return Value** No return value.

**See Also** —

## InsertObject

**Include Files** `#include <svpbases.h>`

**Call** `virtual void InsertObject( SVPBase * pSvpObject );`

**Description** Inserts an object into an object's child list and increments its reference count. You can use this function to insert:

- a test into a scenario
- a testcard into a test

To insert a scenario into the SVP application object, use `CreateObject(T_SCENARIO)`.

**Return Value** No return value.

**Input Parameters** **pSvpObject** Pointer to an object. The object can be:

- a test (defined by `SVPTTestBase`)  
To get the pointer to a specific test, use the `SVPClass::GetTestList()` call.
- a testcard (defined by `SVPPICard` or `SVPPICIXCard`)  
To get the pointer to a specific testcard, use the `SVPClass::GetCardList()` call.

**See Also** *"CreateObject" on page 19*  
*"GetCardList" on page 22*

**Example**

```
// assign testcard to a new test (the first testcard that has been scanned is used)
SVPBase * theCard = (*(svp.GetCardList()))[0];
newTest->InsertObject(theCard);
```

## Log

**Include Files** `#include <svpbases.h>`

**Call** `virtual BLog & Log( void );`

**Description** Gets reference to `m_log`. `m_log` is a `BLog` object that records the test report. (Each `SVPBase` object owns a `BLog` object for reporting.) See *“BLog” on page 105*.

**Return Value** Reference to `m_log`.

**See Also** *“GetLog” on page 72*  
*“GetNewLog” on page 74*

## Name

**Include Files** `#include <bstring.h>`  
`#include <svpbases.h>`

**Call** `virtual const BString Name( void ) const;`

**Description** Reads the name of an object.

**Return Value** Name of the object.

**See Also** *“GetNameByObjectType” on page 73*

## PrepareRun

**Include Files** `#include <svpbases.h>`

**Call** `virtual void PrepareRun( void );`

**Description** Prepares this object for running. This method verifies the settings of the object that is to be run.

**Return Value** No return value.

**See Also** *“Run” on page 80*

## RemoveObject

**Include Files** `#include <bstring.h>`  
`#include <svpbases.h>`

**Call** `virtual void RemoveObject(  
                                const ESVPObjectType objectType,  
                                const BString              & strID );`

**Description** Removes a child from the object's child list. The child object is specified by its string ID and type.

You can query the string ID of an object with the `GetID` call. The `RemoveObject` call decrements the reference count of the object by one.

**Return Value** No return value.

**Input Parameters** **objectType** Type of the object that is removed; see “*ESVPObjectType*” on page 109.

**strID** String that identifies the object.

**See Also** “*GetChildList*” on page 71  
“*GetID*” on page 72  
“*SetID*” on page 80

**Example** `// remove second testcard from test:  
SVPTestCard * theCard = (*theTest->GetChildList())[1];  
theTest->RemoveObject(T_TESTCARD, theCard->GetID());`

## Run

**Include Files** `#include <svpbases.h>`

**Call** `public virtual void Run( void );`

**Description** Runs an object. The object can be the entire SVP application, a scenario and a test. This method also checks if a test or a testcard has been locked. If the object has been locked, an error is thrown.

Before you use this function, call `PrepareRun` to verify the settings.

**NOTE** To run single tests and scenarios, you must call `UpdateStatus` to query the current object status.

**Return Value** No return value.

**See Also** *"Stop" on page 82*  
*"PrepareRun" on page 78*

## SetID

**Include Files** `#include <bstring.h>`  
`#include <svpbases.h>`

**Call** `virtual void SetID( const BString & theID );`

**Description** Assigns a user-specific name to this object.

**Return Value** No return values.

**Input Parameters** **theID** User name of the object.

**See Also** *"GetID" on page 72*

**Example**

```
// assign a custom test function
newTest->SetTestFct(&g_customFct);
newTest->SetID("My custom test");
```



## StaticReport

**Include Files** `#include <svpbases.h>`

**Call** `virtual ostream & StaticReport( ostream & o );`

The static report is generated after the setup has been finished and before any actual testing takes place.

This report gives information about:

- Software  
Version, build number or DLL versions
- System configuration  
Operating system, number of processors, number of busses
- Testcard configuration  
Number of cards, types of cards, firmware version(s), possibly serial numbers to uniquely identify cards
- Test setup and the number of scheduled tests

**Return Value** Reference to the ostream object input parameter.

**Input Parameters** **o** ostream object. The ostream object can be, for example, cout, cerr, or an ofstream file.

**See Also** *“GetLog” on page 72*  
*“GetNewLog” on page 74*

## Status

**Include Files** `#include <svpbases.h>`

**Call** `virtual EState Status( void ) const;`

**Description** Queries status information of an object. For status information, see *“EState” on page 108*.

**Return Value** Status of the object.

**See Also** *“UpdateStatus” on page 82*

## Stop

**Include Files** `#include <svpbases.h>`

**Call** `virtual void Stop( void );`

**Description** Stops the running object. The object can be the entire SVP application, a scenario or a test.

**Return Value** No return value.

**See Also** *“Run” on page 80*

## UpdateStatus

**Include Files** `#include <svpbases.h>`

**Call** `virtual EState UpdateStatus( void );`

**Description** Returns the updated status of this object. For status information, see *“EState” on page 108*.

**Return Value** Updated status of the object.

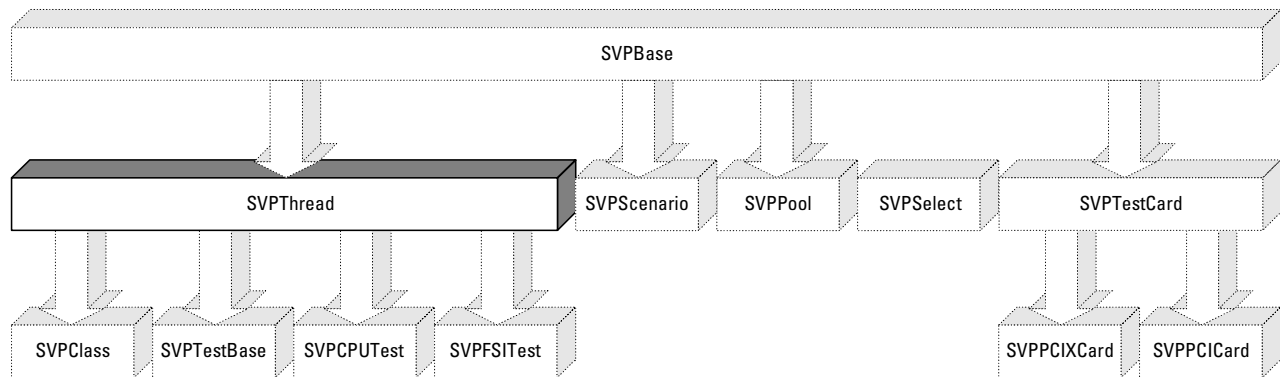
**See Also** *“Status” on page 81*

# SVPThread Class

**Description** The SVPThread class includes the SVPBase members. This class overrides some of the SVPBase methods to provide basic threading functionality for the following classes:

- SVPClass
- SVPCPUTest
- SVPTTestBase
- SVPFSITest

All members of the SVPThread class are platform-dependent.



**Characteristic Members** The following table lists all public members of the SVPThread class that are recommended for direct use.

Public Methods	
virtual void	Run (void)
virtual void	Stop (void)

**Inherited Members** The following tables list the members inherited from the SVPBase class.

Constructor
No public constructor available. Use SVPClass::CreateObject instead.

Destructor
No public destructor available. Use SVPClass::RemoveObject instead.

Various Public Methods	
virtual void	Init (void)
virtual void	Check (const BErr & errLog)
virtual void	PrepareRun (SVPPropSection * propsection)
virtual EState	Status (void) const
virtual EState	UpdateStatus (void)
virtual void	RemoveObject (const ESVPObjectType objectType, const BString & strID)
virtual void	InsertObject (SVPBase & pSvpObject)

Access to Lists of Enumerations/Direct Object Access	
virtual const SVPList *	GetChildList (void)

Identification Routines	
virtual const ESVPObjectType	GetObjectType (void) const
virtual const BString	Name (void) const
virtual void	SetID (const BString & theID)
virtual const BString &	GetID (void) const

Default Settings	
virtual void	Default (void)

Property Publication	
virtual CPropEl &	GetProp (const BString & key) const
virtual bool	CheckProp (const BString & key) const

Reporting	
virtual BLog &	Log (void)
virtual ostream &	GetLog (ostream & o)
virtual ostream &	GetNewLog (ostream & o)
virtual void	ClearLog (void)
virtual ostream &	StaticReport (ostream & o)
virtual const BErr &	GetErr (void) const

Static Member Methods	
static const ESVPObjType	GetObjectTypeByName (const BString & name)
static const BString	GetNameByObjectType (const ESVPObjType objectType)

For detailed description of the inherited members, refer to “*SVPBase Class*” on page 65.

## Run

**Include Files** `#include <svpthrd.h>`

**Call** `public virtual void Run( void );`

**Description** Executes the thread.

**Return Value** No return value.

**See Also** “*Stop*” on page 85

## Stop

**Include Files** `#include <svpthrd.h>`

**Call** `public virtual void Stop( void );`

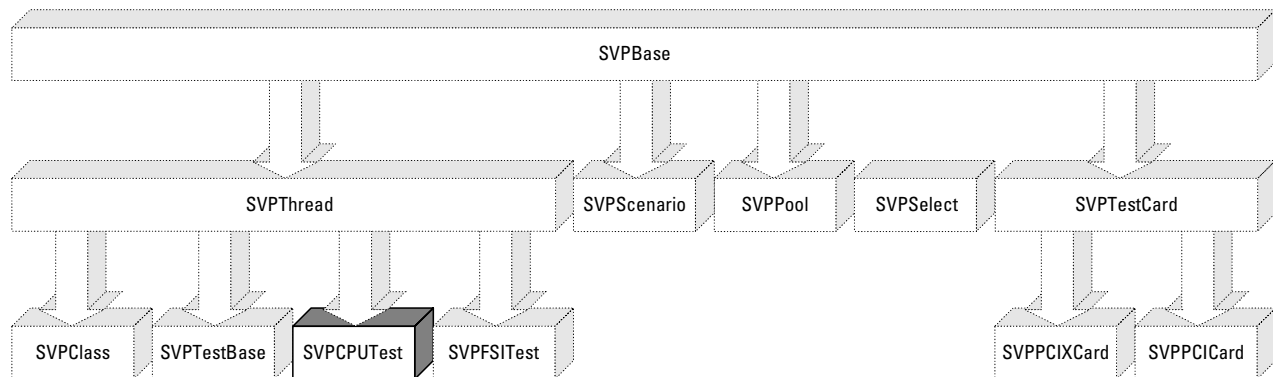
**Description** Stops the running thread.

**Return Value** No return value.

**See Also** “*Run*” on page 85

## SVPCPUTest Class

**Description** The SVPCPUTest class is used to perform tests with CPU interaction.



**Characteristic Members** This class contains no additional members that can be directly called by the user.

**Inherited Members from SVPThread** The following table lists the members inherited from the SVPThread class.

Public Methods	
virtual void	Run (void)
virtual void	Stop (void)

For detailed description of these inherited members, refer to “SVPTThread Class” on page 83.

**Inherited Members from SVPBase** The following tables list the members inherited from the SVPBase class.

Constructor
No public constructor available. Use <code>SVPClass::CreateObject</code> instead.

Destructor
No public destructor available. Use <code>SVPClass::RemoveObject</code> instead.

Various Public Methods	
virtual void	Init (void)
virtual void	Check (const BErr & errLog)
virtual void	PrepareRun (SVPPropSection * propsection)
virtual EState	Status (void) const
virtual EState	UpdateStatus (void)
virtual void	RemoveObject (const ESVPObjectType objectType, const BString & strID)
virtual void	InsertObject (SVPBase & pSvpObject)

Access to Lists of Enumerations/Direct Object Access	
virtual const SVPList *	GetChildList (void)

Identification Routines	
virtual const ESVPObjectType	GetObjectType (void) const
virtual const BString	Name (void) const
virtual void	SetID (const BString & theID)
virtual const BString &	GetID (void) const

Default Settings	
virtual void	Default (void)

Property Publication	
virtual CPropEl &	GetProp (const BString & key) const
virtual bool	CheckProp (const BString & key) const

Reporting	
virtual BLog &	Log (void)
virtual ostream &	GetLog (ostream & o)
virtual ostream &	GetNewLog (ostream & o)
virtual void	ClearLog (void)
virtual ostream &	StaticReport (ostream & o)
virtual const BErr &	GetErr (void) const

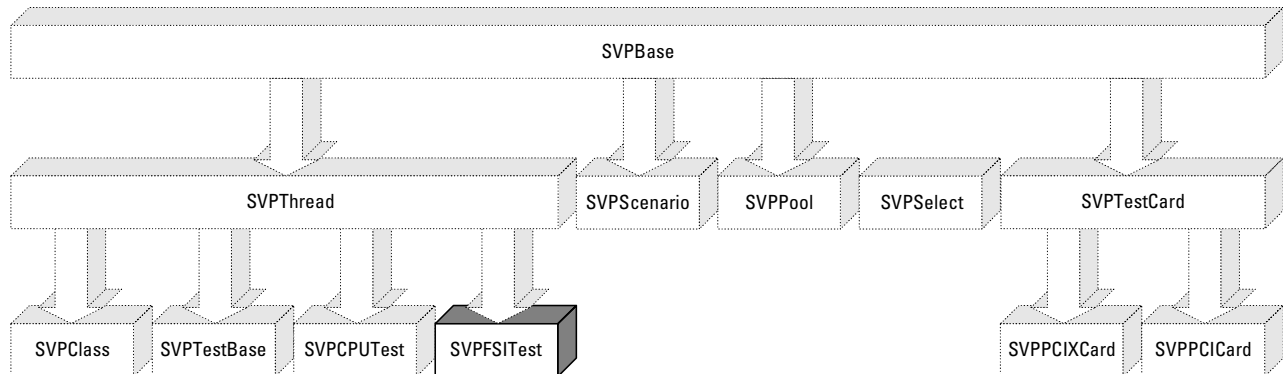
Static Member Methods	
static const ESVPObjectType	GetObjectTypeByName (const BString & name)
static const BString	GetNameByObjectType (const ESVPObjectType objectType)

For detailed description of these inherited members, refer to “*SVPBase Class*” on page 65.



# SVPFSITest Class

**Description** The SVPFSITest class is used to perform tests with FSI (Front Side Interface) interaction. The FSI is used whenever internal control of testcards is not desired or cannot be achieved.



**Characteristic Members** This class contains no additional members that can be directly called by the user.

**Inherited Members from SVThread** The following table lists the members inherited from the SVThread class.

Public Methods	
virtual void	Run (void)
virtual void	Stop (void)

For detailed description of these inherited members, refer to “SVThread Class” on page 83.

**Inherited Members from SVPBase** The following tables list the members inherited from the SVPBase class.

Constructor
No public constructor available. Use SVPClass::CreateObject instead.

Destructor
No public destructor available. Use SVPClass::RemoveObject instead.

Various Public Methods	
virtual void	Init (void)
virtual void	Check (const BErr & errLog)
virtual void	PrepareRun (SVPPropSection * propsection)
virtual EState	Status (void) const
virtual EState	UpdateStatus (void)
virtual void	RemoveObject (const ESVPObjectType objectType, const BString & strID)
virtual void	InsertObject (SVPBase & pSvpObject)

Access to Lists of Enumerations/Direct Object Access	
virtual const SVPList *	GetChildList (void)

Identification Routines	
virtual const ESVPObjectType	GetObjectType (void) const
virtual const BString	Name (void) const
virtual void	SetID (const BString & theID)
virtual const BString &	GetID (void) const

Default Settings	
virtual void	Default (void)

Property Publication	
virtual CPropEl &	GetProp (const BString & key) const
virtual bool	CheckProp (const BString & key) const

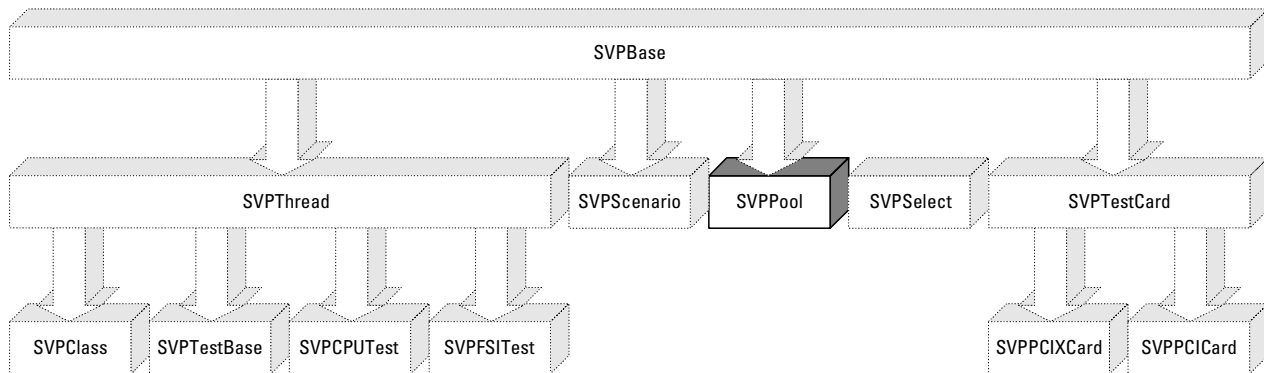
Reporting	
virtual BLog &	Log (void)
virtual ostream &	GetLog (ostream & o)
virtual ostream &	GetNewLog (ostream & o)
virtual void	ClearLog (void)
virtual ostream &	StaticReport (ostream & o)
virtual const BErr &	GetErr (void) const

Static Member Methods	
static const ESVPObjType	GetObjectTypeByName (const BString & name)
static const BString	GetNameByObjectType (const ESVPObjType objectType)

For detailed description of these inherited members, refer to “*SVPBase Class*” on page 65.

## SVPPool Class

<b>Description</b>	The SVPPool class is not to be used directly. It is used by SVPClass to handle the pool of testcards and the pool of tests.
--------------------	---



**Characteristic Members** This class contains no additional members that can be directly called by the user.

**Inherited Members** The following tables list the members inherited from the *SVPBase* class.

Constructor
No public constructor available. Use SVPClass::CreateObject instead.

Destructor
No public destructor available. Use <code>SVPClass::RemoveObject</code> instead.

Various Public Methods	
virtual void	Init (void)
virtual void	Run (void)
virtual void	Stop (void)
virtual void	Check (const BErr & errLog)
virtual void	PrepareRun (SVPPPropSection * propsection)
virtual EState	Status (void) const
virtual EState	UpdateStatus (void)
virtual void	RemoveObject (const ESVPObjType objectType, const BString & strID)
virtual void	InsertObject (SVPBase & pSvpObject)

Access to Lists of Enumerations/Direct Object Access	
virtual const SVPList *	GetChildList (void)

Static Member Methods	
static const ESVPObjType	GetObjectTypeByName (const BString & name)
static const BString	GetNameByObjectType (const ESVPObjType objectType)

Identification Routines	
virtual const ESVPObjType	GetObjectType (void) const
virtual const BString	Name (void) const
virtual void	SetID (const BString & theID)
virtual const BString &	GetID (void) const

Default Settings	
virtual void	Default (void)

Property Publication	
virtual CPropEl &	GetProp (const BString & key) const
virtual bool	CheckProp (const BString & key) const

Reporting	
virtual BLog &	Log (void)
virtual ostream &	GetLog (ostream & o)
virtual ostream &	GetNewLog (ostream & o)
virtual void	ClearLog (void)
virtual ostream &	StaticReport (ostream & o)
virtual const BErr &	GetErr (void) const

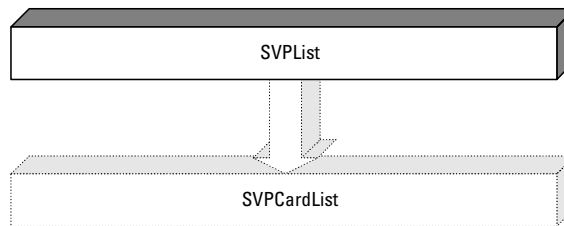
For detailed description of the inherited members, refer to “*SVPBase Class*” on page 65.

# SVPList Class

**Description** The `SVPList` class provides a container class for SVP objects. SVP objects can be:

- scenarios (SVPScenario objects)
- tests (SVPTestBase objects)
- testcards (SVPTestCard, SVPPCICard or SVPPCIXCard objects)

This class provides methods for modifying settings files and methods for walking through the lists.



**Characteristic Members** The following tables list all public members of the `SVPList` class that are recommended for direct use.

Constructor	
	<code>SVPList (SVPBase * parent, const ESVPObjectType objectType)</code>

Destructor	
<code>virtual ~</code>	<code>SVPList (void)</code>

Other Public Methods	
<code>virtual void</code>	<code>DiscardList (void)</code>

Identification Routines	
<code>const ESVPObjectType</code>	<code>GetObjectType (void) const</code>

Element Access	
virtual SVPBase *	operator [ ] (int index) const
virtual SVPBase *	operator [ ] (const BString & strID) const
virtual int	Count (void) const

String List of all Object IDs	
BString	GetObjectIDList (void) const

## Count

**Include Files** `#include <svplist.h>`

**Call** `virtual int count( void ) const;`

**Description** Counts the number of objects in this list object.

**Return Value** Number of objects as integer.

**See Also** –

## DiscardList

**Include Files** `#include <svplist.h>`

**Call** `virtual void DiscardList( void );`

**Description** Removes all elements of this list object.

**Return Value** No return value.

**See Also** –



## GetObjectIdList

**Include Files** `#include <bstring.h>`  
`#include <svplist.h>`

**Call** `BString GetObjectIdList( void ) const;`

**Description** Gets a string with a list of all object strings that are included in this list object.

**Return Value** BString object that contains all object strings of this list.

**See Also** *"GetObjectType" on page 98*

**Example**

```
BString theStr;  
theStr = svp.GetChildList()->GetObjectIdList();  
cout << "Names of all scenarios: " << theStr << endl;
```

## GetObjectType

**Include Files** `#include <svplist.h>`

**Call** `const ESVPObjectType GetObjectType( void ) const;`

**Description** Gets the type of the objects that are included in this list object.

**Return Value** Type of the objects; see “*ESVPObjectType*” on page 109.

**See Also** –

## Operator [] (int index)

**Include Files** `#include <svplist.h>`

**Call** `virtual SVPBase * operator [] ( int index ) const;`

**Description** Gets the list element (object) with the specified number.

**NOTE** The index starts at 0. The last element in the list is (Count() -1).

**Return Value** Pointer to the object.

**Input Parameter** **index** Index.

**See Also** –

## Operator [] (const BString & strID)

**Include Files** `#include <svplist.h>`

**Call** `virtual SVPBase * operator [] ( const BString & strID) const;`

**Description** Gets the list element (object) with the specified user-defined ID.

**Return Value** Pointer to the object.

**Input Parameter** **strID** User-defined name of the object.

**See Also** –

## SVPList, Constructor

**Include Files** `#include <svplist.h>`

**Call**

```
public SVPList(  
    SVPBase * parent,  
    const ESVPObjType & objectType);
```

**Description** Defines a list of objects. This list is specified by the parent object. The objects within this list are specified by the object type.

**Return Value** SVPList object.

**Input Parameters** **parent** Pointer to the parent object of this list.

**objectType** Type of the objects in this list; see “*ESVPObjType*” on page 109.

**See Also** “*SVPList, Destructor*” on page 99

## SVPList, Destructor

**Include Files** `#include <svplist.h>`

**Call**

```
public virtual ~SVPList( void );
```

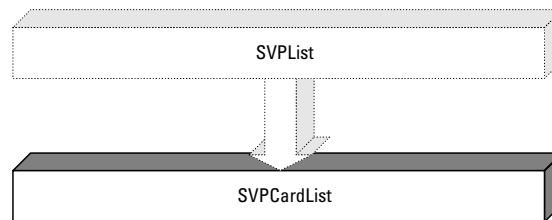
**Description** Frees the memory space that has been allocated for this SVPList object.

**Return Value** No return value.

**See Also** “*SVPList, Constructor*” on page 99

# SVPCardList Class

**Description** The SVPCardList class is able to scan the system for available testcards and to build a list of these testcards.



**Characteristic Members** This class contains no additional members that can be directly called by the user.

**Inherited Members** The SVPCardList class includes the members of the SVPList class. The following tables list the members inherited from the SVPList class.

Constructor	
	SVPList (SVPBase * parent, const ESVPObjectType objectType)

Destructor	
virtual ~	SVPList (void)

Other Public Methods	
virtual void	DiscardList (void)

Identification Routines	
const ESVPObjectType	GetObjectType (void) const

Element Access	
virtual SVPBase *	operator [ ] (int index) const
virtual SVPBase *	operator [ ] (const BString & strID) const
virtual int	Count (void) const

String List of all Object IDs	
BString	GetObjectList (void) const

For detailed description of the inherited members, refer to “*SVPList Class*” on page 95.



# Classes of the Service Library

The Service Library includes a number of classes with the following general purposes:

- Error handling; see “*BErr*” on page 103.
- String handling; see “*BString*” on page 104.
- Address handling; see “*BAddress*” on page 104.
- Generation and access of random data fields; see “*BRandomData*” on page 104.
- Live report handling; see “*BLog*” on page 105.
- Testcard’s and other device’s location handling; see “*BLocation*” on page 106.

## **BErr**

The `BErr` class handles all errors that occur within the SVP application. It is used both in the Test API library (see “*Classes of the Test API Library*” on page 11) and the FSILib library. (The FSI library is not needed for direct use. For detailed information, refer to header file `<fsilib.h>`.)

The `BErr` class provides methods for passing back and forth error messages and other error information.

For further reference information, refer to header file `<berr.h>`.

# BString

The BString class provides a common class for handling strings. This class provides basic functionality such as string concatenation, formatting and stream insertion.

For further reference information, refer to header file `<bstring.h>`.

# BAddress

The BAddress class provides methods for address handling in configuration space, memory space and I/O space. This class also provides the functionality for converting physical to virtual addresses and vice-versa.

For further reference information, refer to header file `<baddress.h>`.

# BRandomData

The BRandomData class provides a container for generation and access of random data fields. This class also provides random functions and can be used in environments where random data generation needs to be customized.

For further reference information, refer to header file `<brandom.h>`.



# BLog

**Description** The BLog class is a stream class for handling live reports. Each SVPBase object (see “SVPBase Class” on page 65) owns a BLog object for reporting.

**Characteristic Members** Only one public method is user-callable. Neither the public constructor nor the public destructor are needed outside the TestAPI library.

Constructor	
Public constructor is not needed outside the Test API library.	

Destructor	
Public constructor is not needed outside the Test API library.	

Public Method	
ostream &	GetOStream (void)

## GetOStream

**Include Files** `#include <blog.h>`

**Call** `ostream & GetOStream( void );`

**Description** Returns the reference to an ostream object. The ostream object can be, for example, cout, cerr, or an ofstream file.

**Return Value** Reference to the ostream object. This reference can be used for streaming strings into the live report.

**See Also** *“GetLog” on page 72*  
*“GetNewLog” on page 74*

## BLocation

The BLocation class provides an interface for handling the locations of testcards and other devices in the system under test. This class handles bus numbers, device numbers and function numbers and provides methods for storing and printing the bus widths and the bus speeds.

For further reference information, refer to header file `<brandom.h>`.

# Enumeration Definitions

All enumeration definitions are listed in alphabetically order:

- “*EAddressSpace*” on page 107 defines available address spaces such as I/O or system memory.
- “*ECardType*” on page 108 defines available types of testcards for PCI and PCI-X bus systems.
- “*EState*” on page 108 defines different states of an object, for example, whether a test object is running or waiting.
- “*ESVPObjectType*” on page 109 defines available object types, such as test or testcard objects.

## EAddressSpace

enum EAddressSpace	Description
SPACE_IO	I/O address space.
SPACE_MEM	System main memory or memory of the testcard (depends on the kind of test).
SPACE_CONF	Configuration address space.

# ECardType

enum ECardType	Description
UNKNOWN	Unknown testcard
PCI	PCI testcard (for example, E2925B, E2926B)
COMPACT_PCI	Compact PCI testcard (E2940A)
PCI-X	PCI-X testcard (E2929A, E2922A)

# EState

enum Estate	Description
S_UNKNOWN	Unknown state.
S_INIT	Object is initialized and ready to run.
S_WAIT	Object is preparing to run.
S_STOP	Object is stopped.
S_RUN	Object is running.
S_PAUSE	Object is paused.
S_ERROR	Object has encountered a run-time error (recoverable).
S_EXCLUDE	Object is excluded or has an unrecoverable error condition.

# ESVPObjectType

enum ESVPObjectType	Description
T_TESTCARD	Defines the object type of a testcard.
T_SVP	Defines the object type of the whole SVP application.
T_SCENARIO	Defines the object type of a scenario.
T_TEST	Defines the object type of a test.
T_CPUTEST	Defines a test that performs CPU interaction.
T_FSITEST	Defines a test that performs FSI interaction.



# Setup File Reference

All settings of the System Validation Package 2.1 can easily be set or modified by the user via C++ programming.

For property setting and modifying, use the names and value ranges that are listed in all following tables. You can find:

- One scenario property and properties of the available tests  
The scenario property and the test properties with their value ranges are described in *“Scenario and Test Parameter” on page 112.*
- Properties of the available testcards  
The testcard properties and their value ranges are described in *“Testcard Parameters” on page 113.*

For more information, especially on PPR properties, please refer to the *Agilent C-API/PPR Programming Reference*, which is delivered with the respective testcard.

# Scenario and Test Parameter

**Scenario Property** Scenarios allow several tests to be run concurrently. Any testcard can only be used once per scenario. Scenarios have no special settings except for the list of tests that are used.

Scenario Property	Scenario Property used in the User Program	Values	Description
Scenario	items.list	string list	List of tests (by names) that are used in this scenario

**Test Properties** All tests share some or all of the following properties:

Test Property	Test Property used in the User Program	Values	Description
Address Offset	address	address value	Physical address of the memory
Address Space	address.space	"mem" or "io"	Memory space or I/O space is used
Bytes to Transfer	size	DWORD	Size of the memory (in bytes) that is use.
Prefetchable Decoder	address.prefetch	True or False	The pre-fetchable decoder is used if available
Bandwidth %	bandwidth	0 ... 100	Value of the maximum bandwidth. <b>Note:</b> Using a bandwidth < 1.0 will cause that the PPR testcard setting B_M_DELAY is overridden.
Description	description	string	User-defined description
Function	function	string	Short name of test function
Start Delay	starttimeoffset	DWORD	Start Delay in seconds (from start of scenario)
Duration	duration	DWORD	Duration of the test (in seconds)



# Testcard Parameters

Testcard parameters can be divided into:

- Testcard and Location Information
- Card Features Settings
- Master Settings (for PCI Testcards)
- Target Settings (for PCI Testcards)
- Requester Settings (for PCI-X Testcards)
- Completer Settings (for PCI-X Testcards)
- Protocol Checker (Rule Masking)

## Testcard and Location Information

Card Property	Testcard Property used in the User Program	Type	Range	Description
Serial Number	card.serialnumber	string	valid serial number	Serial number of the testcard
Port	connection.port	port string	rs232   fhif   pci   usb (PCI-X only)	Connection port of the testcard
Port Number	connection.portnum	DWORD	depends on port	Connection port number
Model Number	card.model	string	valid models	Model number of the testcard
Location	card.location	string	valid locations	Location of the testcard (for example, Bus 0 Device 13 Function 0)

## Card Features Settings

Testcard Property	Testcard Property used in the User Program	Type	Range	Description
Use Performance	use.performance	boolean	True or False	Performance counters of the testcard are enabled or disabled
Use Protocol Checker (Rule Masking)	use.observer	boolean	True or False	Protocol Observer of the testcard is enabled or disabled
Use Master	use.master	boolean	True or False	Master of the testcard is enabled or disabled
Use Target	use.target	boolean	True or False	Target of the testcard is enabled or disabled
Use PPR	use.ppr	boolean	True or False	Attribute permutation is enabled or disabled
Use Analyzer	use.tracememory	boolean	True or False	Analyzer of the testcard is enabled or disabled
Use Trigger I/O Lines	use.triggerio	boolean	True or False	Cross-triggering is enabled or disabled
Upload Trace on Trigger	tracememory.upload	boolean	True or False	Trace memory upload is enabled or disabled
n/a	tracememory.trigger.proterr	boolean	True or False	Trace memory trigger on protocol error is enabled or disabled
n/a	tracememory.trigger.datacmp	boolean	True or False	Trace memory trigger on data compare error is enabled or disabled
n/a	tracememory.trigger.bushang	boolean	True or False	Trace memory trigger on bushang is enabled or disabled
n/a	tracememory.trigger.spliterr	boolean	True or False	Trace memory trigger on split error is enabled or disabled
Inhibit FSI	inhibit.fsi	boolean	True or False	Inhibit connection to FSI on host
File	tracememory.upload.file	string		Name of the file to which the card's trace memory is written (without extension)
n/a	performance.measure	DWORD	0 ... 7	Measure used by performance
n/a	performance.cardmeasure	DWORD	0 ... 7	Measure used by card's performance

## Master Settings (for PCI Testcards)

Testcard Property	Testcard Property used in the User Program	Type	Range	Description
n/a	master.blockpage	DWORD	0 ... 16	Blockpage used by the master
Write Command	master.block.cmd.write	DWORD	mem_write, mem_writeinvalidate	PCI bus command for the block transfer (mem)
Read Command	master.block.cmd.read	DWORD	mem_read, mem_readline, mem_readmultiple	PCI bus command for the block transfer during address phase (mem)
Master Internal Address	master.address.internal	DWORD	0 ... size of data memory	Internal address of the testcard's data memory; used by the master
Block Size List (Master PPR)	ppr.master.block.size.list	string	Multiple of 4 in the range of 4 ... 128k	List of numeric values for block sizes, measured in bytes
Block Algorithm (Master PPR)	ppr.master.block.alg	DWORD	0 ... 3	Algorithm used to pick values from the value list of master block properties
Block Byte Enable List (Master PPR)	ppr.master.block.byten.list	string	0 ... 15	List of numeric values for C/BE byte enables
Block Commands List (Master PPR)	ppr.master.block.cmds.list	DWORD	0 ... 15	List of PCI bus commands used for permutations
Block Alignment List (Master PPR)	ppr.master.block.align.list	string	<i>Granularity:</i> Power of 2 between cacheline size and 8192.  <i>Offset:</i> Multiple of 4 between 0 and 8188	Granularity and offset within this granularity that restrict the start of a block
PPR Report (Master Block)	ppr.master.block.report	boolean	True or False	Writing of the PPR report is enabled or not
PPR Block Report File (Master Block)	ppr.master.block.report.file	string		Name of the file used for the PPR report or master block permutations
Master Attribute Page (memory)	master.attrpage.memory	DWORD	0 ... 63	Attribute page used for access to the memory space by testcard
Master Attribute Page (i/o)	master.attrpage.io	DWORD	0 ... 63	Attribute page used for access to the I/O space by testcard
Waits List (Master PPR Attribute)	ppr.master.attr.waits.list	string	0 ... 30	List of numbers of waits
Burst Length List (Master PPR Attribute)	ppr.master.attr.last.list	string	0 ... 2 <sup>32</sup>	List of last phases of bursts (this is, burst lengths)
Release Request (Master PPR Attribute)	ppr.master.attr.rreq.list	string	0 ... 15	List of number of cycles after which REQ# is released after assertion of FRAME#

Testcard Property	Testcard Property used in the User Program	Type	Range	Description
DPERR List (Master PPR Attribute)	ppr.master.attr.dperr.list	string	0 or 1	List of parity errors, signaled or not signaled
SPERR List (Master PPR Attribute)	ppr.master.attr.dserr.list	string	0 or 1	List of system errors in the data phase, signaled or not signaled
APERR List (Master PPR Attribute)	ppr.master.attr.aperr.list	string	0 or 1	List of system errors in the address phase, signaled or not signaled
DWRPAR List (Master PPR Attribute)	ppr.master.attr.dwp.list	string	0 or 1	List of wrong parities set one clock after a write data transfer, inverted or not inverted
AWRPAR List (Master PPR Attribute)	ppr.master.attr.awp.list	string	0 or 1	List of wrong parities set one clock after the address phase, inverted or not inverted
WAITMODE List (Master PPR Attribute)	ppr.master.attr.waitmode.list	string	0 or 1	List of values to keep the address constant during the WAITS phases or not
STEPMODE List (Master PPR Attribute)	ppr.master.attr.stepmode.list	string	0 or 1	List of values to keep the address constant during the STEPS phases or not
STEPS List (Master PPR Attribute)	ppr.master.attr.steps.list	string	0 or 1	List of numbers of additional clocks during an address phase They are added between assertion of GNT# and assertion of FRAME#.
TRYBACK List (Master PPR Attribute)	ppr.master.attr.tryback.list	string	0 or 1	List of Fast Back-to-Back cycle tries
DELAY List (Master PPR Attribute)	ppr.master.attr.delay.list	string	2 ... 2 <sup>21</sup>	List of numbers of clocks a master transaction is delayed before its start <b>Note:</b> Delay will be modified if bandwidth < 100 % (specified in test).
REQ64 List (Master PPR Attribute)	ppr.master.attr.req64.list	string	0 or 1	List of 64-bit transfer tries
AWRPAR64 List (Master PPR Attribute)	ppr.master.attr.awp64.list	string	0 or 1	List of wrong parities (PAR64) set one clock after the address phase, inverted or not inverted
DACWRPAR (Master PPR Attribute)	ppr.master.attr.dacwp.list	string	0 or 1	List of wrong parities signaled in the second cycle of a dual address cycle, inverted or not inverted
DACWRPAR64 (Master PPR Attribute)	ppr.master.attr.dacwp64.list	string	0 or 1	List of wrong parities (PAR64) signaled in the second cycle of a dual address cycle, inverted or not inverted
DACPERR List (Master PPR Attribute)	ppr.master.attr.dacperr.list	string	0 or 1	List of system errors in the second cycle of a dual address cycle, signaled or not signaled
DWRPAR64 List (Master PPR Attribute)	ppr.master.attr.dwp64.list	string	0 or 1	List of wrong parities (PAR64) set one clock after a write data transfer, inverted or not inverted

Testcard Property	Testcard Property used in the User Program	Type	Range	Description
RESUMEDELAY List (Master PPR Attribute)	ppr.master.attr.resumelay.list	string	2 ... 127	List of clock numbers after which the master resumes after a target termination
PPR Report (Master Attribute)	ppr.master.attr.report	boolean	True or False	Writing of the PPR report is enabled or not
PPR Report	ppr.report	boolean	True or False	Writing of the PPR report is enabled or not
PPR Report File	ppr.report.file	boolean	True or False	Name of the file used for the PPR report of completer permutations

## Target Settings (for PCI Testcards)

Testcard Property	Testcard Property used in the User Program	Type	Range	Description
Target Attribute Page Number	target.attrpage	DWORD	0 ... 63	Attribute page used by the target
Termination List (Target PPR Attribute)	ppr.target.attr.term.list	string	0 ... 3	List of termination modes, for example, "32*noterm, 2*retry, disconnect, abort"
WAITS List (Target PPR Attribute)	ppr.target.attr.waits.list	string	0 ... 30	List of number of waits
DPERR List (Target PPR Attribute)	ppr.target.attr.dperr.list	string	0 or 1	List of parity errors, signaled or not signaled
SPERR List (Target PPR Attribute)	ppr.target.attr.dserr.list	string	0 or 1	List of system errors in the data phase, signaled or not signaled
APERR List (Target PPR Attribute)	ppr.target.attr.aperr.list	string	0 or 1	List of parity errors in the address phase, signaled or not signaled
WRPAR List (Target PPR Attribute)	ppr.target.attr.wp.list	string	0 or 1	List of wrong parities set one clock after a write data transfer, inverted or not inverted
ACK64 List (Target PPR Attribute)	ppr.target.attr.ack64.list	string	0 or 1	List of 64-bit requests, acknowledged or not acknowledged
Target Attribute DACPERR List	ppr.target.attr.dacperr.list	string	0 or 1	List of address parity errors, signaled or not signaled
Target Attribute WRPAR64 List	ppr.target.attr.wp64.list	string	0 or 1	List of wrong parities set one clock after a write data transfer, inverted or not inverted
Target Attribute Report	ppr.target.attr.report	boolean	True or False	Writing of the PPR report is enabled or not
Target Attribute Report File	ppr.target.attr.reportfile	string		Name of the file used for the PPR report or target attribute permutations

## Requester Settings (for PCI-X Testcards)

Testcard Property	Testcard Property used in the User Program	Type	Range	Description
BUSCMD List (RI PPR Block)	ppr.ri.blk.buscmd.list	DWORD	0 ... 15	List of PCI-X bus commands used for permutations
BYTEN List (RI PPR Block)	ppr.ri.blk.byten.list	string	0 ... 15	List of numeric values for byte enables  This parameter is only valid for block transfers with the command <i>memwrite</i> .
ALIGN List (RI PPR Block)	ppr.ri.blk.align.list	string	0 ... 7	List of alignments to QWORD boundary
NUMBYTES List (RI PPR Block)	ppr.ri.blk.numbytes.list	string	0 ... MaxDWord	List of number of bytes in the current block
RELAXORDER List (RI PPR Block)	ppr.ri.blk.relaxorder.list	string	0 or 1	List of relaxed ordering bits (relaxed ordering is done or not done)
NOSNOOP List (RI PPR Block)	ppr.ri.blk.nosnoop.list	string	0 or 1	List of bits that signify whether no snoop will be done
BYTECOUNT List (RI PPR Behavior)	ppr.ri.beh.bytecount.list	string	0 ... 4096	List of numeric values for byte counts. Sequences are generated with these byte counts
DISCONNECT List (RI PPR Behavior)	ppr.ri.beh.disconnect.list	string	0 ... 32	List of values that determine whether and how often a requester-initiator will disconnect its current sequence
DELAY List (RI PPR Behavior)	ppr.ri.beh.delay.list	string	1 ... 65535	List of numbers of clock delays
STEPS List (RI PPR Behavior)	ppr.ri.beh.steps.list	string	0 ... 32	List of address steps numbers
REQ64 List (RI PPR Behavior)	ppr.ri.beh.req64.list	string	0 or 1	List of 64-bit data transfer tries
RELREQ List (RI PPR Behavior)	ppr.ri.beh.relreq.list"	string	1 ... 2047	List of number of cycles after which REQ# is released after assertion of FRAME#
DECSPEED List (RT PPR Behavior)	ppr.rt.beh.decspeed.list"	string	A, B, C	List of decode speeds

Testcard Property	Testcard Property used in the User Program	Type	Range	Description
ACK64 List (RT PPR Behavior)	ppr.rt.beh.ack64.list	string	No, Yes	List that determine s whether a 64-bit data transfer acknowledge will be asserted or will not be asserted
INITIAL List (RT PPR Behavior)	ppr.rt.beh.initial.list	string	Accept Single Retry TAbort	List that determines the initial behavior of the requester-target response
LATENCY List (RT PPR Behavior)	ppr.rt.beh.latency.list	string	3 ... 34	List of numbers of initial latency clocks
SUBSEQ List (RT PPR Behavior)	ppr.rt.beh.subseq.list	string	0 or 1	List that specifies the target response in subsequent data phases  0: accepts all subsequent data phases 1: disconnects in the selected data phase
SUBSEQPHASE List (RT PPR Behavior)	ppr.rt.beh.subseqphase.list	string	0 ... 2047	List of selected subsequent data phases

## Completer Settings (for PCI-X Testcards)

Testcard Property	Testcard Property used in the User Program	Type	Range	Description
PPR Report	ppr.report	boolean	True or False	Writing of the PPR report is enabled or not
PPR Report File	ppr.report.file	boolean	True or False	Name of the file used for the PPR report of completer permutations
PARTITION List (CI PPR Behavior)	ppr.ci.beh.partition.list	string	0 ... 63	List that defines the sizes of the (partial) completion transactions.
ERRMESSAGE List (CI PPR Behavior)	ppr.ci.beh.errmessage.list	string	0 or 1	0: normal split completion transaction 1: split completion error message
DELAY List (CI PPR Behavior)	ppr.ci.beh.delay.list	string	1 ... 65535	List of numbers of clock delays before REQ# is asserted
STEPS List (CI PPR Behavior)	ppr.ci.beh.steps.list	string	2 ... 6	List of numbers of address steps
REQ64 List (CI PPR Behavior)	ppr.ci.beh.req64.list	string	0, 1	1: 64-bit data transfer  0: No 64-bit data transfer access

Testcard Property	Testcard Property used in the User Program	Type	Range	Description
RELREQ List (CI PPR Behavior)	ppr.ci.beh.relreq.list	string	1 ... 2047	List of number of cycles after which REQ# is released after assertion of FRAME#
DECSPEED List (CT PPR Behavior)	ppr.ct.beh.decspeed.list	string	A, B, C	List of decode speeds
ACK64 List (CT PPR Behavior)	ppr.ct.beh.ack64.list	string	No, Yes	List that determine s whether a 64-bit data transfer acknowledge will be asserted or will not be asserted
INITIAL List (CT PPR Behavior)	ppr.ct.beh.initial.list	string	Accept Single Retry TAbort	List that determines the initial behavior of the completer-target response
LATENCY List (CT PPR Behavior)	ppr.ct.beh.latency.list	string	3 ... 34	List of numbers of initial latency clocks
SUBSEQ List (CT PPR Behavior)	ppr.ct.beh.subseq.list = "Accept,Disconnect"	string	Accept Disconnect	List that specifies the target response in subsequent data phases.  0: accepts all subsequent data phases 1: disconnects in the selected data phase
SUBSECPHASE List (CT PPR Behavior)	ppr.ct.beh.subseqphase.list	string	0 ... 2047	List of selected subsequent data phases
SPLITLATENCY List (CT PPR Behavior)	ppr.ct.beh.splitlatency.list	string	3 ... 18	List of numbers of wait cycles.  A split response is signaled after the specified number of wait cycles.
SPLITENABLE List (CT PPR Behavior)	ppr.ct.beh.splitenable.list	string	0 or 1	0: No split response will be generated  1: A split response will be generated



## Protocol Checker (Rule Masking)

Testcard Property	Testcard Property used in the User Program	Type	Range	Description
Protocol Rule Masking State Disabled (lower bits)	protocolrule.mask.lo	DWORD	32	Masked protocol rules (bit 0 ... 31).
Protocol Rule Masking State Disabled (higher bits)	protocolrule.mask.hi	DWORD	20	Masked protocol rules (bit 32 ... 51).
Mask Rule(s) After x Occurrences	protocolrule.mask.count	DWORD	0 ... ( $2^{32} - 1$ )	Number of occurrences after which a rule is masked automatically. ( 0 : rule(s) will never be masked)





# Overall Example Programs

The following sections provide example programs of setting up tests for system validation:

- A simple command line executable shows how to run tests immediately.
- A custom test function shows how to configure tests for individual test needs. In this example, a custom memory read test will be set up.

Both example programs show how to encapsulate calls to Test API functions and class methods in `try` blocks. A `try` block identifies a code block in which an exception can occur.

# Simple Command Line Executable

The following example shows:

- How to run tests.
- How to get the static report.
- The C++ exception handling mechanism.

You will find this example under `svp\samples\svpexe.cpp`.

```
#include <iostream.h>

// include servlib (for BErr mainly)
#include <servlib.h>

// include testapi
#include <testapi.h>

//-----
// int main (int argc, char * argv[])
//
// Purpose : main function
// Returns : error code on error, 0 on success
//-----

int main (int argc, char * argv[])
{
    SVPClass svp; // The SVP object

    int retval = 0; // return value of program (0 - no errors)

    try
    {
        cerr << "Initializing SVP class..." << endl;

        // initialize svp from command line
        // call svpexe -h for help on command line options
        svp.CommandLineInit(argc, argv);

        cerr << "Printing out static report..." << endl;
        // print static report to stdout
        svp.StaticReport(cout);

        cerr << "Running Tests..." << endl;

        // run tests
        svp.Run();
    }
}
```

```

        // print log (test report) to std out
        svp.GetLog(cout);
        cerr << "Querying status..." << endl;

        while (svp.Status() == SVPBase::S_RUN || svp.Status() ==
                SVPBase::S_WAIT)
        {
            // wait for ten seconds

            Sleep (10000); // replace this with/add additional test
                           code

            // report additions to test report to stdout
            svp.GetNewLog(cout);
        }
    } // end of try block

    // any errors? All TestAPI functions will throw BErr objects as
    error

    catch (BErr theErr)
    {
        // print error report to std err

        cerr << argv[0] << ": ERROR condition:" << endl << theErr <<
        endl;

        // convert error code to return value
        retval = (int) theErr();
    }

    // catch other errors generated elsewhere
    catch (...)
    {
        cerr << "Caught unknown exception somewhere" << endl;
    }

    cout << flush; // make sure everything is on console here
    // wait for return key pressed
    cout << "Press return key to finish" << flush;
    getchar();
    cout << endl;
    return retval;
}

```

# Custom Test Function

The following example shows you how to set up a custom memory read test. It shows you:

- How to create a new test.
- How to set test properties.
- How to assign a testcard to a test.
- How to assign a test to a scenario.
- The C++ exception handling mechanism.
- Two custom memory read functions, one of these functions initializes the test and the other runs it.

You will find this example under `svp\samples\svpcustom.cpp`.

```
#include <iostream.h>
#include <servlib.h>
#include <testapi.h>

//*****
// function declarations:
// declare your custom test functions here
// NOTE: declare these before setting up STestFct object!
//*****

void custommemread_init (SVPTestBase * svpTest);
void custommemread_run (SVPThread * svpThread);
// no stop function
```

```

/*****
 * this struct contains the actual test definitions
 *****/

STestFct g_customFct =
{
    "Custom Memory Read", // descriptive name
    "custommemread", // short name
    custommemread_init, // init function
    custommemread_run, // run function
    0, // no stop function
    1, 1, // one card only
    "Reads from main memory and does other custom testing", // long
    description
    tpf_FULL_ADDRESS | tpf_BANDWIDTH // test flags
};

/*****
 * main function body
 *****/

//-----
// int main (int argc, char * argv[])
//
// Purpose : main function
// Returns : error code on error, 0 on success
//-----

int main (int argc, char * argv[])
{
    SVPClass svp; // THE SVP object
    int retval = 0; // return value
    try
    {
        // some initial setup
        cerr << "Initializing SVP class..." << endl;
        svp.Default(); // set everything to default
        svp.Init(); // initialize SVP object
        // create new test object
        SVPTTestBase * newTest = (SVPTTestBase *) svp.CreateObject(T_TEST);
        SVPAssert(newTest); // make sure there are no errors
    }
}

```

```

// assign custom test function
newTest->SetTestFct(&g_customFct);
newTest->SetID("My custom test");
// set some test properties
newTest->GetProp("duration") = 120UL; // duration 120 seconds
BAddress testAddress(0xb8000UL, 0UL, BAddress::SPACE_MEM);
// address to use for test
newTest->GetProp("address") = testAddress;
// set address property of test
// assign card to test (first testcard found)
SVPBase * theCard = (*(svp.GetCardList()))[0];
newTest->InsertObject(theCard);
// insert test into first scenario
(*svp.GetChildList())[0]->InsertObject(newTest);
// run test
cerr << "Running Tests..." << endl;
svp.Run();
// print report to cout
svp.GetLog(cout);
// start query loop
cerr << "Querying status..." << endl;

while (svp.Status() == SVPBase::S_RUN || svp.Status() ==
      SVPBase::S_WAIT)
{
    Sleep(10000); // replace this with/add additional test code
    svp.GetNewLog(cout); // place additions to log on standard out
}
}

```



```
// any errors?
catch (BErr theErr)
{
    // print program name and error msg:
    cerr << argv[0] << ": ERROR condition:" << endl;
    // print error message to std err:
    cerr << theErr << endl;
    // prepare main's return value
    retval = (int) theErr();
}

// catch unknown errors (SVP will only throw BErr objects)
catch (...)
{
    cerr << "Caught unknown exception" << endl;
}

cout << flush; // make sure everything is on console here
// wait for return key pressed
cout << "Press return to exit" << flush;
getchar();
cout << endl;
return retval;
}
```

```

//*****
// custom memory read test
//*****
//-----
// custommemread_init
// Purpose : initialize custom memory read test
// Inputs : testbase pointer
//-----

void
custommemread_init (SVPTTestBase * svpTest)
{
    BErr::DbgPrint("Entering custommemread_init\n");
    SVPAssert (svpTest);
    // get address of test (from address property)
    BAddress address = svpTest->GetProp("address");
    address.SetSize(svpTest->GetProp("size"));
    // get first card in test's list
    SVPTTestCard * theCard = (SVPTTestCard *) svpTest->GetCardList()[0];
    theCard->MasterReadSetup(address);
    // important: give info about test function to testcard!
    SVPTThread * newCPU = new SVPCPUTest(
        theCard, svpTest->GetTestFct(), address);
    theCard->InsertObject(newCPU);
}

//-----
// custommemread_run
// Purpose : run custom memory read test
//-----

void
custommemread_run (SVPTThread * svpThread)
{
    BErr::DbgPrint("Entering custom memread test run \n");
    SVPAssert (svpThread->GetObjectType() == T_CPUTEST);
    SVPCPUTest * svpCPUTest = (SVPCPUTest *) svpThread;
    const BAddress & theAddr = svpCPUTest->GetAddress();
    BLog & cpuLog = svpCPUTest->Log();
    // the following line is put in the test log...
    cpuLog.GetOStream() << "Entered Run Function" << endl;
    // this is the place to code your own test function, e.g.
    // (...)
}

```

# Index

## A

---

accessing objects for direct modification 14

## B

---

BAddress  
    Service Library class 104  
BAddress class 104  
BErr  
    Service Library class 103  
BErr class 103  
BLocation  
    Service Library class 106  
BLocation class 106  
BLog  
    Service Library class 105  
BLog class 105  
BLog member  
    GetOStream 106  
BRandomData  
    Service Library class 104  
BRandomData class 104  
BString  
    Service Library class 104  
BString class 104

## C

---

C++ compiler 9  
C-API  
    drivers 9  
card features  
    properties (Test API) 114  
class SVPPropBase 13  
classes  
    BAddress 104  
    BErr 103  
    BLocation 106  
    BLog 105  
    BRandomData 104  
    BString 104  
    SVPCardList 100  
    SVPClass 14  
    SVPCPUTest 86  
    SVPFSTest 89  
    SVPList 95  
    SVPList package 13  
    SVPPCICard 59  
    SVPPCIXCard 62  
    SVPPool 92  
    SVPSenario 28  
    SVPSelect 12

SVPTTestBase 32  
SVPTTestCard 36  
SVPThread 83  
completer  
    PCI-X testcard properties (Test API) 119  
constants  
    COMPACT\_PCI 108  
    PCI 108  
    PCI-X 108  
    S\_ERROR 108  
    S\_EXCLUDE 108  
    S\_INIT 108  
    S\_PAUSE 108  
    S\_RUN 108  
    S\_STOP 108  
    S\_UNKNOWN 108  
    S\_WAIT 108  
    SPACE\_CONF 107  
    SPACE\_IO 107  
    SPACE\_MEM 107  
    T\_CPUTEST 109  
    T\_FSITEST 109  
    T\_SCENARIO 109  
    T\_SVP 109  
    T\_TEST 109  
    T\_TESTCARD 109  
    UNKNOWN 108

constructor  
    SVPClass 27  
    SVPList 99  
creation of objects 15  
custom test function  
    example 126

## D

---

deletion of objects 15  
destructor  
    SVPClass 27  
    SVPList 99  
direct modification of objects 14  
drivers  
    C-API 9  
    Windows NT 9

## E

---

EAddressSpace  
    enumeration definition 107  
ECardType  
    enumeration definition 108

enumeration definition  
    EAddressSpace 107  
    ECardType 108  
    EState 108  
    ESVPObjType 109  
EState  
    enumeration definition 108  
ESVPObjType  
    enumeration definition 109  
example program  
    custom test function 126  
    simple command line executable 124  
exception  
    try block 10

## F

---

file handling 14

## H

---

hierarchy  
    objects 15

## I

---

initialization 14

## M

---

master  
    testcard properties (Test API) 115

## N

---

naming  
    classes 10  
    constants 10  
    enumerations 10  
    methods 10  
    variables 10

## O

---

objects  
    access 14  
    creation 15  
    deletion 15  
    hierarchy 15  
operators  
    SVPList 98  
overview  
    SVPPBase package 11

**P**

- package
  - SVPPropBase 12
- PCI-X testcard settings (Test API)
  - completer 119
  - requester 118
- platform independence 9
- platform-dependent features 9
- properties (Test API)
  - card features 114
  - master settings 115
  - protocol checker 121
  - rule masking 121
  - target settings 117
  - testcard and location information 113
- protocol checker
  - testcard properties (Test API) 121
- public members 7
  - SVPBase 65
  - SVPClass 15
  - SVPList 95
  - SVPScenario 28
  - SVPTestBase 32
  - SVPTestCard 36
  - SVPThread 83

**R**

- reporting 14
- requester
  - PCI-X testcard properties (Test API) 118
- rule masking
  - testcard properties (Test API) 121

**S**

- Service Library classes 103
  - BAddress 104
  - BErr 103
  - BLocation 106
  - BLog 105
  - BRandomData 104
  - BString 104
- simple command line executable
  - example program 124
- SVPBase
  - public members 65
  - purpose 65
- SVPBase member
  - Check 68
  - CheckProp 69
  - ClearLog 70
  - Default 70
  - GetChildList 71
  - GetErr 71
  - GetID 72
  - GetLog 72
  - GetNameByObjectType 73
  - GetNewLog 74

- GetObjectType 75
- GetObjectTypeByName 74
- GetProp 75
- Init 76
- InsertObject 77
- Log 78
- Name 78
- PrepareRun 78
- RemoveObject 79
- Run 80
- SetID 80
- StaticReport 81
- Status 81
- Stop 82
- UpdateStatus 82

- SVPBase package
  - overview 11

- SVPCardList
  - class 100
  - purpose 100

- SVPClass
  - class 14
  - constructor 27
  - destructor 27
  - public members 15
  - purpose 14

- SVPClass member
  - Check 18, 19
  - CommandLineInit 18
  - CommandLineUsage 19
  - Default 20
  - FileLoad 20
  - FileSave 21
  - FileSaveAs 21
  - GetCardList 22
  - GetNewLog 22
  - GetTotalDuration 24
  - GetSelectionObject 23
  - GetTestList 23
  - InsertObject 24
  - OfflineMode 25
  - RemoveObject 25
  - Run 26
  - StaticReport 26
  - Stop 27
  - SVPClass, constructor 27
  - SVPClass, destructor 27

- SVPCTest
  - class 86
  - purpose 86

- SVPPFSITest
  - purpose 89

- SVPList
  - class 95
  - constructor 99
  - destructor 99
  - public members 95
  - purpose 95

- SVPList member
  - Count 96
  - DiscardList 96
  - GetObjectIdList 97
  - GetObjectType 98
  - Operators 98
  - operators 98
  - SVPList, constructor 99
  - SVPList, destructor 99

- SVPList package
  - classes 13

- SVPPCICard
  - class 59

- SVPPCIXCard
  - class 62

- SVPPFSITest
  - class 89

- SVPPool
  - class 92
  - purpose 92

- SVPPPropBase class 13

- SVPPPropBase package 12

- SVPSScenario
  - class 28
  - public members 28

- SVPSScenario member
  - GetTotalDuration 31

- SVPSSelect class 12

- SVPTTestBase
  - class 32
  - public members 32
  - purpose 32

- SVPTTestBase member
  - GetTotalDuration 35

- SVPTTestCard
  - class 36
  - public members 36

- SVPTTestCard member
  - AllocateBuffer 40
  - BandwidthSet 40
  - CardType 41
  - CardTypeFromString 41
  - CheckPPR 42
  - ConfigScan 43
  - CPUTargetSetup 44
  - Default 45
  - GetAddress 46
  - GetLocation 47
  - GetRule 47
  - GetRuleCount 48
  - GetRuleDescription 48
  - GetSystemID 49
  - MasterReadSetup 50
  - MasterWRCSetup 51
  - MasterWriteSetup 52
  - New 53
  - ObserverTestSetup 54
  - Operator == 54
  - Ping 55

- ResetFactoryDefault 55
- Run 56
- SetRule 56
- StaticReport 57
- Stop 58
- SVPThread
  - class 83
  - public members 83
  - purpose 83
- SVPThread member
  - Run 85
  - Stop 85

## T

---

- target
  - testcard properties (Test API) 117
- Test API Library
  - classes 11
- test execution 14
- testcard information
  - properties (Test API) 113
- testcard location
  - properties (Test API) 113
- testcard settings
  - testcard location 113
- testcard settings (Test API)
  - master 115
  - protocol checker 121
  - target 117
  - testcard features 114
  - testcard information 113
- try block 10

## W

---

- Windows NT drivers 9