User Guide

Tcl Automation on Abacus™

October 2009

P/N 71-006069 Rev A

Spirent Communications, Inc.

26750 Agoura Road
Calabasas, CA
91302 USA

Copyright

Limited Warranty

Contents

About this Guide

In this chapter...

Introduction

This user manual provides an overview of Tcl Automation. It describes common product architecture and details the product functionality through a class reference. The manual also provides information that allows users to compose scripts and perform remote tests using the Tcl application programming interface (API).

This user manual is meant for personnel who will prepare scripts to run tests with Abacus performance-analysis hardware and software using the Tcl API. These computers may employ operating systems (OS) other than Microsoft Windows.

Related Documentation

- *Tcl Automation on Abacus Data Holder Classes Reference Guide*
- *Abacus 5000 IP Telephony Migration Test System Software Manual*
- *Abacus 5000 IP Telephony Migration Test System Hardware Manual*

How to Contact Us

To obtain technical support for any Spirent Communications product, please contact our Support Services department using any of the following methods:

Americas

E-mail: support@spirent.com
Web: http://support.spirent.com
Toll Free: +1 800-SPIRENT (+1 800-774-7368) (US and Canada)
Phone: +1 818-676-2616
Fax: +1 818-880-9154
Hours: Monday through Friday, 05:30 to 18:00, Pacific Time

Europe, Africa, Middle East

E-mail: support@spirent.com
Web: http://support.spirent.com
Phone: +33 (0) 1 61 37 22 70
Fax: +33 (0) 1 61 37 22 51
Hours: Monday through Thursday, 09:00 to 18:00, Friday, 09:00 to 17:00, Paris Time

Asia Pacific

E-mail: supportchina@spirent.com
Web: http://support.spirent.com
Phone: 400 810 9529 (mainland China)
Phone: +86 400 810 9529 (outside China)
Fax: +86 10 8233 0022
Hours: Monday through Friday, 09:00 to 18:00, Beijing Time

The latest versions of user manuals, application notes, and software and firmware updates are available on the Spirent Communications support website at http://support.spirent.com.

Information about Spirent Communications and its products and services can be found on the main company website at http://www.spirent.com.

Company Address

Spirent Communications, Inc.
26750 Agoura Road
Calabasas, CA 91302
USA

SPIRENT

Chapter 1
Introduction

In this chapter...

Product Overview

Tcl Automation allows users to control and automate Abacus™ 5000 Windows-based software from a remote computer running Windows® or a different operating system (OS). Through the Tcl application program interface (API), a remote user can apply automation scripts to run Abacus 5000 software and perform tests with Abacus hardware. The provided Tcl Automation API is potentially cross-platform because of freely distributed Tcl packages compatible with other operating systems (OS), including Linux/Unix and SUN Solaris. Thus, Tcl Automation can be easily modified to be installed on other operating systems.

The following Abacus systems are supported:

- Abacus 5000 IP Telephony Migration Test System with SC3 System Controller and any of its circuit generators (CGs)
- Abacus 50 Analog Test System
- Abacus 50 T1/E1 Test System
- Abacus 100 Analog Subscriber Generator

Throughout this manual, the term system controller (SC) is used to refer to the serial and local area network (LAN) interface any of the supported Abacus systems.

Tcl Automation is potentially a universal cross-platform solution because of the availability of Tcl packages compatible with a variety of operating systems, including:

- AIX® (IBM Advanced Interactive EXecutive, proprietary Unix®)
- HP-UX® (Hewlett-Packard UniX, proprietary Unix®)
- Linux® (open-source Unix-like OS)
- Macintosh® OS X (Apple Computer)
- Solaris™ (Sun Microsystems™)
- Windows® ME, 2000, XP, Vista® Business (Microsoft®)

Abacus 5000 Tcl automation utilizes the ActiveState (a division of Sophos) ActiveTcl® package and documentation available from:

http://www.activestate.com/Products/ActiveTcl/

As of the publication date of this document, the standard version of Active Tcl and applicable documentation were available for download at no charge.

Important:
- ActiveTcl version 8.4.13 or later is supported.
- For 8.4.x, “INSTALL_DIR/TCL/APIClasses” and “INSTALL_DIR/TCL” are both OK for system variable TCLLIBPATH.
- For 8.5.x, TCLLIBPATH can only include “INSTALL_DIR/TCL”.
- If version of ActiveTcl is less than 8.5.3, you also need to copy directory “INSTALL_DIR/TCL/itcl3.4” into “TCL_ROOT_DIR/lib” manually.

System requirements, additional documentation, and links to programming tool packages are included with the Tcl descriptions at the ActiveState website, above.

Tcl API main features are:

- It covers the majority of Abacus software functionality.
- It operates with familiar Abacus logical entities, such as Application, Test, System Information, Report, Protocol Selection, Partition and Timing, Results, Codecs, System under Test, and Path Confirmation.
- It provides the capability to retrieve the current configuration of any logical entity instance, modify or store instance properties on the client side, and reapply the obtained configuration.
- It provides an interface in object-oriented style.
- It provides remote invocation of methods with desired parameters on Abacus Software and indicates the received result.
- It uses XML notation to represent internal states of instances, method parameters, and responses.
- It provides calls to obtain and handle the last error.

High-Level Architecture

Tcl Automation provides two types of Tcl classes: API classes (ApiXXX) and data holder classes (DatXXX). API classes provide high-level Tcl API for communication with Abacus SCs and for processing any configuration data. API classes and their methods always correspond with Abacus Software entities. Object-oriented applications typically have a number of classes containing fields with methods for *setting* and methods for *getting* the values. API classes use data holder class instances as input (setting) parameters and method results. (Data holder classes will sometimes be referred to as *dump data holders*.

Data holder classes represent almost all settings that could be obtained and configured with Abacus 5000 software. Also, Tcl Automation contains some subsidiary procedures for connection and interaction with instances of Abacus 5000 software over a network. These methods make up the Communication Library.

The Abacus Server Unit (SU) is an intermediary between the Communication Library and an Abacus Software instance. It receives a call from the client-side Communication Library in XML format, parses the string, and uses the Abacus software COM interface to invoke the method with the obtained name and specified parameters.

Important: The COM interface is internal, not to be confused with the serial port or COM port of the Abacus hardware.

The result, packed in an XML-marked-up string, is delivered over the network to the calling API class method as a response.

The Abacus SU runs as a Windows service and launches a single Abacus Software instance for every Tcl Automation application on the same Windows-driven machine. This instance handles one or more Abacus SCs as usual. (Abacus software typically supports a large number of Abacus SCs simultaneously.)

The high-level Tcl Automation product architecture is illustrated in *Figure 1-1 on page 20*.

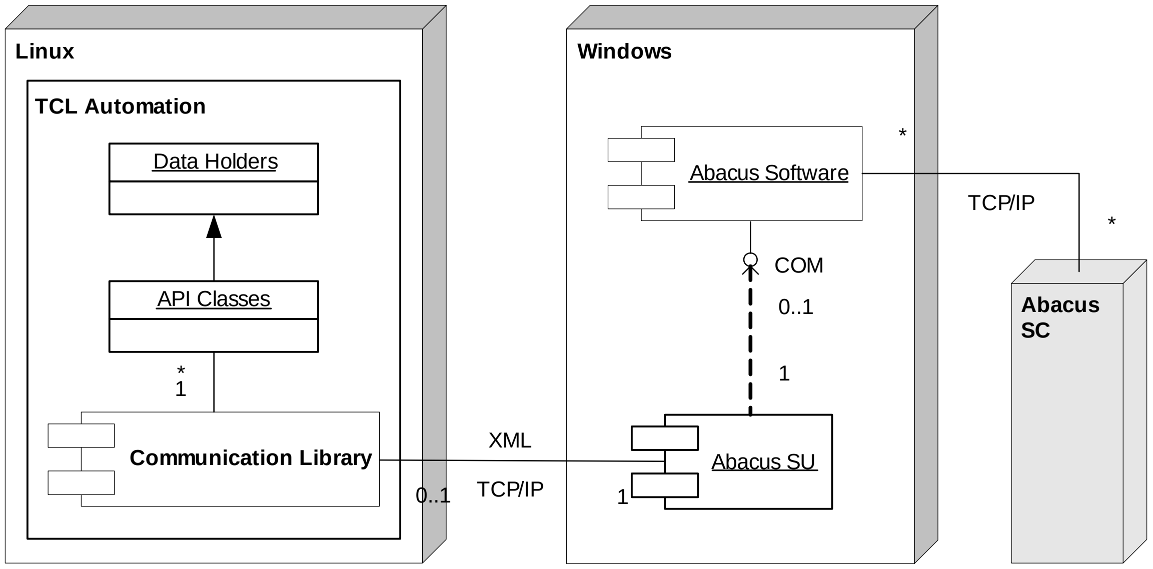


Figure 1-1. TCL Automation Architecture

API Classes

API classes are defined in detail in *Chapter 3, "API Reference."*

Data Holder Classes

Data holders serve as containers for storage of configuration data. One data holder can contain child data holders that represent some subsets of configuration data. Thus, data holders must be carefully initialized.

If a data holder class contains only variable fields, their values can be set and retrieved explicitly by using the **DatXXX::SetValue** and **DatXXX::GetValue** methods, respectively. All data holder classes provide **DatXXX::ImportFromXMLString** methods for object initialization from an XML-marked-up string, and **DatXXX::ExportToXMLString** methods for the serialization of an object inner state to an XML-marked-up string.

If the data holder class contains other child data holders or variables, you can retrieve them by compound name. Typically, however, data holders contain sequences of some data holder class instances. Such data holders are like lists. In this case, use the subsidiary **FindInObjList** procedure to retrieve the child data holder by an attribute, using the parent data holder list, the attribute name, and the attribute value as the parameters. Use the **DatXXX::GetList_xxx** method of a parent data holder instance to obtain the corresponding object list. The procedure returns the index of the specified data holder instance in the list, and you can retrieve this instance with the **DatXXX::GetByIndex_xxx** method, using the obtained index as the parameter.

Chapter 2

Getting Started

This chapter provides preliminary procedures required to use Abacus Automation and introduces script composition.

Tcl Automation is potentially a universal cross-platform solution. Tcl Automation Client deployment for Linux platform is described here.

In this chapter...

Prerequisites

Before you can start using Tcl Automation, you must install the required software on your Abacus host computer and on all computers that will be used to control Abacus remotely.

Abacus Host Computer

Abacus software must be installed on a PC running Microsoft Windows XP Professional (32-bit), operating at a minimum processing speed of 1.7 GHz (a processing speed of 3.0 GHz is preferable), or on Microsoft Vista® Business. CPU speed and RAM requirements increase when one PC is running one or more instances of Abacus software supporting multiple system controllers (SCs).

Abacus Automation Server Unit (Abacus SU)

During Abacus software installation, Abacus Automation Server Unit (Abacus SU) files are copied into the folder <Abacus root> BIN. However, the service is not installed or started. This must be done manually, as described in the next procedure.

To install the Abacus Automation Server Unit (**SU**)**:**

Note: Install Abacus software by following the installation procedure provided in the *Abacus Hardware Manual* (Chapter 5) received with your Abacus hardware.

1 Run the Abacus software at least once to register as a COM object.

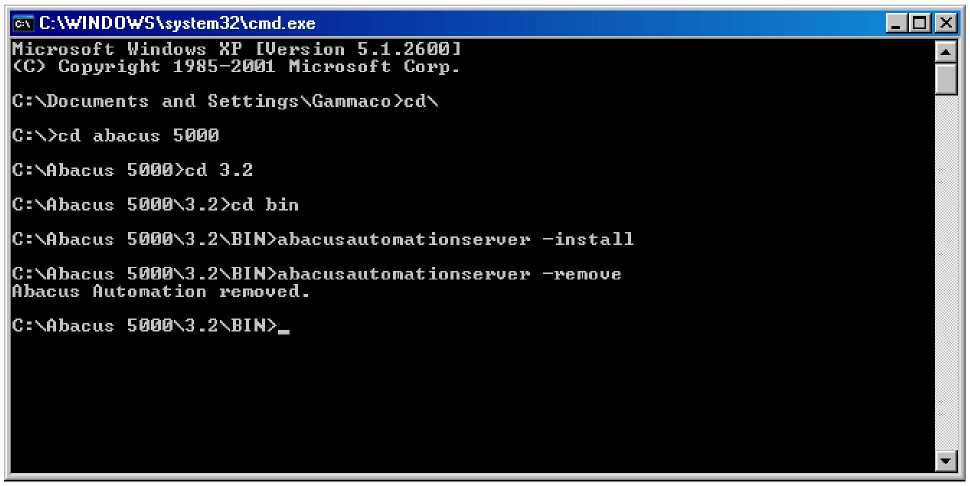
2 Install the Abacus SU:

Tip: See *page 26* for the 3 initial steps if you use Windows Vista.

a Use the Windows **Run** command (**Start** | **Run**) to open a **Run** window.

b In the **Open** field, type `cmd`.

c Click **Enter** or press `<Enter>` to open a DOS command window (*Figure 2-1*).



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Gammaco>cd\

C:\>cd abacus 5000

C:\Abacus 5000>cd 3.2

C:\Abacus 5000\3.2>cd bin

C:\Abacus 5000\3.2\BIN>abacusautomationserver -install

C:\Abacus 5000\3.2\BIN>abacusautomationserver -remove
Abacus Automation removed.

C:\Abacus 5000\3.2\BIN>_
```

Figure 2-1. DOS Command Window

d On the command line, enter `cd\ <Enter>` to change to your root drive.

e At the `C:\>` prompt, type `cd dir <Enter>`, where **`dir`** is the name of the directory (folder) containing the Abacus software (generally Abacus 5000).

f At the `C:\> Abacus 5000` prompt, change to the directory of the software version you are using. (In *Figure 2-1 on page 24*, this is version 3.2.)

g At the next prompt, type `cd BIN <Enter>` to change to the BIN directory.

h Type the following command:

```
AbacusAutomationServer.exe -install <Enter>
```

If the command is successful, DOS will respond with `Abacus Automation started`, followed by a prompt.

i If the response is not seen, the Abacus Automation Service might already be running. In that case, type:

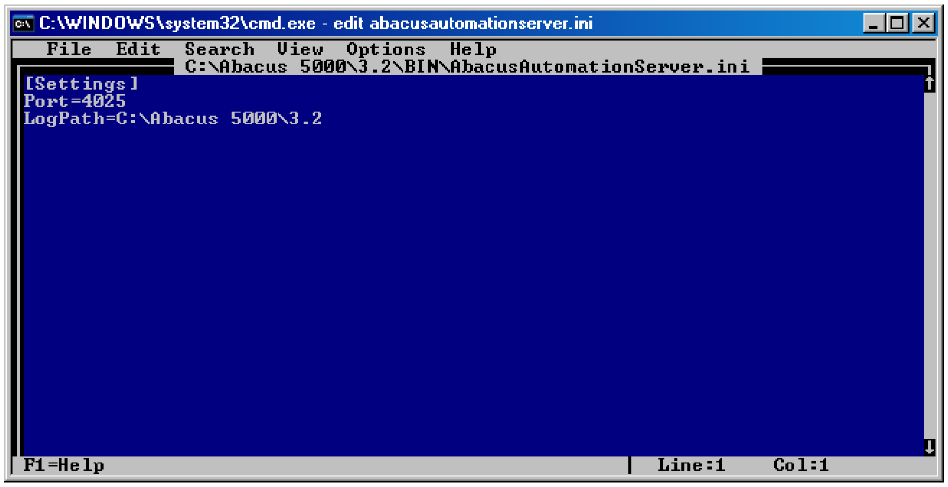
```
AbacusAutomationServer.exe -remove <Enter>
```

j If this response `Abacus Automation Removed` is seen, the service was already running and has now been removed. To activate the service, type the following command:

AbacusAutomationServer.exe -install `<Enter>`

If the command is successful, DOS will respond with `Abacus Automation started`, followed by a prompt.

k The Abacus Automation Service is controlled by a file within the BIN directory: *AbacusAutomationServer.ini*. You can manually edit this at any time using a text editor (such as Windows Notepad or Wordpad), or edit the ini (initialization) file in the DOS command window by invoking the *edit* function (*Figure 2-2*):


C:\WINDOWS\system32\cmd.exe - edit abacusautomationserver.ini
File Edit Search View Options Help
C:\Abacus 5000\3.2\BIN\AbacusAutomationServer.ini
F1=Help
Line:1 Col:1

```
[Settings]
Port=4025
LogPath=C:\Abacus 5000\3.2
```

Figure 2-2. DOS Edit Window

l At the prompt `C:\Abacus 5000\Version\BIN` (*Figure 2-1 on page 24*), type: `edit abacusautomationserver.ini <Enter>`.

m In the DOS Edit window (*Figure 2-2 on page 25*), you can modify the IP Port setting from its default value of 4025, or change the logging path to a different directory. The default is to place the log files directly in the BIN directory.

n To save the new values, select **File | Save As** and click on the filename.

o To abandon the changes, select **File | Close** or **File | Exit**. In either case, you will be given the opportunity to save any changes you made before closing the edit window.

p If you modified the LogPath to a new subdirectory within the BIN directory, create the new subdirectory. In the following example, the LogPath was changed from the default to `C:\Abacus5000\3.2\abacusautomationlog`:

- At the prompt `C:\Abacus 5000\Version\BIN` (*Figure 2-1*), type:

  ```
  md abacusautomationlog <Enter>
  ```

  to make the new directory

- Verify that the new directory was created by typing `dir <Enter>` at the prompt `C:\Abacus 5000\Version\BIN`. If the new subdirectory was created correctly. you should see:

  ```
  <DIR> abacusautomationlog
  ```

q Close the command window.

For Windows Vista replace steps *Step a* - *Step c* (on *page 24*) with the following steps:

a Click **Start** button

b Select **All Programs > Accessories**. Right-click on **Command Prompt** shortcut. Select **run as administrator** in the context menu

c Click **Continue** if UAC prompt window appears

Note: TCL client files are copied into the folder <Abacus root>\TCL. There are two subdirectories in TCL: **APIClasses** and **Samples**. **APIClasses** contains the Abacus Automation API lib for TCL. **Samples** contains samples for using Abacus Automation in TCL.

Important: The Abacus Automation Service remains installed until it is removed. If you run the Abacus Automation Service, always stop and remove the service before uninstalling Abacus 5000 by using the command:

```
AbacusAutomationServer.exe -remove
```

Remote Computers

Before using Abacus Automation, you must install Tcl with Tk toolkit, tDOM,and Itcl packages on each remote computer that will be used to control Abacus operations. Installation instructions for different types of operating systems are available at:

http://aspn.activestate.com/ASPN/docs/ActiveTcl/at.install.html#install

After you have downloaded Tcl, you must configure it to use with Abacus Automation. The following example provides instructions for installing Tcl on a Linux-based machine.

Install Tcl on a Linux-based machine as follows:

1. Log in as *root*.
2. Copy the *tcl-install.tar* installation file to any folder.
3. Go to this folder.
4. Run the tar `xf *.tar` command. Ignore any warnings.
5. Run the `./install.sh` command. This installs Tcl/Tk with tdom Itcl and tcl-readyapi packages.
6. Edit config.tcl by setting the AUTOMATION_SERVER_PORT value to match the port specified for *AbacusAutomationServer.ini* in *Step k on page 25*.

Tcl Automation is now ready for use. You can use the sample scripts provided in *“::aba::Api Classes and Features Examples” on page 187* as-is, edit these scripts, or write your own scripts.

Namespace

A namespace is a way to identify a group of items, such as procedures and variables, used within a program as well as to distinguish them from other items having the same name but residing in different namespaces. The items within a namespace must have unique names. Namespaces prevent collisions between different libraries that use the same sets of variables. By adding a prefix, such as *Utils*, to procedures and variable names, you can group all items belonging to the same library. To create a namespace, separate the name from its corresponding procedure or variable with a double colon (::), for example, MyNamespace::MyVar.

Every namespace can contain another namespace within it and can be nested hierarchically. A nested namespace is encapsulated inside its parent namespace and cannot interfere with other namespaces. To access a member of a nested namespace, you must specify the path within its namespace hierarchy (for example, Utils::Graphics::Rect). Variables, commands, and namespaces declared outside of any namespace are considered to be the members of the predefined global namespace. You can access these items by providing an empty prefix (for example, ::aba).

You can create several namespaces at the same hierarchical level. Declarations belonging to the same namespace can be placed in several different files. In the following example, the namespace MyNamespaceA spans two files: FileX.tcl and FileY.tcl.

FileX.tcl

```
namespace eval MyNamespaceA {
  proc A
}
namespace eval MyNamespaceB {
  proc B
}
```

FileY.tcl

```
namespace eval MyNamespaceA {
  proc C
}
```

To access a same-level namespace:

```
namespace eval MyNamespaceA {
  namespace eval MyNamespaceB1 {
    itcl::class Bclass {
    }
  proc…
  }
}
namespace eval MyNamespaceB2 {
  proc UseB1 { } {
    MyNamespaceB1::Bclass b   #accessing a class from differ-
ent same-level namespace
    }
  proc…
  }
}

MyNamespaceA::MyNamespaceB1::Bclass x  #accessing the same
class from the global namespace
```

Abacus aba Namespace

Abacus Tcl Automation classes and procedures use the prefix **::aba::**, so the TCL method for creating an *ApiApplication* instance for Abacus would look as follows:

```
::aba::ApiApplication app
```

Another example:

```
::aba::WriteToFile "_App_Enter.txt"
"\n**************************\n"

app Enter localhost
```

Writing the First Script

In this section, you will learn how to compose and run a simple Abacus Tcl script. This tutorial is presented in the form of answers to the most frequently asked “How to…” questions.

The common Tcl script includes the following actions:

1. Establish a connection with the Abacus SU.
2. Load the environment (optional).
3. Establish connections with the Abacus SCs (optional).
4. Acquire cards (optional).
5. Configure Protocol Selection (optional).
6. Configure Partition and Timing (optional).
7. Configure test duration parameters (optional).
8. Start the test.
9. Check test status.
10. Stop the test.
11. Get and save results into a text report.
12. Break the connection with the Abacus SU.

Figure 2-3 on page 30 shows the corresponding User Activity diagram.

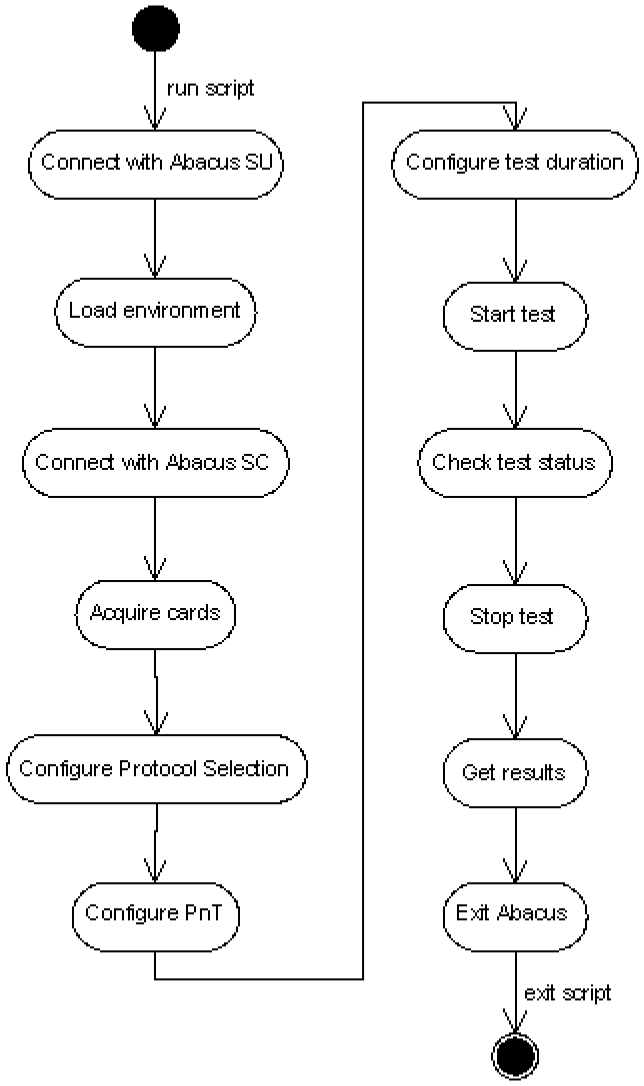


Figure 2-3. Tcl-Automation User Activity Diagram

How Do I Start Writing a Script?

Before writing the first script you should learn Tcl as a language. Documentation is available online at:

http://aspn.activestate.com/ASPN/docs/TclDevKit

A script can be composed in any text editor, such as vi, EMAC, or Windows Notepad. This procedure describes the basic steps in creating a script to configure an Abacus environment and run a test. The procedures that follow provide details on different parts of the script.

Create a script as follows:

1. At the beginning of the script, include this declaration:

   ```
   package require tclreadyapi
   ```

   This package provides you with everything necessary to work with an Abacus SC remotely and perform Abacus tests. It also supports object-oriented constructions over basic Tcl and provides the ability to manipulate XML-marked-up strings to pass parameters and receive data remotely. Thus, your work will look like creating instances of API classes and their methods calls with the desired parameters.

   If this error appears - "can't find package tclreadyapi", there are three ways to resolve this issue.

   a. Set system variable TCLLIBPATH (recommended): Create system variable TCLLIBPATH, and include "INSTALL_DIR/TCL" in it.
   b. Copy all directories "INSTALL_DIR/TCL/*" to "TCL_ROOT_DIR/lib".
   c. Add Tcl code "lappend ::auto_path INSTALL_DIR/TCL" before "package require tclreadyapi".

   If this error appears - “can't find package Itcl”, and your ActiveTcl version is 8.5.x, then you need to manually copy the directory “INSTALL_DIR/TCL/itcl3.4” into “TCL_ROOT_DIR/lib”.

2. Following the Tcl declaration, create the **::aba::ApiApplication** instance:

   ```
   ::aba::ApiApplication app
   ```

   This class represents the Abacus Software instance as a whole and allows you to connect to the Abacus SU, load and save environments, and recognize whether the Abacus SU is ready to serve the Abacus Client Unit (Abacus CU) of your Tcl Automation application. The methods defined in the **::aba::ApiApplication** class are described in *“::aba::ApiApplication Methods” on page 65*.

3. Try to establish a connection with the Abacus Software, using the **::aba::ApiApplication::Enter** method (*“::aba::ApiApplication::Enter” on page 65*). This procedure assumes that the Abacus SU is located on the same computer with the Abacus CU of the Tcl Automation application. Therefore, the “localhost” string is used as the IP parameter.

   The **::aba::ApiApplication::Enter** method returns **1** if the connection to the Abacus SU was successful, or **0** if the connection was unsuccessful. You can check this value and write a warning to a specified file. Use the subsidiary **::aba::WriteToFile** procedure and enter the output filename and the warning string as the first and the second parameters correspondingly. There is another subsidiary **::aba::WriteToFileEx** procedure for file output. The only difference is that it places the current timestamp before the second parameter string in the output file. See *“How Do I Handle Errors?” on page 46* for other error handling methods.

```
if { 0 == [app Enter localhost] } {
  ::aba::WriteToFile "_test_.txt" "Unable to establish con-
nection"
  exit
}
```

4. If you want to configure test properties by loading an Abacus environment, use the **::aba::ApiApplication::Load** method and enter the environment name as the parameter.

   ```
   app Load "Demo_A5K_SIP.env"
   ```

5. Use this subsidiary procedure to suspend script execution for a time span in seconds:

   ```
   ::aba::sleep 10
   ```

   This is necessary because some operations of the Abacus SC use shared resources, so that the Abacus SU may not be ready to supply your further queries. Use this instruction after each time-consuming command, such loading an environment or running a test. (Refer to *“Sleep Procedure” on page 68* for more information.)

6. Modify and reapply the existing environment configuration, or create a new configuration. Each “How Do I” section that follows this provides instructions for configuring a different portion of the environment or test setup. Details on each command used are provided in *“::aba::Api Classes Summary Table” on page 55*.

7. Use the symmetric **::aba::ApiApplication::Save** method to save the current environment. Enter the environment name as the parameter:

   ```
   app Save "SYSTEM.env"
   ```

8. End the script by closing the Abacus Software instance and disconnecting the Abacus SU. Use the **::aba::ApiApplication::Exit** method as follows:

   ```
   app Exit
   ```

9. Close the script with the exit operator to end script execution:

   ```
   exit
   ```

How Do I Manage a Connection to an Abacus System Controller?

Before you can acquire cards for your test, you must establish a connection with at least one Abacus SC. This capability is supported by the **::aba::ApiSysInfo::SetConnection** method.

Use the ::aba::ApiSysInfo::SetConnection method as follows:

1 Create the **::aba::ApiApplication** instance:

```
::aba::ApiSystemInformation sysinfo
```

2 Set the IP address of the available Abacus SC you want to connect to, using the Set-Connection method:

```
SetConnection { IPAddress Password }
```

   a For `IP address`, use the IP address of the available Abacus SC; for example:

   ```
   set ip "192.168.10.53"
   ```

   For HypermetricAP, the IP address should be `IP:SLOT`; for example:

   ```
   set ip "192.168.10.53:1"
   ```

   b For this example, use an empty string as the password parameter:

   ```
   sysinfo SetConnection $ip ""
   ```

3 At any time, you can obtain the current connection configuration and write it to a file using the **::aba::ApiSysInfo::GetConnectionString** method. This method returns the names of connecting SCs and their IP addresses:

```
sysinfo GetConnectionString
```

4 Connections to all SCs are automatically removed when script execution ends, but you can also remove connection(s) to SC(s) by using one of the following methods:

- To remove the connection to a specific SC:

  ```
  sysinfo RemoveConnection $ip
  ```

- To remove the connection to *all* connected SCs:

  ```
  sysinfo RemoveAllConnections
  ```

How Do I Manage Cards?

Before starting a test, all cards required for the test must be acquired and set up.

The **::aba::ApiSystemInformation::AddCard** method is used to add a card. The type of card added is determined by its physical slot number and the IP address of the SC that controls the card:

```
set addcard1 [sysinfo AddCard $ip 1]
set addcard2 [sysinfo AddCard $ip 2]
```

Abacus 5000 will automatically assign a *logical slot number* to the card when it is added.

Note: Logical slot numbers are assigned to cards sequentially as they are acquired. This logical slot number is used when accessing the card.

How Do I Manage Protocol Selection?

You need to change the protocol configuration to perform the various protocol tests supported by the Abacus software. The **::aba::ApiProtocolSelection** class defines methods used for protocol selection.

There are five basic entities to be set, corresponding to panels within the **Protocol Selection** window in the Abacus User Interface:

- **Cards**
- **Channels**
- **ICG** (for the ICG card type only)
- **VoIP** (for the ICG card type only)
- **SUT**

There are five common steps in editing protocol selection parameters, as described in the following paragraphs.

Step 1: View Current Protocol Configuration

The first step is to obtain information on the current protocol configuration and save it to an output file. This is done via the **::aba::ApiProtocolSelection** controller class. The **:::aba::ApiProtocolSelection::GetCurrentConfig** method returns an XML-marked-up string compatible with the **::aba::DatProtocolSelection** data holder class.

To save the current configuration to an output file, include the following lines in your script:

```
::aba::ApiProtocolSelection ps_api
set res [ps_api GetCurrentConfig]
::aba::WriteToFile "_ps_.txt" $res
```

The output file saved as `_ps_.txt` will contain the whole current configuration, marked up with XML.

Step 2: Configure Protocol Selection Parameters

You will use a data holder class to configure protocol selection. Data holder classes are introduced in *"Data Holder Classes" on page 20*.

Start configuring parameters by importing XML data into a **::aba::DatProtocolSelection** data holder object:

```
::aba::DatProtocolselection ps_data
ps_data ImportFromXMLString $res
```

The following example shows a change of signaling type for the first set. You can change other parameters the same way.

```
set idx [::aba::FindInObjList [ps_data.card
GetList_slotcardspec] "slot" 1]
::aba::WriteToFile "_ps_.txt" "Index for slot 1 is $idx"
```

Further operations are available only if the index corresponds an existing logical slot.

Step 3: Set Parameters

To get the current variable value, use the **::aba::DatXXX::GetValue** method:

```
if {$idx != -1} {
    set slotcardspec [ps_data.card GetByIndex_slotcardspec $idx]
    $slotcardspec.signaling GetValue
```

For the desired parameter (e.g. signaling), call the **::aba::DatXXX::SetValue** method. It is defined for every data holder class and is similar to the standard Tcl setvalue procedure.

```
$slotcardspec.signaling SetValue "SKINNY"
::aba::WriteToFile "_ps_.txt" "New signaling:"
::aba::WriteToFile "_ps_.txt" [$slotcardspec.signaling
GetValue]
```

Step 4: Export Object

Export the modified **::aba::DatProtocolselection** data holder object into an XML-marked-up string.

Step 5: Reapply New Parameters

Finally, reapply the new parameters via the controller class:

```
    set new_conf [ps_data ExportToXMLString]
}
    ps_api SetCurrentConfig $new_conf
```

How Do I Manage Partition and Timing?

Methods for setting partition and timing parameters are defined in the **::aba::ApiPnT** class.

```
::aba::ApiPnT pnt
```

The **::aba::ApiPnT::GetCurrentConfig** method is used to retrieve the current partition and timing configuration.

Note: Unlike **::aba::ApiProtocolSelection::GetCurrentConfig** and **::aba::ApiSUT::GetCurrent Config**, with **::aba::ApiPnT::GetCurrentConfig** you must enter the parameters (card type as a string and card side as a string).

```
set sipsub [pnt GetCurrentConfig "ICG_MGCP" "Subscriber"]
```

You use the **::aba::DatSetList** data holder class to set partition and timing parameters, importing configuration data into its instances. The operation is the same as in previously described examples. This example shows the change of “from” number:

```
DatSetlist setlist
setlist ImportFromXMLString $sipsub
set idx [FindInObjList [setlist GetList_set] "num" 2]
if { $idx != -1} {
  set tmpset [setlist GetByIndex_set $idx]
  ::aba::WriteToFile "_pnt_.txt" [$tmpset ExportToXMLString]
```

For the desired parameter (e.g. setassociation-from, num) call the **::aba::DatXXX::SetValue** method. It is defined for every data holder class and is similar to the standard Tcl setvalue procedure.

```
$tmpset.setassociation.from SetValue 34
$tmpset.num SetValue 2
```

Export the data holder object value to XML by calling the **::aba::DatXXX::ExportToXMLString**.

```
pnt UpdateSet [$tmpset ExportToXMLString]
  ::aba::WriteToFile "_pnt_.txt" [$tmpset ExportToXMLString]
}
```

How Do I Write a Configuration to a File?

The configuration of every Abacus logical entity can be saved to a file. Every configuration can be represented as an XML-marked-up string or as the inner state of the appropriate data holder object. Therefore, two methods are provided.

- You can get the current configuration string by using the **::aba::ApiXXX::GetCurrent Config** method. This method is available for the **::aba::ApiCodecs**, **::aba::ApiProtocolSelection**, **::aba::ApiSUT**, and **::aba::ApiPnT** classes. Then just write the configuration string to a file using the **::aba::WriteToFile** procedure.

  ```
  ::aba::ApiCodecs codecs
  set xmlstring [codecs GetCurrentConfig]
  ::aba::WriteToFile "_codecs_.txt" $xmlstring
  ```

- Use the **::aba::DatXXX::SaveToFile** method. Enter a filename as the parameter. This method is available for every data holder class and writes its inner state into the file with XML markup. In fact, it calls the **::aba::WriteToFile** procedure, including.

  ```
  ::aba::DatCodecsconfiguration cfg
  cfg ImportFromXMLString $xmlstring
  cfg SaveToFile "codec1.txt"
  ```

Which method you use is your choice.

Note: You must have the data holder object initialized to use the **::aba::DatXXX::SaveToFile** method. Only the configuration string is necessary when you use the **::aba::WriteToFile** procedure.

How Do I Manage Test Execution?

First, define the **::aba::ApiTest** instance. This definition allows you to use **::aba::ApiTest** methods to configure test duration parameters, and to start and stop a test.

```
::aba::ApiTest test
```

To set test duration parameters, use the **::aba::DatTestduration** data holder class.

```
::aba::DatTestduration duration

duration.type SetValue "Interval"

duration.minutes SetValue 1
```

Modifying a data holder object does not mean that a new configuration is automatically applied to an Abacus Software Instance. The user is responsible for updating any configuration. The **::aba::ApiTest::SetDuration** method reapplies a **::aba::DatTestDuration** data holder instance to the Abacus Software and changes the current test configuration.

```
test SetDuration [duration ExportToXMLString]
```

Note: You must call the **ExportToXMLString** data holder method before entering the **::aba::DatTestDuration** object as the parameter. All Tcl methods take only string parameters, but a data holder has a complex structure; therefore, it is converted to an XML-marked-up string.

Run your test with the **::aba::ApiTest::Start** method.

```
test Start

sleep 10
```

The subsidiary **::aba::sleep** method is placed here because some operations of the Abacus SC use shared resources, so that the Abacus SU may not be ready to supply your further queries. (Refer to *"Sleep Procedure" on page 68* for more information.)

While a test is running, you can not reconfigure test parameters, but you can check whether it is still running. Use the **::aba::ApiTest::GetStatus** method. Refer to *"::aba::ApiTest::GetStatus" on page 154* for a list of the responses you may receive.

```
test GetStatus

sleep 200

test GetStatus
```

If a finite test duration is set, you need not stop the test explicitly. To stop the test explicitly, do one of the following:

- Use the **::aba::ApiTest::Stop** method without parameters to stop the test gracefully.

- Use the **::aba::ApiTest::StopNow** method to stop the test immediately.

  ```
  test Stop
  ```

How Do I Get and Store Results?

Test results are obtained with the **::aba::ApiResults::GetResults** method. First, you should configure your request. In your request, state which parameters to return.

```
::aba::DatRge RGE
```

Common request structure is described in *Appendix A, “RGE Configuration Interface”*.

This is an example of a request for results:

```
::aba::DatSection SectionTestStatus
::aba::DatSection SectionChannelSelection
::aba::DatSection SectionMeasurementCounts
::aba::DatSectionParametersParam Param1
::aba::DatSectionParametersParam Param2

RGE.Section.type SetValue "root"
RGE.Section.name SetValue "Root"

SectionTestStatus.type SetValue "test-status-info"
SectionTestStatus.name SetValue "Test Status Information"

RGE.Section AddItem_Section SectionTestStatus

SectionChannelSelection.type SetValue "channel-selection"
SectionChannelSelection.name SetValue "Channel Selection"
SectionChannelSelection.ChannelSelection.selectionmethod
SetValue "All"
SectionChannelSelection.ChannelSelection.includeoriginate
SetValue "YES"
SectionChannelSelection.ChannelSelection.includeterminate
SetValue "YES"
SectionChannelSelection.ChannelSelection.averaged SetValue "NO"

RGE.Section AddItem_Section SectionChannelSelection

SectionMeasurementCounts.type SetValue "measurement-counts"
SectionMeasurementCounts.name SetValue "Measurement Counts"
```

```
Param1.name SetValue "hide-vc-output-config"
Param1 SetValue "YES"
Param2.name SetValue "hide-vc-details"
Param2 SetValue "YES"

SectionMeasurementCounts.Parameters AddItem_Param Param1
SectionMeasurementCounts.Parameters AddItem_Param Param2

RGE.Section AddItem_Section SectionMeasurementCounts

::aba::WriteToFile "_rge_.txt" "RGE request:"
::aba::WriteToFile "_rge_.txt" [RGE ExportToXMLString]

::aba::ApiTest test

::aba::WriteToFile "_rge_.txt" "Starting test..."
::aba::WriteToFile "_rge_.txt" [test Start]
::aba::WriteToFile "_rge_.txt" "Run test for some time..."
::aba::sleep 10

test StopNow

::aba::ApiResults res

::aba::WriteToFile "_rge_.txt" "RGE response:"
set response [res GetResults [RGE ExportToXMLString]]

::aba::DatReport Report
```

Results specified in the request are obtained as an XML string that is to be exported to the data holder class.

```
Report ImportFromXMLString $response
::aba::WriteToFile "_rge_.txt" [Report ExportToXMLString]
app Exit
```

If you want to get information only on certain events, use the **::aba::ApiResults::GetEvents Count** method:

```
set eventsamount [results GetEventsCount]
::aba::WriteToFileEx "_test_.txt" "Amount of events: $event-
samount"
```

Following this, call the **::aba::ApiResults::GetEvent** method with a parameter of event number (the last one in this example) to get a string with event parameters.

```
if { $eventsamount > 0 } {
  set outvalue [results GetEvent [expr $eventsamount - 1]]
  ::aba::WriteToFileEx "_test_.txt" "Last event: $outvalue"
}
```

To get information only on Variances, use the **::aba::ApiResults::GetVariances** method.

```
set variances [results GetVariances 0 1]
::aba::CheckAndContinue $variances "ERROR" app
::aba::WriteToFileEx "_test1_.txt" $variances
```

How Do I Generate a Report Using a Template?

You can generate reports formatted as PDF, HTML, or XML either after a test has been started, or after the completion of a test. A complete description of the Report Generation Engine (RGE) configuration parameters for the Abacus COM server is provided in *Appendix A, “RGE Configuration Interface.”*

Sample scripts which access a report template containing a default set of section and parameter configuration definitions are provided for you in the installation directory of Abacus 5000 software: <installation dir path>\Abacus 5000\version number\TCL\Samples\. For example:

C: \Abacus 5000\5.00\TCL\Samples\

The template used by the sample scripts has the same report configuration as the Abacus 5000 GUI *Generate Reports* dialog default. To see the template, you can use the sample script: **_Rge_Template.tcl** to do the following:

```
package require tclreadyapi

set OUTFILE "_Rge_Template.txt"
::aba::WriteToFile $OUTFILE
"\n*********************************************\n"

set template [::aba::GetRGETemplate]
::aba::WriteToFile $OUTFILE "RGE template:"
::aba::WriteToFile $OUTFILE $template

::aba::WriteToFile $OUTFILE "\nSCRIPT DONE"
exit
```

The sample script: **_Reports_SetRGEConfig.tcl** shows you how to set your report configuration to the template configuration. The principal lines of code are shown in this excerpt of the sample script (refer to *“::aba::GetRGETemplate” on page 103*, and *“::aba::ApiReports::SetRGEConfig” on page 104*):

```
::aba::ApiReports report
report SetRGEConfig [::aba::GetRGETemplate]
```

To customize the template by adding, removing or changing sections and/or parameters before generating your report, you can use the approach presented in the sample script, **_RGE_Update**. This script shows an example of how to change the report name and a parameter within the channel-selection section using the ::aba::DatRGE::GetByIndex_xxx method. This method is described in detail in *“::aba::DatXXX::GetByIndex_xxx { idx }” on page 280*.

Here in this excerpt from **_RGE_Update.tcl**, you can see how the template is used to set the report format, and then the report name is set to a value in quotes, “UpdatedReport Name”:

```
set template [::aba::GetRGETemplate]
::aba::WriteToFile $OUTFILE "Original RGE template:"
::aba::WriteToFile $OUTFILE $template

::aba::DatRge RGE
RGE ImportFromXMLString $template
# customize report-file-name-root
set reportFileNameParam [RGE.Section.Parameters
GetByIndex_Param 3]
$reportFileNameParam SetValue "UpdatedReportName"
::aba::WriteToFile $OUTFILE [$reportFileNameParam Export-
ToXMLString]
```

Finally, to produce the report, the configuration must be generated (refer to *“::aba::ApiReports::Generate” on page 106*) and a report must be downloaded (*“::aba::ApiReports::Download” on page 106*). The sample script: **_Reports_Generate.tcl** includes examples of all of the steps required to produce a report using the methods described above.

How Do I Manage Data Holders?

Data holders are a set of classes that serve as containers for storage of configuration data. One data holder can contain child data holders (as fields of the class) that represent some subsets of configuration data. Parent-child relationships can be traced easily on the corresponding data holder class XSD-schema.

Thus, data holders must be carefully initialized. If a data holder class contains a variable field (although such field is a data holder itself it does not have any child data holders), its value can be set and retrieved explicitly by using the **::aba::DatXXX::SetValue** and

::aba::DatXXX::GetValue methods, respectively. For example, look through the following **::aba::DatTestduration** data holder class definition:

```
itcl::class ::aba::DatTestduration {
  inherit ::aba::Dat
  public variable days
  public variable hours
  public variable minutes
  public variable date
  public variable time
  public variable scriptsperchannel
  public variable type
  constructor {}
  method IsInitialized {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

::aba::DatTestduration::days, hours, minutes, date, time and **type** are variable string fields (wrapped with the **::aba::DatString** data holder class). **::aba::DatTestduration::Scriptsperchannel** field is also a variable and does not have any child data holders, since it is of the **::aba::DatXstestdurationenum** data holder class (enumeration). So the following sample code is correct:

```
::aba::DatTestduration duration
duration.type SetValue "Interval"
duration.minutes SetValue 1
::aba::WriteToFile "_test_.txt" [duration ExportToXMLString]
```

An XML-marked-up string is a universal representation of the data holder object inner state. Tcl language defines only string type, so all parameters, as well as result values, must be strings. Also, such notation is useful for performing remote method calls over TCP/IP network protocol. Therefore, all data holder classes provide **::aba::DatXXX::ImportFrom XMLString** methods for object initialization from an XML-marked-up string and the **::aba::DatXXX::ExportToXMLString** methods for the serialization of an object inner state to an XML-marked-up string. Look through the sample above again for use of these methods.

If the data holder class contains any child data holder, it can be either a single entity (**::aba::DatString** data holder class instance) or a composite entity (a potentially unbounded sequence of some data holder class instances). In the first case, you can get such a field by the compound name. In the second case, such an entity is like an object list.

For example, the definition of the **::aba::DatProtocolselectioncard** data holder class is as follows:

```
itcl::class ::aba::DatProtocolselectionCard {
  inherit ::aba::Dat
  public variable slotcardspec
  constructor {}
  method IsInitialized {}
  method GetByIndex_slotcardspec { idx }
  method AddItem_slotcardspec { NewItem }
  method RemoveItem_slotcardspec { ItemIndex }
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Here the **::aba::DatProtocolselectioncard::slotcardspec** field is an object list of string (wrapped in the **::aba::DatString** data holder) variables with the given card slot parameters values. The object list field is mapped as a sequence element on the appropriate data holder class XSD-schema:

```
<xs:element name="card" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="slot-card-spec" type="slot-parameters"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In this case, use the subsidiary **::aba::FindInObjList** procedure to get the child data holder from the list by its attributes (one or many), passing the list variable and the sequence of attribute names with the attribute values as the parameters. All data holder features are used by the following sample code:

```
DatProtocolselection ps_data
ps_data ImportFromXMLString $res
```

Use the **::aba::DatXXX::GetList_xxx** method of the parent data holder instance to obtain the corresponding object list.

```
set idx [::aba::FindInObjList [ps_data.card
GetList_slotcardspec] "slot" 1]
```

::aba::DatProtocolselection::card is the field of the **::aba::DatProtocolselectioncard** data holder class described above.

The **::aba::FindInObjList** procedure returns the index of the first data holder instance that matches the query and omits all others (if any). You can get the returned instance with the **::aba::DatXXX::GetByIndex_xxx** method by passing the obtained index as the parameter. Note that only one **::aba::DatXXX::GetList_xxx** and one **::aba::DatXXX::GetByIndex_xxx** method is defined for each list.

Returned index **idx** equals '-1' if there are no elements in the list with the specified attribute value (method call is unsuccessful).

```
if {$idx != -1} {
  set slotcardspec [ps_data.card GetByIndex_slotcardspec $idx]
  $slotcardspec.signaling GetValue
  $slotcardspec.signaling SetValue "SKINNY"
  ::aba::WriteToFile "_ps_.txt" "New signaling:"
  ::aba::WriteToFile "_ps_.txt" [$slotcardspec.signaling
GetValue]
  set new_conf [ps_data ExportToXMLString]
}
ps_api SetCurrentConfig $new_conf
```

Sometimes it is necessary to compose a data holder object by hand, without getting its configuration from the Abacus software. The **::aba::DatXXX::AddItem_xxx** and **::aba::DatXXX::RemoveItem_xxx** methods are defined for every object list field of any data holder class and provide such a possibility. Pass the configured data holder object as a new element to the first method and as a deleted element to the second. The following code is an example:

```
   ::aba::DatSection SectionTestStatus
   RGE.Section.type SetValue "root"
   RGE.Section.name SetValue "Root"
   SectionTestStatus.type SetValue "test-status-info"
   SectionTestStatus.name SetValue "Test Status Information"
   RGE.Section AddItem_Section SectionTestStatus
```

How Do I Run Multiple Instances of the Abacus UI?

In earlier versions of Abacus Tcl Automation, only one connection from a TCL client to an Abacus TCL automation server was allowed. Among other things, it meant that it was impossible to have several Abacus UI instances running on the same machine. This restriction was eliminated in version 4.10. Currently, a TCL client can access up to four instances of Abacus UI on one Windows-operated PC or any number of Abacus UI instances running on different PCs. Also one Abacus Automation server can now be controlled by up to four TCL clients.

To use more than one instance of Abacus UI, you must create an **::aba::ApiApplication** object for each instance. Then each **::aba::ApiApplication** object can be used in the same way as it was possible in the earlier version of Abacus TCL automation support. The previously created scripts will work without any changes.

For all other API classes (besides **::aba::ApiApplication)**, a new method was introduced to make it possible to select a connected instance to be associated with. This new method **SetChannel** receives as a single parameter a connection channel identifier: this is the value returned by the **::aba::ApiApplication::Enter** method upon establishing a new connection. If the channel wasn't set explicitly, an API class instance will use the last Abacus UI instance that was opened.

To use the multiple instance support feature, do not import any new packages or libraries into a TCL script.

To create a single script running two Abacus UI instances:

1 Create two **::aba::ApiApplication** objects:

```
::aba::ApiApplication app1
::aba::ApiApplication app2
```

(In other scenarios, more than two **::aba::**ApiApplication objects can be created).

2 Specify the actual IP address of the Abacus Automation server. This example assumes that the Abacus SU is located on the same computer as the Abacus CU of the Tcl Automation application. For this reason, the "localhost" string is used as the IP parameter.

```
set ip localhost
```

3 Establish communication channels with the Abacus software, using the **::aba::Api-Application::Enter** method (*“::aba::ApiApplication::Enter” on page 65*). This method will automatically associate ApiApplication instances with corresponding communication channels:

```
set chan1 [app1 Enter $ip]
set chan2 [app2 Enter $ip]
```

4 Create a test object and associate it with the first communication channel using the **::aba::ApiTest::SetChannel** method:

```
::aba::ApiTest test
test SetChannel $chan1
```

5 Create a PnT object and associate it to the second communication channel using the **::aba::ApiPnT::SetChannel** method:

```
::aba::ApiPnT pnt
pnt SetChannel $chan2
```

Once you have followed the above procedure, both instances will run simultaneously.

For an extended example on multiple instances support, refer to *“::aba::Api Classes and Features Examples” on page 187*.

How Do I Handle Errors?

When you call a method, there are three possibilities:

- The method returns a successful result.
- An error occurs and the method returns an unsuccessful result.
- An internal error occurs and the method call fails. Any error message can be returned as the result.

The result of a Tcl method depends on the case.

Every Api-prefixed class provides the **::aba::ApiXXX::GetLastError** method and holds the last called method name. If the method call is unsuccessful (“ERROR”, “False”, or “0” strings are returned), the error message is generated and stored. When you call the **::aba::ApiXXX::Get LastError** method, the last error message is returned.

```
app GetLastError
```

One of the ways to handle errors is the subsidiary **::aba::CheckIfFailed** procedure. The procedure allows you to compare a method result with a presumed successful result. If the strings are not equal, the **::aba::CheckIfFailed** procedure displays an error message box, closes the Abacus Software instance, and stops the script execution. The similar **::aba::CheckAnd Continue** procedure displays an error message box and asks the user whether the Abacus Software instance should be closed and the script execution should be stopped. If the parameter strings are equal, script execution could continue.

```
set loadenvresult [app Load "Demo_A5K_MEGACO.env"]
::aba::CheckIfFailed $loadenvresult "OK" app
::aba::WriteToFileEx "_test_.txt" "Load environment: $load-
envresult"
```

You must decide which method of handling errors works best for you. It is strongly recommended that you use the **::aba::CheckIfFailed** or **::aba::CheckAndContinue** procedure after every method call that can end unsuccessfully. Use the **::aba::CheckIfFailed** procedure if the presumed error is unrecoverable and the **::aba::CheckAndContinue** procedure if you manage to recover.

Analog/SIP Phone Book

The Abacus 5000 GUI provides the ability to create an Analog or SIP phone book in XML format. The GUI also allows you to import and export the phone book from and to an external XML file. Before release 4.20, the Tcl API only supported GET/SET of phone book XML for SIP. Now, the API provides GET/SET methods and data holder classes to create, import and export phone book text**,** for both Analog and SIP. Other Tcl API GET/SET (*Address Book*) methods also support both Analog and SIP.

Note: The methods **::aba::ApiPnT::GetAddressBook** and **::aba::ApiPnT::SetAddressBook** are deprecated. Please use the GetPhoneBookBySet, SetPhoneBookBySet, and data holder classes ::aba::DatEndpoint and ::aba::DatEPConfig4Set which support PhoneBook Txt. Refer to *"PhoneBook Txt" on page 49*.

Definitions

Endpoint Database (EPDB)

EPDB is a table where each row defines parameters of one Analog/SIP endpoint. Each column in this database represents a particular parameter. Abacus stores the EPDB at <abacus_root>/PHONES/Abacus.epdb.

Endpoint

An *endpoint* is a set of parameters that defines the generic phone number of a channel.

PhoneBook XML

PhoneBook XML is the external representation of the EPDB data. Its XML schema is defined by PhoneBookType.

Note: Phone book XML is deprecated in favor of Phonebook Txt. Refer to *"PhoneBook Txt" on page 49*.

The schema for PhoneBookType is as follows:

```
<xs:complexType name="AnalogAddressType">
  <xs:sequence>
   <xs:element name="address" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="phone" minOccurs="0">
       <xs:complexType>
        <xs:simpleContent>
         <xs:extension base="xs:long">
```

```
                  <xs:attribute name="increment" type="xs:int" use="optional" />
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
          <xs:element name="cid-phone" minOccurs="0">
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base="xs:long">
                  <xs:attribute name="increment" type="xs:int" use="optional" />
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
          <xs:element name="cid-name" minOccurs="0">
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base="xs:string">
                  <xs:attribute name="increment" type="xs:int" use="optional" />
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="repetitions" type="xs:int" use="optional" />
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="PhoneBookType">
  <xs:sequence>
    <xs:choice minOccurs="0">
      <xs:element name="voip-address" type="VoIPAddressType"/>
      <xs:element name="analog-endpoints" type="AnalogAddressType" />
    </xs:choice>
    <xs:element name="apply-to" type="XS_PhoneApplyToEnum" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="profile" type="xs:string"/>
  <xs:attribute name="type" type="XS_TypeEnum"/>
  <xs:attribute name="side" type="XS_SideEnum"/>
  <xs:attribute name="phtype" type="XS_PhoneNoTypesEnum"/>
</xs:complexType>
```

PhoneBook Txt

PhoneBook Txt is an external representation of the EPDB data as a simple INI file. PhoneBook Txt is compatible with ExportToText or ImportFromText of the data holder class ::aba::DatEPConfig4Set.(Refer to *"::aba::DatEPConfig4Set" on page 235*.)

Support for Contiguous or Non-contiguous Sets

The format of PhoneBook Txt is a typical *.ini file, with each row of EPDB stored as an *.ini section. Because there is no channel information specified in a section, the Abacus GUI kernel maps the section to a particular channel of a set. This way, the PhoneBook Txt file format can facilitate the phone book requirement for a contiguous or non-contiguous set.

For example, given a contiguous set with the range 1-3, the Abacus GUI maps section 1 to channel 1, section 2 to channel 2 and section 3 to channel 3. In another example, given the range of the set is 4, 9-10, the Abacus GUI maps section 1 to channel 4, section 2 to channel 9 and section 3 to channel 10.

Example File

The following is an example of PhoneBook Txt for a set of three endpoints. In it, there are three sections with labels: [1], [2] and [3]. Note that the labels have to be contiguous integer numbers. Each section contains configuration parameters for one endpoint.

In the case where the set contains more channels than the number of sections in the PhoneBook Txt, the last section will be assigned to the rest of the channels.

```
[1]
User Name=Joe4
IP v4=10.2.16.11
Domain IPv4=10.2.16.1
Gateway IPv4=10.2.16.1
Subnet Mask IPv4=255.255.0.0

[2]
User Name=Joe9
IP v4=10.2.16.11
Domain IPv4=10.2.16.1
Gateway IPv4=10.2.16.1
Subnet Mask IPv4=255.255.0.0
```

```
[3]
User Name=Joe12
IP v4=10.2.16.11
Domain IPv4=10.2.16.1
Gateway IPv4=10.2.16.1
Subnet Mask IPv4=255.255.0.0
```

Related API Classes

API Classes Used with PhoneBook Txt

The Tcl API provides ::aba::ApiEPDB::GetPhoneBookBySet and ::aba::ApiEPDB::SetPhoneBookBySet to support get and set of **PhoneBook Txt** for Analog and SIP. For details, refer to:

- *“::aba::ApiEPDB::GetPhoneBookBySet” on page 80*
- *“::aba::ApiEPDB::SetPhoneBookBySet” on page 81*

API Classes Used with PhoneBook XML (deprecated)

API classes: ::aba::ApiPnT::GetAddressBook and ::aba::ApiPnT::SetAddressBook support get and set of **PhoneBook XML** for Analog and SIP. For details, refer to:

- *“::aba::ApiPnT::GetAddressBook” on page 92*
- *“::aba::ApiPnT::SetAddressBook” on page 93*

Related Data Holder Classes

Data Holder Classes Used with PhoneBook Txt

The Tcl API provides data holder classes ::aba::DatEndpoint and ::aba::DatEPConfig4Set to support **PhoneBook Txt**.

Note: These data holder classes have no knowledge of the data types of column parameters and thus do not perform validation on submitted PhoneBook data. Instead, they depend on the Abacus GUI to validate the PhoneBook Txt data at environment compilation time.

For further details, refer to:

- *“::aba::DatEndpoint Data Holder Class” on page 233*
- *“::aba::DatEPConfig4Set” on page 235*

Data Holder Classes Used with PhoneBook XML (deprecated)

The data holder class ::aba::DatSetlistitemtype contains optional data holder class ::aba::DatSetphones, which is used to specify the phone numbers for a set. ::aba::DatSetphones deals solely with PhoneBook XML.

If ::aba::DatSetlistitemtype is used to create or update a set, the phone numbers for the set have to be specified in PhoneBook XML. However, you have the option to create or update a set without specifying phone numbers. Use a separate call to ::aba::ApiEPDB::SetPhoneBookBySet (*"::aba::ApiEPDB::SetPhoneBookBySet" on page 81*) to set the endpoints of the set using the new PhoneBook Txt.

Data holder classes ::aba::ApiPnT::GetAddressBook and ::aba::ApiPnT::SetAddressBook support Analog/SIP PhoneBook XML only. For details, refer to:

- *"::aba::ApiPnT::GetAddressBook" on page 92*
- *"::aba::ApiPnT::SetAddressBook" on page 93*

Chapter 3
API Reference

In this chapter...

API Classes Diagram

Figure 3-1 shows a diagram of the API Classes.

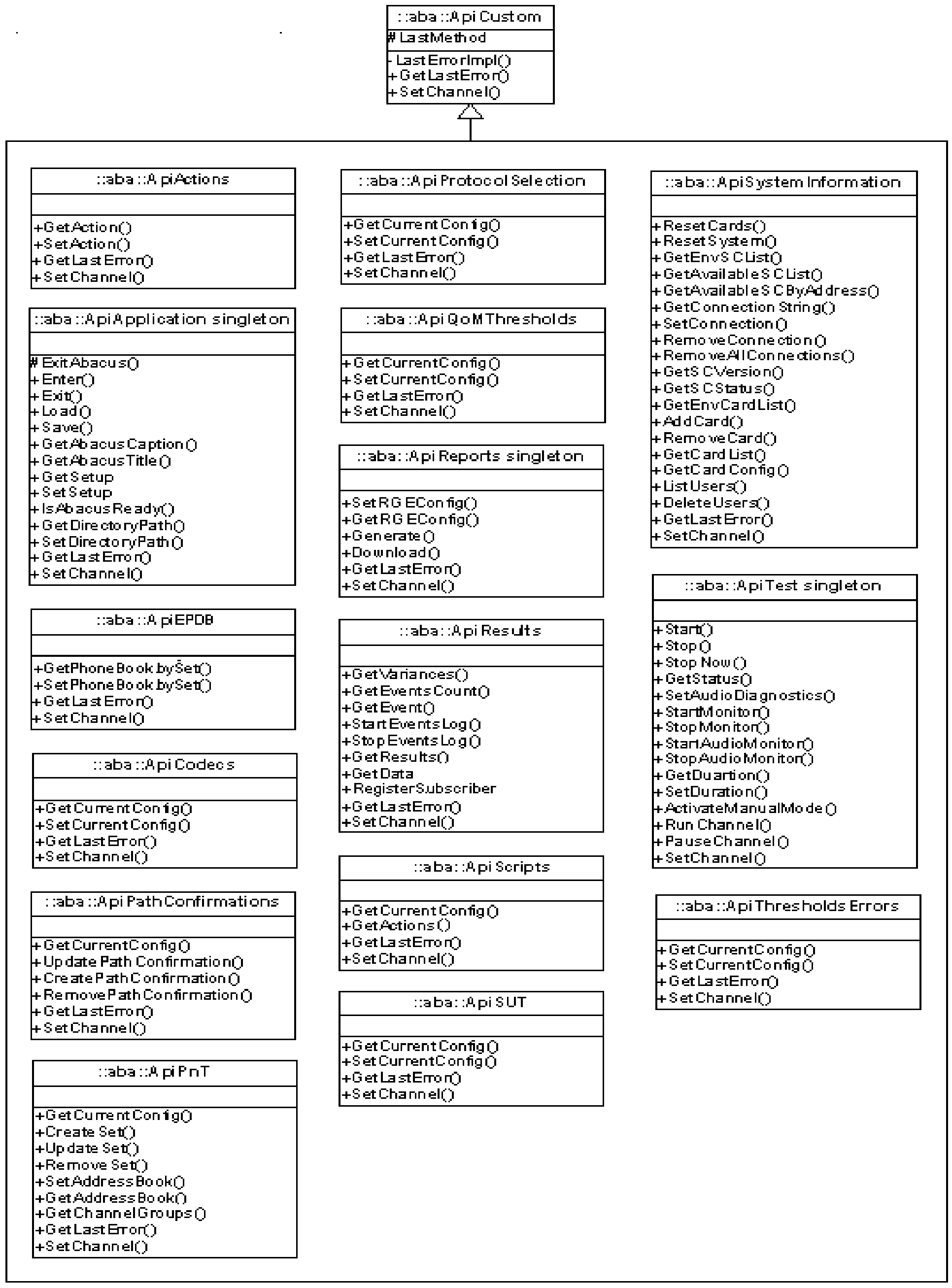


Figure 3-1. API Classes Class Diagram

::aba::Api Classes Summary Table

Table 3-1 lists the ::aba::Api classes and methods with a short description of each. Details about each of these methods are provided later in the chapter.

Table 3-1. ::aba::Api Classes Summary Table

Class	Methods	Description
::aba::ApiActions		
	GetAction {ActionName}	Get the call sequence action and associated tasks and settings.
	SetAction {XMLData}	Write the call sequence action and associated tasks and settings.
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {CommChannel}	Associate API class instance with Abacus UI instance
::aba::ApiApplication		
	Enter {IP Timeout}	Connect with the SU and create an Abacus Software instance
	Exit {}	Terminate tests, close connection with system controller, close Abacus UI, and break connection with the SU
	Load {EnvName}	Load specified environment from disk to the current Abacus Software instance
	Save {EnvName}	Save current configuration under specified environment name
	GetAbacusCaption {}	Get the caption of the Abacus main window
	GetAbacusTitle {}	Get the title of the Abacus Software application
	GetSetup {}	Get the global setup parameters
	SetSetup {XML Param}	Update global setup parameters

Table 3-1. ::aba::Api Classes Summary Table (continued)

Class	Methods	Description
	IsAbacusReady {}	Query the Abacus software instance to check if it is ready to communicate with the Tcl Automation application
	GetDirectoryPath {Directory}	Get the path of specified Abacus directory
	SetDirectoryPath {Directory NewName}	Set an Abacus directory name and path
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {CommChannel}	*Undefined. Use **Enter** method.*
::aba::ApiCodecs		
	GetCurrentConfig {}	Return the current Codec configuration
	SetCurrentConfig {XMLData}	Set a new Codec configuration
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {NewChannel}	Associate API class instance with Abacus UI instance
::aba::ApiEPDB		
	GetPhoneBookBySet {CardType Side SetNo Meth}	Return the endpoint addresses of a given set in text, serialized as a string, from current Abacus environment
	SetPhoneBookBySet {CardType Side SetNo Meth PhoneBook}	Set the endpoint addresses of a given set in text, serialized as a string, from current Abacus environment
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {CommChannel}	Associate API class instance with Abacus UI instance
::aba::ApiPathConfirmations		

Table 3-1. ::aba::Api Classes Summary Table (continued)

Class	Methods	Description
	GetCurrentConfig {}	Return current Path Confirmation profiles
	UpdatePathConfirmation { XMLData}	Update a Path Confirmation profile
	CreatePathConfirmation { XMLData}	Create a new path confirmation profile
	RemovePathConfirmation { Name }	Delete a specified path confirmation
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {NewChannel}	Associate API class instance with Abacus UI instance
::aba::ApiPnT		
	GetCurrentConfig {CardType Side}	Return Partition and Timing configuration for the current test
	CreateSet {XMLData}	Create a new set of channels with the specified parameters
	UpdateSet {ParamRecXML}	Update an existing channel set with the specified parameters
	RemoveSet {CardType Side SetNo}	Remove a specified set of channels
	GetAddressBook {CardType Side SetNumber}	Retrieve address book for a specified set (**DEPRECATED**)
	SetAddressBook {BookParameters}	Create new address book and save under a designated name (**DEPRECATED**)
	GetChannelGroups {}	Return circuit groups of current set
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {NewChannel}	Associate API class instance with Abacus UI instance
::aba::ApiProtocolSelection		

Table 3-1. ::aba::Api Classes Summary Table (continued)

Class	Methods	Description
	GetCurrentConfig {}	Return the current Protocol Selection configuration
	SetCurrentConfig { XMLData}	Set a new Protocol Selection configuration
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {NewChannel}	Associate API class instance with Abacus UI instance
::aba::ApiQoMThresholds		
	GetCurrentConfig {CardType}	Return the current PESQ and PSQM threshold settings
	SetCurrentConfig {ParamRecXML}	Configure the PSQM and PESQ thresholds
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {CommChannel}	Associate API class instance with Abacus UI instance
::aba::ApiReports		
	GetRGETemplate {}	Retrieve a template of a default XML report configuration. This method is a common method belongs to "::aba::"
	GetRGEConfig {}	Retrieve existing report configuration
	SetRGEConfig {XMLData}	Set up parameters for generation of a report in current Abacus Software instance
	Download {ClientPath ChunkSize}	Save generated report to disk in a specified folder
	GetLastError {}	Return the last error that occurred in last called method

Table 3-1. ::aba::Api Classes Summary Table (continued)

Class	Methods	Description
	SetChannel {NewChannel}	Associate API class instance with Abacus UI instance
::aba:: ApiResults		
	MOTSOTGetData{SubscriberID}	Return result records as specified by the subscriber since the last data retrieval
	MOTSOTSubscribe{XMLParam}	Request specified MOT/SOT result from Abacus COM interface during test setup
	GetVariances {VrType OutputOpt}	Return variances that have non-zero count values
	GetEventsCount {}	Return the total number of events that occurred during the test
	GetEvent {number}	Return information on a specified event number
	StartEventsLog {FileName}	Write the events generated from the current test into specified file
	StopEventsLog {}	Stop logging events
	GetResults {ResParameters}	*Replaced by ApiReports methods.*
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {NewChannel}	Associate API class instance with Abacus UI instance
::aba::ApiScripts		
	GetCurrentConfig {}	Return the current script settings
	GetActions {ScriptName}	Return list of actions for specified script
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {CommChannel}	Associate API class instance with Abacus UI instance

Table 3-1. ::aba::Api Classes Summary Table (continued)

Class	Methods	Description
::aba::ApiSUT		
	GetCurrentConfig {}	Return the current SUT configuration
	SetCurrentConfig {ParamRecXML}	Set a new SUT configuration
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {NewChannel}	Associate API class instance with Abacus UI instance
::aba::ApiSystemInformation		
	ResetCards {}	Reset all acquired subsystems
	ResetSystem {}	Reboot all connected SCs
	GetEnvSCList {}	Return list of system controllers defined in the active environment
	GetAvailableSCList {}	Return list of all available SCs
	GetAvailableSCByAddress{Address}	Return list of attributes for SC at specified IP address
	GetConnectionString {}	Return a string with current connection configuration
	SetConnection {IPAddress Password Timeout}	Connect to SC at specified IP address
	RemoveConnection {IP}	Terminate connection to SC at specified IP address
	RemoveAllConnections {}	Terminate connections to all SCs
	GetSCVersion {IPAddress}	Return software version installed on SC at specified IP address
	GetSCStatus {IPAddress}	Return status of SC at specified IPaddress
	GetEnvCardList {IPAddress}	Return list of acquired subsystems
	AddCard {IP Slot}	Acquire subsystem at specified IP address and physical slot

Table 3-1. ::aba::Api Classes Summary Table (continued)

Class	Methods	Description
	RemoveCard {SlotNo}	Remove subsystem in specified slot
	GetCardList {}	Return list of all subsystems acquired in connected systems
	GetCardConfig {Slot}	Return configuration of subsystem in specified slot
	ListUsers {}	Return list of users logged onto SCs connected to the current Abacus Software instance
	DeleteUsers {UserName}	Delete specified user
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {NewChannel}	Associate API class instance with Abacus UI instance
::aba::ApiTest		
	Start {XMLParam}	Start (or restart) test with the current configuration
	Stop {XMLParam}	Gracefully stop test
	StopNow {}	Stop test immediately
	GetStatus {}	Return current test status
	SetAudioDiagnostics {bON Port cType cSide Chan Dir Volume }	Configure audio portion of Media Monitor
	StartAudioMonitor {cType cSide Channel Dir}	Capture audio stream on specified channel and output to files
	StopAudioMonitor {cType cSide Channel Dir}	Stop capturing audio
	StartMonitor {MonitorType A_B cType cSide Channel FileName }	Start Data Link Monitor, Line Protocol Monitor, or VoIP Signaling Monitor
	StopMonitor {MonitorType A_B cType cSide Channel}	Stop Data Link Monitor, Line Protocol Monitor, or VoIP Signaling Monitor

Table 3-1. ::aba::Api Classes Summary Table (continued)

Class	Methods	Description
	ActivateManualMode {ONOFF}	Set Manual mode to either ON or OFF to control initial channel state
	RunChannel {XMLData}	Start specified channels
	GetDuration {}	Retrieve test duration
	SetDuration {ParamRecXML}	Set test duration type and duration
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {NewChannel}	Associate API class instance with Abacus UI instance
::aba::ApiThresholdsErrors		
	GetCurrentConfig {}	Return the current threshold and error settings
	SetCurrentConfig { ParamRecXML }	Set threshold and error configuration
	GetLastError {}	Return the last error that occurred in last called method
	SetChannel {NewChannel}	Associate API class instance with Abacus UI instance
::aba::ApiResultRecords		
	GetRowCount {TableID}	Retrieve number of records in the result table.
	GetRowData {TableID RowIdx}	Retrieve XML string with list of fields of a particular record in the result table.
::aba:: ApiFileTransfer		
	GetFileSize { FolderID FileName }	Return size of specified file
	Download { FolderID FileName ClientPath ChunkSize }	Download specified file from server local client path.

::aba::Api Classes Reference

Note: If you are using version 4.10 or later of the Abacus software, you must prefix all classes and procedures with “::aba::” to avoid conflicts with other automation APIs. (Refer to *“Namespace” on page 27* for more information.)

::aba::ApiActions Methods

::aba::ApiActions provides Abacus call sequence action management.

::aba::ApiCustom

::aba::ApiActions

::aba::ApiActions::GetAction

Purpose

GetAction requests the action and associated BEFORE, HOOK, WAIT, SEND, DO and AFTER settings.

Command

```
::aba::ApiActions::GetAction { ActionName }
```

Parameters

`ActionName`

The name of the action as a string.

Response

Returns the set of six action elements (tasks) on success, or an empty string on failure. A detailed error description can be retrieved by calling the GetLastError method.

Example

```
::aba::ApiActions act
::aba::DatAction dat

set name "A calls B, Pulse, confirms for Call Length"
dat ImportFromXMLString [act GetAction $name]
```

::aba::ApiActions::SetAction

Purpose

SetAction writes the action and associated BEFORE, HOOK, WAIT, SEND, DO and AFTER settings.

Note: If applied XML does not include all six tasks or all task settings, Abacus will preserve the current configuration (obtained from previous SetAction call or loaded ENV) for those settings not written by the SetAction command.

Command

```
::aba::ApiActions::SetAction { XMLData }
```

Parameters

`XMLData`
This is a string in XML format (compatible with the **::aba::DatAction** data holder) that defines new parameters for the action configuration. For a detailed description of the data holder class, refer to the *Tcl Automation on Abacus Data Holders Reference Guide.*

Response

Returns "FALSE" on failure. A detailed error description can be retrieved by calling the GetLastError method.

Example

```
::aba::ApiActions act


set XMLString "<action name=\" A calls B, Pulse, confirms for
Call Length\">
<task name=\"AFTER\" mode=\"NOTHING\" /></action>"


act SetAction $XMLString
```

::aba::ApiActions::GetCurrentConfig

Purpose

GetCurrentConfig returns the current Actions configuration.

Command

```
::aba::ApiActions::GetCurrentConfig  {  }
```

Parameters

None

Return Value

This command returns a string in XML format (compatible with the **::aba::DatAction-slist** data holder class) with data on the current configuration if it was successful, or an empty string if it wasunsuccessful. For a detailed description, refer to the *Abacus Tcl Automation Data Holders Reference.*

Example

```
::aba::ApiActions  actions


set  result  [actions GetCurrentConfig  ]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiApplication Methods

::aba::ApiApplication provides Abacus Software instance management.

::aba::ApiCustom

::aba::ApiApplication

::aba::ApiApplication::Enter

Purpose

Enter establishes a connection with the Abacus SU (server unit) and creates a corresponding Abacus Software instance. This method is always required to start interaction with an Abacus Software instance and perform tests and must be included at the beginning of a script.

Command

```
::aba::ApiApplication::Enter { IP Timeout}
```

Parameters

`IP`
The IP address of the device operating as the Abacus SU you want to connect to.

`Timeout`
This string designates the timeout value when connecting to the Abacus SU. This parameter is optional; its default value is 0, which means infinite.

Response

The command returns the socket handle if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiApplication application
...
set result [application Enter $IP]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
```

::aba::ApiApplication::Exit

Purpose

Exit terminates any tests currently running, closes connectivity with the system controller, closes the Abacus UI, and breaks the connection with the Abacus SU. This method is always required after all operations with the Abacus Software instance are complete, and must be included at the end of a script.

Command

```
::aba::ApiApplication::Exit { }
```

Parameters

None

Response

None

Example

```
...
::aba::ApiApplication application
...
set result [application Exit ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiApplication::Load

Purpose

Load loads a specified environment from disk to the current Abacus Software instance.

This method should be followed by an **::aba::sleep** procedure, as described in *"Sleep Procedure" on page 68*.

Command

```
::aba::ApiApplication::Load { EnvName }
```

Parameters

`EnvName`

The absolute or relative path of the environment to be loaded on the computer from which the Abacus Software instance was launched.

Note: Always enter the path name string in braces; e.g:

```
app Load {c:\some folder\myenv.env}
```

Response

The command returns **OK** if the file was loaded successfully, or an error message if the file did not exist or if it could not be read.

Example

```
...
::aba::ApiApplication application
```

```
...
set result [application Load $EnvName]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

Sleep Procedure

```
sleep <time>
```

This subsidiary procedure suspends script execution for a time span in seconds. This is necessary because some operations of the Abacus SC use shared resources, and therefore the Abacus server may not be ready to supply further queries. Use this instruction after each time-consuming command, such as loading an environment or running a test.

::aba::ApiApplication::Save

Purpose

Save saves the current configuration under a specified environment name.

Command

```
::aba::ApiApplication::Save { EnvName }
```

Parameters

`EnvName`

The absolute or relative path of the environment to be saved to the computer from which the Abacus Software instance was launched.

Note: Always enter the path name string in braces; e.g:

```
app Load {c:\some folder\myenv.env}
```

Response

The command returns **OK** if the operation was successful, or an error message if it was unsuccessful.

Example

```
...
::aba::ApiApplication application
...
```

```
set result [application Save $EnvName]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiApplication::GetAbacusCaption

Purpose

GetAbacusCaption requests the caption of the Abacus main window.

Command

```
::aba::ApiApplication::GetAbacusCaption { }
```

Parameters

None

Response

Returns the requested caption on success, or an empty string on failure.

Example

```
...
::aba::ApiApplication application
...
set result [application GetAbacusCaption ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiApplication::GetAbacusTitle

Purpose

GetAbacusTitle requests the title of the Abacus Software application.

Command

```
::aba::ApiApplication::GetAbacusTitle { }
```

Parameters

None

Response

Returns the requested title on success, or an empty string on failure. The title is returned in the format:

- Software version number
- Name of loaded environment
- Name of the loaded results (if exists)
- Asterisk (*) character if the environment configuration was modified

Example

```
...
::aba::ApiApplication application
...
set result [application GetAbacusTitle ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiApplication::GetSetup

Purpose

GetSetup requests the global setup information from the Abacus Software application (accessed through the UI by **File** | **Setup**).

Command

```
::aba::ApiApplication::GetSetup { }
```

Parameters

None

Response

XML document that conforms to the **::aba::DatSetup** data holder class. For a detailed description, refer to *“::aba::DatSetup Data Holder Class” on page 265.*

Example

```
::aba::ApiApplication  application

set  result  [application  GetSetup  ]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiApplication::SetSetup

Purpose

SetSetup updates the setup information in the Abacus Software application (accessed through the UI by **File** | **Setup**).

Command

```
::aba::ApiApplication::SetSetup { XML Param }
```

Parameters

XML parameter document that conforms to the same dataholder class as the Response of *GetSetup*. For a detailed description, refer to *“::aba::DatSetup Data Holder Class” on page 265.*

Response

The command returns a value of **1** if the operation was successful; **0** if it was unsuccessful.

Example

```
::aba::ApiApplication  application

#  Configuring  dataholder.
::aba::DatSetup  appsetup

set  result  [application  SetCurrentConfig[appsetup
ExportToXMLString]]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiApplication::IsAbacusReady

Purpose

Is Abacus Ready queries the Abacus software instance whether it is ready to communicate with the Tcl Automation application.

This method is frequently used because some operations of the Abacus SC use shared resources and thus an Abacus Software instance may not be ready to supply further queries. If this happens, use the subsidiary **::aba::sleep** procedure described in *"Sleep Procedure" on page 68*.

Command

```
::aba::ApiApplication::isAbacusReady { }
```

Parameters

None

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiApplication application
...
set result [application IsAbacusReady ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiApplication::GetDirectoryPath

Purpose

GetDirectoryPath returns the path to one of the Abacus directories.

Command

```
::aba::ApiApplication::GetDirectoryPath { Directory }
```

Parameters

`Directory`
This string represents one of the Abacus directories. Valid directory strings include:

“Actions”	“Phones”
“Audio”	“Protocols”
“Batch“	“Results”
“Environment”	“Scripts”
“Images”	“Sounds”
“PathConf”	“Templates”

Response

This command returns the absolute path of the requested directory type on the computer from which the Abacus Software instance was launched.

Example

```
...
::aba::ApiApplication application
...
set result [application GetDirectoryPath $Directory]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiApplication::SetDirectoryPath

Purpose

SetDirectoryPath specifies a new path to one of the Abacus directories.

Command

```
::aba::ApiApplication::SetDirectoryPath { Directory NewName }
```

Parameters

`Directory`
This string represents one of the Abacus directories. Valid directory strings include:

- “Actions”*
- “Audio”
- “Batch“*
- “Environment”
- “Images”
- “PathConf”*
- “Phones”*
- “Protocols”*
- “Results”
- “Scripts”*
- “Sounds”*
- “Templates”

* Directory name is user-defineable but must reside under Abacus root.

`NewName`
New absolute or relative path of the selected directory on the computer from which the Abacus Software instance was launched.

Note: Always enter the path name string in braces; e.g:

```
set NewName {C:\Abacus 5000}
```

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiApplication application
...
set result [application SetDirectoryPath $Directory $NewName]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiApplication::GetLastError

::aba::ApiApplication::GetLastError inherits the **::aba::ApiCustom::GetLastError** method.

Purpose

GetLastError displays the text of the last error that occurred in the last called method of the current class instance.

Command

```
::aba::ApiApplication::GetLastError { }
```

Parameters

None

Response

The command returns the text string of the last error that occurred, or the string "$none" if no errors occurred.

Example

```
...
::aba::ApiApplication application
...
set result [application GetLastError ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiApplication::SetChannel

Purpose

SetChannel associates the API class instance with an Abacus UI instance.

Important: Behavior of this method is undefined for **::aba::ApiApplication**. The only way to associate **::aba::ApiApplication** with an Abacus UI instance is to use its **Enter** method. For more information refer to *"How Do I Run Multiple Instances of the Abacus UI?" on page 44*.

Command

```
::aba::ApiApplication::SetChannel { }
```

Parameters

`None.`

Response

None.

Example

None.

::aba::ApiCodecs Methods

Note: If you are using version 4.10 or later of the Abacus software, you must prefix all classes and procedures with "::aba::" to avoid conflicts with other automation APIs. (Refer to *"Namespace" on page 27* for more information.)

::aba::ApiCodecs provides Audio/Video codecs for test data stream management.

::aba::ApiCustom

::aba::ApiCodecs

::aba::ApiCodecs::GetCurrentConfig

Purpose

GetCurrentConfig returns the current Codec configuration.

Command

```
::aba::ApiCodecs::GetCurrentConfig { }
```

Parameters

None

Response

This command returns a string in XML format (compatible with the **::aba::DatCodec-sconfiguration** data holder) with data on the current configuration if it was successful, or an empty string if it was unsuccessful. For a detailed description, refer to *“::aba::Dat-CardconfigCodec Data Holder Class” on page 230*.

Example

```
...
::aba::ApiCodecs codecs
...
set result [codecs GetCurrentConfig ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiCodecs::SetCurrentConfig

Purpose

SetCurrentConfig sets a new Codec configuration.

Command

```
::aba::ApiCodecs::SetCurrentConfig { XMLData }
```

Parameters

`XMLData`

This is a string in XML format (compatible with the **::aba::DatCodecsconfiguration** data holder) defining Codec parameters for the configuration. For a detailed description, refer to *“::aba::DatCardconfigCodec Data Holder Class” on page 230*.

Important: You must be very careful to enter all parameters correctly. Abacus checks for syntax errors, but cannot verify that parameters are in range. If you enter an out-of-range parameter, errors may be generated during compilation.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiCodecs codecs
...
# Configuring dataholder.
::aba::DatCodecsconfiguration codecsconfiguration
...
set result [codecs SetCurrentConfig  [codecsconfiguration
ExportToXMLString]]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiCodecs::GetLastError

::aba::ApiCodecs::GetLastError inherits the **::aba::ApiCustom::GetLastError** method.

Purpose

GetLastError displays the text of the last error that occurred in the last called method of the current class instance.

Command

```
::aba::ApiCodecs::GetLastError { }
```

Parameters

None

Response

The command returns the text string of the last error that occurred, or an empty string if no errors occurred.

Example

```
...
::aba::ApiCodecs codecs
```

```
...
set result [codecs GetLastError ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiCodecs::SetChannel

Purpose

SetChannel associates API-class's instance with an Abacus UI instance. If channel wasn't set explicitly, an API class instance will use te Abacus UI instance that was opened last. For more information refer to "*“How Do I Run Multiple Instances of the Abacus UI?” on page 44*.

Command

```
::aba::ApiCodecs::SetChannel { NewChannel }
```

Parameters

`NewChannel`
Channel identifier - the value that **::aba::ApiApplication::Enter** method returns upon establishing a new connection.

Response

None

Example

```
::aba::ApiApplication app
::aba::ApiCodecs codecs
...
set ip localhost
set chan [app Enter $ip]
codecs SetChannel chan
```

::aba::ApiEPDB Methods

::aba::ApiEPDB::GetPhoneBookBySet

Purpose

GetPhoneBookBySet retrieves the endpoint addresses of a given set in text, serialized as a string, from the current Abacus test environment for the specified card type, side, setno and meth.

Command

```
::aba::ApiEPDB::GetPhoneBookBySet { CardType, Side, SetNo, Meth }
```

Parameters

`CardType`
This string is a literal code representing the card type, where type may be: Analog or ICG_SIP.

`Side`
This string specifies the side: Subscriber, Exchange, or Switch.

`SetNo`
This integer specifies the set number. When the `Side` parameter is set to “Switch,” this field should be “0”. This is because the “Switch” type does not have a set number.

`Meth`
This string specifies the endpoint type: Own or External.

Response

The command returns phone book text as a string on success, or an empty string on failure.

Example

```
::aba::ApiEPDB epdb2
set phtxt [epdb2 GetPhoneBookBySet "ICG_SIP" "Subscriber" "1"
"External"]
::aba::DatEPConfig4Set epcfg2
epcfg2 ImportFromText $phtxt
epcfg2 SaveToFile {c:\temp\ep_ext.txt}
```

::aba::ApiEPDB::SetPhoneBookBySet

Purpose

SetPhoneBookBySet sets the endpoint addresses of a given set in text, serialized as a string, in the current Abacus test environment for the specified card type, side, setno and meth.

Command

```
::aba::ApiEPDB::SetPhoneBookBySet { CardType Side SetNo Meth
PhoneBook }
```

Parameters

`CardType`
This string is a literal code representing the card type, where type may be: Analog or ICG_SIP.

`Side`
This string specifies the side: Subscriber, Exchange, or Switch.

`SetNo`
This integer specifies the set number. When the `Side` parameter is set to “Switch,” this field should be “0”. This is because the “Switch” type does not have a set number.

`Meth`
This string specifies the endpoint type: Own or External.

`PhoneBook`
This string is the complete content of the phone book text (compatible with ::aba::DatEPConfig4Set data holder class) that contains the endpoints of the specified set.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiEPDB epdb
epdb SetPhoneBookBySet "ICG_SIP" "Subscriber" "1" "Own" [epcfg
ExportToText]
```

::aba::ApiPathConfirmations Methods

Note: If you are using version 4.10 or later of the Abacus software, you must prefix all classes and procedures with "::aba::" to avoid conflicts with other automation APIs. (Refer to *"Namespace" on page 27* for more information.)

::aba::ApiPathConfirmations provides Path Confirmation profile management.

::aba::ApiCustom

::aba::ApiPathConfirmations

::aba::ApiPathConfirmations::GetCurrentConfig

Purpose

GetCurrentConfig returns current Path Confirmation profiles.

Command

```
::aba::ApiPathConfirmations::GetCurrentConfig { }
```

Parameters

None

Response

This command returns a string in XML format (compatible with the **DatPathconfirmationlist** data holder) with data on the current configuration if it was successful, or an empty string if it was unsuccessful. For a detailed description, refer to *"::aba::DatPathconfirmationlist Data Holder Class" on page 238*. Note that the returned value is a list of all the path confirmations defined in the current test environment.

Example

```
...
::aba::ApiPathConfirmations pathconfirmations
...
set result [pathconfirmations GetCurrentConfig ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiPathConfirmations::UpdatePathConfirmation

Purpose

UpdatePathConfirmation updates a Path Confirmation profile.

Command

```
::aba::ApiPathConfirmations::UpdatePathConfirmation { XMLData }
```

Parameters

`XMLData`
This is a string in XML format (compatible with the **::aba::DatPathconfirmationparams** data holder) with new Path Confirmation parameters. For a detailed description, refer to *“::aba::DatPathconfirmationparams Data Holder Class” on page 239*.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiPathConfirmations pathconfirmations
...
# Configuring dataholder.
::aba::DatPathconfirmationparams pathconfirmationparams
...
set result [pathconfirmations UpdatePathConfirmation
[pathconfirmationparams ExportToXMLString]]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiPathConfirmations::CreatePathConfirmation

Purpose

CreatePathConfirmation creates a new path confirmation profile.

Command

```
::aba::ApiPathConfirmations::CreatePathConfirmation { XMLData }
```

Parameters

`XMLData`
This is a string in XML format (compatible with the **::aba::DatPathconfirmationparams** data holder) with the specified Path Confirmation parameters. For a detailed description, refer to *“::aba::DatPathconfirmationparams Data Holder Class” on page 239*.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiPathConfirmations pathconfirmations
...
# Configuring dataholder.
::aba::DatPathconfirmationparams pathconfirmationparams
...
set result [pathconfirmations CreatePathConfirmation
[pathconfirmationparams ExportToXMLString]]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiPathConfirmations::RemovePathConfirmation

Purpose

RemovePathConfirmation deletes a specified path confirmation profile.

Command

```
::aba::ApiPathConfirmations::RemovePathConfirmation { Name }
```

Parameters

`Name`
Name represents the filename of the path confirmation profile to be deleted.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiPathConfirmations pathconfirmations
...

set result [pathconfirmations RemovePathConfirmation $Name]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiPathConfirmations::GetLastError

::aba::ApiPathConfirmations::GetLastError inherits the **::aba::ApiCustom::Get-LastError** method.

Purpose

GetLastError displays the text of the last error that occurred in the last called method of the current class instance.

Command

```
::aba::ApiPathConfirmations::GetLastError { }
```

Parameters

None

Response

The command returns the text string of the last error that occurred, or an empty string if no errors occurred.

Example

```
...
::aba::ApiPathConfirmations pathconfirm
...
set result [pathconfirm GetLastError ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiPathConfirmations::SetChannel

Purpose

SetChannel associates the API class instance with an Abacus UI instance. If channel wasn't set explicitly, an API class instance will use the Abacus UI instance that was opened last. For more information refer to *"How Do I Run Multiple Instances of the Abacus UI?" on page 44*.

Command

```
::aba::ApiPathConfirmations::SetChannel { NewChannel }
```

Parameters

`NewChannel`
Channel identifier - the value that **::aba::ApiApplication::Enter** method returns upon establishing a new connection.

Response

None

Example

```
::aba::ApiApplication app
::aba::ApiPathConfirmations pathconfirm
...
set ip localhost
set chan [app Enter $ip]
pathconfirm SetChannel chan
```

::aba::ApiPnT Methods

Note: If you are using version 4.10 or later of the Abacus software, you must prefix all classes and procedures with “::aba::” to avoid conflicts with other automation APIs. (Refer to *“Namespace” on page 27* for more information.)

::aba::ApiPnT provides test Partition and Timing management. Settings correspond to settings available in the **Partition and Timing** window.

::aba::ApiCustom

::aba::ApiPnT

::aba::ApiPnT::GetCurrentConfig

Purpose

GetCurrentConfig returns data on the Partition and Timing configuration for the current test.

Command

```
::aba::ApiPnT::GetCurrentConfig { CardType Side }
```

Parameters

`CardType`
This is a literal code representing the card type, where type may be:

PRG	EANALOG	ICG_SIP
T1	SLC96_II	ICG_MGCP
E1	V5_2_BRI	ICG_H323
PRI_1544	AN_CLRCH	VRG
PRI_2048	V5_1_BRI	ICG_MEGACO
SLC96_I	SigIUA	ICG_SIPT
GR303	IUA BRI	ICG_SKINNY
BRI	V5_2_PRI	T1_BICC
V5_1	VOIP_CLRCH	E1_BICC
V5_2	SIGTRAN	IP_BICC
T1_CLRCH	T1_SS7	M2UA
E1_CLRCH	E1_SS7	

`CardSide`
This string specifies the side: *Subscriber*, *Exchange*, or *Switch*.

Response

This command returns a string in XML format (compatible with the **::aba::DatSetList** data holder) with data on the current configuration if it was successful, or an empty string if it was unsuccessful. For a detailed description, refer to *“::aba::DatSetList Data Holder Class” on page 263*.

Example

```
...
::aba::ApiPnT pnt
...
set result [pnt GetCurrentConfig $CardType $Side]
```

```
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiPnT::CreateSet

Purpose

CreateSet creates a new set of channels with the specified parameters.

Command

```
::aba::ApiPnT::CreateSet { XMLData }
```

Parameters

`XMLData`
A string in XML format (compatible with the **::aba::DatSet** data holder) that contains the set parameters. For a detailed description, refer to *"::aba::DatSet Data Holder Class" on page 260*.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiPnT pnt
...
# Configuring dataholder.
::aba::DatSet set
...
set result [pnt CreateSet  [set ExportToXMLString]]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiPnT::UpdateSet

Purpose

UpdateSet updates an existing channel set with the specified parameters.

Note: This method updates all existing set parameters *except* Set Number, Card Type, and Card Side. To change Set Number, Card Type, or Card Side existing, delete the existing set (*“::aba::ApiPnT::RemoveSet” on page 91*) and create a new set (*“::aba::ApiPnT::CreateSet” on page 89*).

Command

```
::aba::ApiPnT::UpdateSet { ParamRecXML }
```

Parameters

`ParamRecXML`
This is a string in XML format (compatible with the **::aba::DatSet** data holder) with the set parameters. For a detailed description, refer to *“::aba::DatSet Data Holder Class” on page 260*.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
 ::aba::ApiPnT pnt
...
# Configuring dataholder.
::aba::DatSet set
...
set result [pnt UpdateSet  [set ExportToXMLString]]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiPnT::RemoveSet

Purpose

RemoveSet removes a set of channels, and is equivalent to unchecking a **Set** box in the **Partition and Timing | Association** panel.

Command

```
::aba::ApiPnT::RemoveSet { CardType Side SetNo }
```

Parameters

`CardType`
This is a literal code representing the card type containing the set to be disabled, where type may be:

PRG	EANALOG	ICG_SIP
T1	SLC96_II	ICG_MGCP
E1	V5_2_BRI	ICG_H323
PRI_1544	AN_CLRCH	VRG
PRI_2048	V5_1_BRI	ICG_MEGACO
SLC96_I	SigIUA	ICG_SIPT
GR303	IUA BRI	ICG_SKINNY
BRI	V5_2_PRI	T1_BICC
V5_1	VOIP_CLRCH	E1_BICC
V5_2	SIGTRAN	IP_BICC
T1_CLRCH	T1_SS7	M2UA
E1_CLRCH	E1_SS7	

`Side`
This is the side of the circuit containing the set to be disabled, where *Sub* is subscriber side, *Ex* is exchange side, and *Sw* is switch side.

`SetNo`
This string designates the number of the set to be disabled. Acceptable range is 1 to 28, corresponding to the maximum number of sets that can be defined for a circuit type.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
 ::aba::ApiPnT pnt
...
set result [pnt RemoveSet $CardType $Side $SetNo]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiPnT::GetAddressBook

Note: This method is deprecated. Please use the GetPhoneBookBySet, SetPhoneBookBySet, and data holder classes ::aba::DatEndpoint and ::aba::DatEPConfig4Set which support PhoneBook Txt.

Purpose

GetAddressBook retrieves the address book for a specified set.

Command

```
::aba::ApiPnT::GetAddressBook { CardType Side SetNumber }
```

Parameters

`CardType`
This is a literal code representing the card type, where type may be EANALOG or ICG_SIP.

`Side`
This string specifies the side: *Subscriber*, *Exchange*, or *Switch*.

`SetNumber`
This string specifies the number of the channel set with the address book being analyzed; e.g., *2*.

Response

The command returns a string in XML format (compatible with the **::aba::DatPhonebook** data holder) with the phone book parameters on success, or an empty string otherwise. For a detailed description, refer to *“::aba::DatPhonebook Data Holder Class” on page 242*.

Example

```
...
 ::aba::ApiPnT pnt
...
set result [pnt GetAddressBook $CardType $Side $SetNo]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiPnT::SetAddressBook

Note: This method is deprecated. Please use the GetPhoneBookBySet, SetPhoneBookBySet, and data holder classes ::aba::DatEndpoint and ::aba::DatEPConfig4Set which support PhoneBook Txt.

Purpose

SetAddressBook creates a new address book and saves it under a designated name.

Command

```
::aba::ApiPnT::SetAddressBook { BookParameters }
```

Parameters

`BookParameters`
This is a string in XML format (compatible with the **::aba::DatPhonebook** data holder) with the phone book parameters. For a detailed description, refer to *“::aba::DatPhonebook Data Holder Class” on page 242*.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
 ::aba::ApiPnT pnt
...
# Configuring dataholder.
::aba::DatPhonebook phonebook
...
set result [pnt SetAddressBook  [phonebook ExportToXMLString]]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiPnT::GetChannelGroups

Purpose

GetChannelGroups returns the physical circuit groups that the set is in.

Command

```
::aba::ApiPnT::GetChannelGroups { }
```

Parameters

None

Response

The command returns a string in XML format (compatible with the **::aba::DatChannelgroups** data holder) with the channel groups partition on success, or an empty string otherwise. For a detailed description, refer to *“::aba::DatChannelgroups Data Holder Class” on page 231*.

Example

```
...
::aba::ApiPnT pnt
...
set result [pnt GetChannelGroups ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiPnT::GetLastError

::aba::ApiPnT::GetLastError inherits the **::aba::ApiCustom::GetLastError** method.

Purpose

GetLastError displays the text of the last error that occurred in the last called method of the current class instance.

Command

```
::aba::ApiPnT::GetLastError { }
```

Parameters

None

Response

The command returns the text string of the last error that occurred, or an empty string if no errors occurred.

Example

```
...
::aba::ApiPnT pnt
...
set result [pnt GetLastError ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiPnT::SetChannel

Purpose

SetChannel associates the API class instance with an Abacus UI instance. If channel wasn't set explicitly, an API class instance will use the Abacus UI instance that was opened last. For more information refer to *"How Do I Run Multiple Instances of the Abacus UI?" on page 44*.

Command

```
::aba::ApiPnT::SetChannel { NewChannel }
```

Parameters

`NewChannel`
Channel identifier - the value that **::aba::ApiApplication::Enter** method returns upon establishing a new connection.

Response

None

Example

```
::aba::ApiApplication app
::aba::ApiPnT pnt
...
set ip localhost
set chan [app Enter $ip]
pnt SetChannel chan
```

::aba::ApiProtocolSelection Methods

Note: If you are using version 4.10 or later of the Abacus software, you must prefix all classes and procedures with "::aba::" to avoid conflicts with other automation APIs. (Refer to *"Namespace" on page 27* for more information.)

::aba::ApiProtocolSelection provides Protocol Configuration management.

::aba::ApiCustom

::aba::ApiProtocolSelection

::aba::ApiProtocolSelection::GetCurrentConfig

Purpose

GetCurrentConfig returns the current Protocol Selection configuration.

Command

```
::aba::ApiProtocolSelection::GetCurrentConfig { }
```

Parameters

None

Response

This command returns a string in XML format (compatible with the **::aba::DatProtocolselection** data holder) with data on the current configuration if it was successful, or an empty string if it was unsuccessful. For a detailed description, refer to *“::aba::DatProtocolselection Data Holder Class” on page 243*.

Example

```
...
::aba::ApiProtocolSelection protocolselection
...
set result [protocolselection GetCurrentConfig ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiProtocolSelection::SetCurrentConfig

Purpose

SetCurrentConfig sets a new Protocol Selection configuration.

Command

```
::aba::ApiProtocolSelection::SetCurrentConfig { XMLData }
```

Parameters

`XMLData`

This is a string in XML format (compatible with the **::aba::DatProtocolselection** data holder) defining protocol selection parameters for the configuration. The XML string defines the parameters found on the **Card** panel of the **Protocol Selection** window (card type, card side, law, impedance, frame, and line). For a detailed description, refer to *"::aba::DatProtocolselection Data Holder Class" on page 243*.

Important: You must be very careful to enter all parameters correctly. Abacus checks for syntax errors, but it cannot verify that parameters are in range. If you enter an out-of-range parameter, errors may be generated during compilation.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiProtocolSelection protocolselection
...
# Configuring dataholder.
::aba::DatProtocolselection protocolselection
...
set result [protocolselection SetCurrentConfig
[protocolselection ExportToXMLString]]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiProtocolSelection::GetLastError

::aba::ApiProtocolSelection::GetLastError inherits the **::aba::ApiCustom::GetLastError** method.

Purpose

GetLastError displays the text of the last error that occurred in the last called method of the current class instance.

Command

```
::aba::ApiProtocolSelection::GetLastError { }
```

Parameters

None

Response

The command returns the text string of the last error that occurred, or an empty string if no errors occurred.

Example

```
...
::aba::ApiProtocolSelection ps
...
set result [ps GetLastError ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiProtocolSelection::SetChannel

Purpose

SetChannel associates the API class instance with Abacus UI instance. If channel wasn't set explicitly, an API class instance will use Abacus UI instance that was opened last. For more information refer to "*"How Do I Run Multiple Instances of the Abacus UI?" on page 44*.

Command

```
::aba::ApiProtocolSelection::SetChannel { NewChannel }
```

Parameters

`NewChannel`
Channel identifier - the value that **::aba::ApiApplication::Enter** method returns upon establishing a new connection.

Response

None

Example

```
::aba::ApiApplication app
::aba::ApiProtocolSelection ps
...
set ip localhost
set chan [app Enter $ip]
ps SetChannel chan
```

::aba::ApiQoMThresholds

Note: If you are using version 4.10 or later of the Abacus software, you must prefix all classes and procedures with "::aba::" to avoid conflicts with other automation APIs. (Refer to *"Namespace" on page 27* for more information.)

::aba::ApiQoMThresholds provides threshold and error configuration settings; it corresponds to the Abacus main menu **Configure | Channels** command.

::aba::ApiCustom

::aba::ApiQoMThresholds

::aba::ApiQoMThresholds::GetCurrentConfig

Purpose

GetCurrentConfig returns the current **PESQ** and **PSQM** threshold settings.

Command

```
::aba::ApiQoMThresholds::GetCurrentConfig { CardType }
```

Parameters

`CardType`
This is a literal code representing the card type, where type may be:

PRG	EANALOG	ICG_SIP

T1	SLC96_II	ICG_MGCP
E1	V5_2_BRI	ICG_H323
PRI_1544	AN_CLRCH	VRG
PRI_2048	V5_1_BRI	ICG_MEGACO
SLC96_I	SigIUA	ICG_SIPT
GR303	IUA BRI	ICG_SKINNY
BRI	V5_2_PRI	T1_BICC
V5_1	VOIP_CLRCH	E1_BICC
V5_2	SIGTRAN	IP_BICC
T1_CLRCH	T1_SS7	M2UA
E1_CLRCH	E1_SS7	

Response

If successful, the command returns a string in XML format (compatible with the **::aba::DatThresholds** data holder) with the threshold parameters, or an empty string if unsuccessful. For a detailed description, refer to *“::aba::DatThresholdserrors Data Holder Class” on page 275*.

Example

```
::aba::ApiQoMThresholds qomthresholds
#  Configuring  dataholder.
::aba:: DatThresholds  thresholdsdata


set  result  [qomthresholds  SetCurrentConfig [thresholdsdata
ExportToXMLString]]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiQoMThresholds::SetCurrentConfig

Purpose

SetCurrentConfig configures the PSQM and PESQ thresholds listed in the **Channels | QoM** window.

Command

```
::aba::ApiQoMThresholds::SetCurrentConfig {ParamRecXML}
```

Parameters

`ParamRecXML`

A string in XML format compatible with the **::aba::DatThresholds** data holder. For a detailed description, refer to *"::aba::DatThresholdserrors Data Holder Class" on page 275*.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiQoMThresholds qomthresholds
...
set result [qomthresholds GetLastError]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiReports Methods

Note: If you are using version 4.10 or later of the Abacus software, you must prefix all classes and procedures with "::aba::" to avoid conflicts with other automation APIs. (Refer to *"Namespace" on page 27* for more information.)

::aba::ApiReports provides reports management.

::aba::ApiCustom

::aba::ApiReports

::aba::GetRGETemplate

Purpose

GetRGETemplate retrieves a template of a default XML report configuration (the same as the Abacus 5000 UI *Generate Reports* dialog default). Retrieved information can be written to a file or sent to DatRGE data holder.

This method is a common method that belongs to "::aba::". It is not a member method of class **::aba::ApiReports**. It cannot be called by **ApiReports** objects.

Command

```
::aba::GetRGETemplate { }
```

Parameters

None

Response

Returns string in XML format that is compatible with DatRGE data holder (*“::aba::DatRge Data Holder Class” on page 252*) with default report configuration for current Abacus Software Instance.

Example

```
set template [::aba::GetRGETemplate]
::aba::WriteToFile $OUTFILE "Original RGE template:"
::aba::WriteToFile $OUTFILE $template

::aba::DatRge RGE
RGE ImportFromXMLString $template
::aba::ApiReports::SetRGEConfig
```

::aba::ApiReports::SetRGEConfig

Purpose

SetRGEConfig sets up parameters for generation of a report in current Abacus Software Instance.

Note: ::aba::ApiReports::ResetRGE should be called before the current instance of the Abacus GUI can be used to generate reports for the second time.

Command

```
::aba::ApiReports::SetRGEConfig {XMLData}
```

Parameters

`XMLData`

This is a string in XML format (compatible with DatRGE data holder) containing report configuration.

For a detailed description, refer to *"::aba::DatRge Data Holder Class" on page 252*.

Response

None

Example

```
...
::aba::ApiReports rep
...
set cfg {
<RGE>
    <Section type="root" name="Root">
        <Parameters>
            <Param name="report-format-xml">Y</Param>
            <Param name="report-file-name-root">rge_xml</Param>
        </Parameters>
        <Section type="channel-selection" name="Channel
Selection">
            <ChannelSelection selection-method="All" include-
originate="YES" include-terminate="YES" averaged="NO">
```

```
                <type-side averaged="NO" enabled="YES" type-
name="SIP" side-name="Sub">
                    <Set averaged="NO" enabled="YES" available-
channels="1-100" number="1" first-physical-channel="1"></Set>
                    <Set averaged="NO" enabled="NO" available-
channels="201-400" number="2" first-physical-channel="201"></
Set>
                </type-side>
            </ChannelSelection>
            <Section type="statistics" name="Statistics"/>
        </Section>
    </Section>
</RGE>
}
rep SetRGEConfig $cfg
```

::aba::ApiReports::GetRGEConfig

Purpose

GetRGEConfig retrieves existing report configuration. Retrieved information could be then written to a file or sent to DatRGE data holder.

Command

```
::aba::ApiReports::GetRGEConfig { }
```

Parameters

None

Response

Returns string in XML format (compatible with DatRGE data holder) with current report configuration for current Abacus Software Instance.

For a detailed description, refer to *“::aba::DatRge Data Holder Class” on page 252*.

Example

```
...
::aba::ApiReports rep
```

```
...
::aba::WriteToFile "rge.txt" [rep GetRGEConfig]
```

::aba::ApiReports::Generate

Purpose

Generate generates report according to current settings and saves it to the computer on which the Abacus Software instance was launched. This report will be deleted upon exiting current Abacus Software Instance or next test launching.

Note: (temporary limitation) **Generate** will only use settings set by **SetRGEConfig** in current Abacus Software instance.

Command

```
::aba::ApiReports::Generate { }
```

Parameters

None

Response

The command returns OK if the report was generated successfully, empty string otherwise.

Example

```
...
::aba::ApiReports rep
...
rep Generate
```

::aba::ApiReports::Download

Purpose

Download saves the generated report in a specified folder on the computer where TCL script is launched. If folder is left unspecified, report will be saved in the working directory of the launched TCL script.

Command

```
::aba::ApiReports::Download {ClientPath ChunkSize}
```

Parameters

```
ClientPath
The absolute or relative path of the report to be saved to the
computer where TCL script is launched. This parameter is
optional; its default value is an empty string.
```

Note: Always enter the path name string in braces; e.g:

```
      reports Download {c:\some folder\}
```

`ChunkSize`
The chunk size for downloading. This parameter is optional; its default value is 10485760.

Response

The command returns OK if the operation was successful, or empty string if it was unsuccessful.

Example

```
...
::aba::ApiReports rep
...
rep Generate
rep Download {Myreport}
```

::aba::ApiReports::GetFilesCount

Purpose

GetFilesCount returns the count number of current reports files.

Command

```
::aba::ApiReports::GetFilesCount  { }
```

Parameters

None

Return Value

This command returns an integer represents the count number of current reports files.

Example

```
::aba::ApiReports  report


set  result  [report GetFilesCount]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiReports::GetFileName

Purpose

GetFileName returns the name of report file at the specified index.

Command

```
::aba::ApiReports::GetFileName  { idx }
```

Parameters

`idx`
This 0-based integer designates the index of report file. It’s optional, default value is 0.

Return Value

This command returns a string represents the file name if it was successful, or an empty string if it was unsuccessful.

Example

```
::aba::ApiReports  report


set  result  [report GetFileName]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiReports::GetFileContent

Purpose

GetFileContent returns the content of report file at the specified index.

Command

```
::aba::ApiReports::GetFileContent  { idx }
```

Parameters

`idx`
This 0-based integer designates the index of report file. It’s optional, default value is 0.

Return Value

This command returns a string contains the file content if it was successful, or an empty string if it was unsuccessful.

Example

```
::aba::ApiReports  report


set  result  [report GetFileContent]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiReports::GetFileSize

Purpose

GetFileSize returns the size of report file at the specified index.

Command

```
::aba::ApiReports::GetFileSize  { idx }
```

Parameters

`idx`
This 0-based integer designates the index of report file. It’s optional, default value is 0.

Return Value

This command returns non-zero integer represents the file size if it was successful, or zero if it was unsuccessful.

Example

```
::aba::ApiReports  report


set  result  [report GetFileSize]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiReports::ResetRGE

Purpose

ResetRGE resets current RGE configuration as empty.

This method is used for reports generation after the first time.

Command

```
::aba::ApiReports:: ResetRGE  { }
```

Parameters

None

Return Value

None

Example

```
::aba::ApiReports  report


report SetRGEConfig [::aba::GetRGETemplate]
set  firstTimeResult  [report Generate]

report ResetRGE
report SetRGEConfig [::aba::GetRGETemplate]
```

```
set  secondTimeResult  [report Generate]
```

::aba::ApiReports::GetLastError

::aba::ApiReports::GetLastError inherits the **::aba::ApiCustom::GetLastError** method.

Purpose

GetLastError displays the text of the last error that occurred in the last called method of the current class instance.

Command

```
::aba::ApiReports::GetLastError { }
```

Parameters

None

Response

The command returns the text string of the last error that occurred, or an empty string if no errors occurred.

Example

```
...
::aba::ApiReports report
...
set result [report GetLastError ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiReports::SetChannel

Purpose

SetChannel associates the API class instance with an Abacus UI instance. If channel wasn't set explicitly, an API class instance will use the Abacus UI instance that was opened last. For more information refer to *"How Do I Run Multiple Instances of the Abacus UI?" on page 44*.

Command

```
::aba::ApiReports::SetChannel { NewChannel }
```

Parameters

`NewChannel`
Channel identifier - the value that **::aba::ApiApplication::Enter** method returns upon establishing a new connection.

Response

None

Example

```
::aba::ApiApplication app
::aba::ApiReports report
...
set ip localhost
set chan [app Enter $ip]
report SetChannel chan
```

::aba::ApiResults Methods

Note: If you are using version 4.10 or later of the Abacus software, you must prefix all classes and procedures with "::aba::" to avoid conflicts with other automation APIs. (Refer to *"Namespace" on page 27* for more information.)

::aba::ApiResults provides test results management.

::aba::ApiCustom

::aba::ApiResults

::aba::ApiResults::MOTSOTGetData

Purpose

MOTSOTGetData accepts the subscriber ID obtained from Abacus as a result of the the *MOTSOTSubscribe* call made during test setup and, upon success, returns an XML string that represents the minimum, maximum and average (measurement or statistic over time) result records as specified by the subscriber since the last data retrieval. The frequency of MOTSOTGetData calls must happen at least once every three hours. If the test is set up to run longer than three hours then the data retrieval operation must be called more than once.

Command

```
aba::ApiResults::MOTSOTGetData { SubscriberID }
```

Parameters

`SubscriberID`

This string identifies the subscriber making a request for measurements/statistics over time test results data. A unique subscriber ID is returned by Abacus as a result of the *MOTSOTSubscribe* call. Refer to *“::aba::ApiResults::MOTSOTSubscribe” on page 114*.

Response

The *MOTSOTGetData* command returns a string in XML format (compatible with the **::aba::DatMotsotdata** data holder) with the requested results (list of records and subscriber ID), or an empty string if no results were retrieved. For a detailed description of the data holder class, refer to the *Tcl Automation on Abacus Data Holders Reference Guide.*

Example

```
...
for { set i 0 } { $i < 10 } { incr i } {
      ::aba::WriteToFile $OUTFILE "Iteration $i..."
      ::aba::sleep 30
      ::aba::WriteToFile $OUTFILE "Call length: [results
MOTSOTGetData $id1]\n"
      ::aba::WriteToFile $OUTFILE "Call setup:  [results
MOTSOTGetData $id2]\n"
...
```

::aba::ApiResults::MOTSOTSubscribe

Purpose

MOTSOTSubscribe makes a request to the Abacus COM interface during test setup for a specific type of test measurement/statistic over time, at a specified granularity, range and aggregation, for a specified card and side type.

Command

```
aba::ApiResults::MOTSOTSubscribe { XMLParam }
```

Parameters

```
XMLParam
```

This is a string in XML format (compatible with the **::aba::DatMotsotsubscriber** data holder) defining the parameters of requested test results (measurements or statistics over time). For a detailed description of the data holder class, refer to the *Tcl Automation on Abacus Data Holders Reference Guide.*

Response

On success, *MOTSOTSubscribe* returns the ID of the associated subscriber. Otherwise, the result is zero. This subscriber ID can then be used by *MOTSOTGetData* to retrieve test results accumulated since the last retrieval. Refer to *"::aba::ApiResults::MOTSOTGet-Data" on page 113*.

Example

```
...
::aba::ApiApplication app
::aba::ApiResults results
::aba::ApiTest test
...
set id1 [results MOTSOTSubscribe [subscriber
ExportToXMLString]]
...
```

::aba::ApiResults::GetVariances

Purpose

If no parameter is listed, *GetVariances* returns a list of all delay type results that have non-zero count values. If a parameter is listed, the value of each output parameter is returned.

Command

```
::aba::ApiResults::GetVariances { VrType OutputOpt }
```

Parameters

`VrType`

VrType specifies a Variance parameter as follows:

0 = All non-zero variance parameters	34 = Not used
1 = Dial Tone Delay	35 = SIP Response Time
2 = Tone Delay	36 = Call Setup
3 = Silence	37 = Tear Down
4 = Energy	38 = Ring Time
5 = Acknowledgement Delay	39 = PESQ LQ
6 = String	40 = RTP Jitter
7 = Round Trip Delay	41 = Packet PC Lost Rate
8 = Hits	42 = RTP Packet Loss {per check interval}
9 = Clips	43 = RTP Packets Out Of Order
10 = Call Length Terminate	44 = RTP Packets Late Arrival
11 = Call Length Originate	45 = Fax Line Error Rate
12 = User Timer	46 = FTP Download Throughput Server
13 = Bit Error Rate	47 = FTP Upload Throughput Server
14 = PSQM Value	48 = RRQ Response Time
15 = One-Way Delay	49 = Sent T.38 packets
16 = Jitter	50 = Received T.38 packets
17 = Fax Connection Speed	51 = T.38 session length
18 = Fax Throughput Speed	52 = T.38 average transmission rate
19 = Modem Bit Error Rate	53 = T.38 average reception rate
20 = Ping Round-trip Delay	54 = R-Factor
21 = Ping Packet Loss Rate	55 = JMOS
22 = FTP Download Throughput Client	56 = Packets Recieved {per RTCP pkt}
23 = FTP Upload Throughput Client	57 = Packets Transmitted {per RTCP pkt}
24 = Not used	58 = Packets Drop OOM {per RTCP pkt}
25 = Modem Connect Speed, Client Tx	59 = Packets OLOAD {per RTCP packet}
26 = Modem Connect Speed, Client Rx	60 = Registration 4XX response time
27 = Modem Connect Speed, Server Tx	61 = Registration 200 response time
28 = Modem Connect Speed, Server Rx	62 = Registration success time

29 = Modem Throughput Speed, Client	63 = PESQ (MOS-LQO)
30 = Modem Throughput Speed, Server	64 = RTP Packet Loss {per RTCP packet}
31 = Not used	65 = RTP Jitter {per RTCP packet}
32 = MOS Value	66 = RTP Loss Rate {%}
33 = PESQ Value	67 = Custom SIP Script Measurement

Setting *VrType* to 0 will return values for all non-zero variance parameters

`OutputOpt`

This parameter indicates the output characteristic statistics that are to be gathered. *OutputOpt* is constructed by adding [1]+[2]+[4]+[8]+[16], where these values specify variance output characteristics, as follows:

1 – Name
2 – Count
4 – Minimum
8 – Average
16 – Maximum

Setting *OutputOpt* to zero will display all five fields.

Response

Abacus returns the variance values for the listed parameters if successful, or an empty string if unsuccessful.

Example

```
...
::aba::ApiResults results
...
set result [results GetVariances $VrType $OutputOpt]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiResults::GetEventsCount

Purpose

GetEventsCount returns the total number of events that occurred during the test.

Command

```
::aba::ApiResults::GetEventsCount { }
```

Parameters

None

Response

The command returns the total number of events that occurred during the test.

Example

```
...
::aba::ApiResults results
...
set result [results GetEventsCount ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiResults::GetEvent

Purpose

GetEvent returns information on a specified Event number.

Command

```
::aba::ApiResults::GetEvent { number }
```

Parameters

`number`
This string represents the number of the requested event.

Response

The command returns a text string with information on the requested event if it was successful, or an empty string if it was unsuccessful.

Example

```
...
::aba::ApiResults results
...
set result [results GetEvent $number]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiResults::StartEventsLog

Purpose

StartEventsLog writes the events generated from the current test into the file that you specify. If you send this command more than once, the old file will be closed, and Abacus will send subsequent events to a new file, even if *StopEventsLog* is called in between.

If you stop a test and do not issue a command to stop the log of events, when you subsequently start a test again, Abacus will overwrite any existing file with a new file, and save the events into the new file.

Command

```
::aba::ApiResults::StartEventsLog { FileName }
```

Parameters

`FileName`
This parameter specifies the absolute or relative path of the log file. Abacus creates the file in the root directory where the Abacus UI is installed.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiResults results
...
```

```
set result [results StartEventsLog $FileName]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiResults::StopEventsLog

Purpose

If events logging was in progress, *StopEventsLog* stops the logging.

If you stop a test and do not issue a *StopEventsLog* command, when you subsequently start a test again, Abacus will overwrite any existing file with a new file, and save the events into the new file.

Command

```
::aba::ApiResults::StopEventsLog { }
```

Parameters

None

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiResults results
...
set result [results StopEventsLog ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiResults::GetResults

Note: ::aba::ApiResults::GetResults method is obsolete. For reports handling, please use ApiReports methods.

Purpose

GetResults retrieves test results for the parameters specified.

Command

```
::aba::ApiResults::GetResults { ResParameters }
```

Parameters

`ResParameters`
This is a string in XML format (compatible with the **::aba::DatRge** data holder) specifying the test results to retrieve. For a detailed parameters description, refer to *“::aba::DatSccardlist Data Holder Class” on page 253* or *Appendix A, “RGE Configuration Interface.”*

Response

The command returns a string in XML format (compatible with the **::aba::DatReport** data holder) with the requested results, or an empty string if no results were retrieved. For a detailed description, refer to *“::aba::DatReport Data Holder Class” on page 247*.

Example

```
...
::aba::ApiResults results
...
# Configuring dataholder.
::aba::DatRge rge
...
set result [results GetResults  [rge ExportToXMLString]]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $resultt
...
```

::aba::ApiResults::GetLastError

::aba::ApiResults::GetLastError inherits the **::aba::ApiCustom::GetLastError** method.

Purpose

GetLastError displays the text of the last error that occurred in the last called method of the current class instance.

Command

```
::aba::ApiResults::GetLastError { }
```

Parameters

None

Response

The command returns the text string of the last error that occurred, or an empty string if no errors occurred.

Example

```
...
::aba::ApiResults results
...
set result [results GetLastError ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiResults::SetChannel

Purpose

SetChannel associates the API class instance with Abacus UI instance. If channel wasn't set explicitly, an API class instance will use Abacus UI instance that was opened last. For more information refer to "*"How Do I Run Multiple Instances of the Abacus UI?" on page 44*.

Command

```
::aba::ApiResults::SetChannel { NewChannel }
```

Parameters

`NewChannel`
Channel identifier - the value that **::aba::ApiApplication::Enter** method returns upon establishing a new connection.

Response

None

Example

```
::aba::ApiApplication app
::aba::ApiResults results
...
set ip localhost
set chan [app Enter $ip]
results SetChannel chan
```

::aba::ApiScripts Methods

::aba::ApiScripts provides the ability ro retrieve script lists and the actions listed within a given script for the current environment.

::aba::ApiCustom

::aba::ApiScripts

::aba::ApiScripts::GetCurrentConfig

Purpose

GetCurrentConfig returns the current list of scripts.

Command

```
::aba::ApiScripts::GetCurrentConfig { }
```

Parameters

None

Response

This command returns the list of scripts defined in the current environment as a string in XML format (compatible with the **::aba::DatScriptsList** data holder). For a detailed description of the data holder class, refer to the *Tcl Automation on Abacus Data Holders Reference Guide*. On failure, it returns an empty string.

Example

```
...
::aba::ApiApplication app
...
::aba::DatScriptslist lst
::aba::ApiScripts api_scr
...
lst ImportFromXMLString [api_scr GetCurrentConfig]
::aba::WriteToFile $OUTFILE [lst ExportToXMLString]
```

::aba::ApiScripts::GetActions

Purpose

GetActions returns the list of actions in the current script.

Command

```
::aba::ApiScripts::GetActions { ScriptName }
```

Parameters

`ScriptName`

The name of a script as a string.

Response

This command returns the list of actions defined in the specified script as a string in XML format (compatible with the **::aba::DatActionsList** data holder). For a detailed description of the data holder class, refer to the *Tcl Automation on Abacus Data Holders Reference Guide*. On failure, it returns an empty string.

Example

```
...
::aba::ApiApplication app
...
::aba::DatScriptslist lst
::aba::ApiScripts api_scr
::aba::ApiActions api_act
...
$script ImportFromXMLString [api_scr GetActions [$script.name
GetValue]]
...
::aba::WriteToFile $OUTFILE [lst ExportToXMLString]
```

::aba::ApiSUT Methods

Note: If you are using version 4.10 or later of the Abacus software, you must prefix all classes and procedures with "::aba::" to avoid conflicts with other automation APIs. (Refer to *"Namespace" on page 27* for more information.)

::aba::ApiSUT provides System Under Test configuration management.

::aba::ApiCustom

::aba::ApiSUT

::aba::ApiSUT::GetCurrentConfig

Purpose

GetCurrentConfig returns the current SUT configuration.

Command

```
::aba::ApiSUT::GetCurrentConfig { }
```

Parameters

None

Response

This command returns a string in XML format (compatible with the **::aba::DatSutlist** data holder) with data on the current SUT configuration if it was successful, or an empty string if it was unsuccessful. For a detailed description, refer to *“::aba::DatSutlist Data Holder Class” on page 270*.

Example

```
...
::aba::ApiSUT sut
...
set result [sut GetCurrentConfig ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSUT::SetCurrentConfig

Purpose

SetCurrentConfig sets a new SUT configuration.

Command

```
::aba::ApiSUT::SetCurrentConfig { ParamRecXML }
```

Parameters

`ParamRecXML`

This is a string in XML format (compatible with the **::aba::DatSutparams** data holder) with the current SUT configuration. For a detailed description, refer to *“::aba::DatSut-params Data Holder Class” on page 271*.

Important: You must be very careful to enter all parameters correctly. Abacus checks for syntax errors, but cannot verify that parameters are in range. If you enter an out-of-range parameter, errors may be generated during compilation.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiSUT sut
...
# Configuring dataholder.
::aba::DatSutparams sutparams
...
set result [sut SetCurrentConfig  [sutparams
ExportToXMLString]]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSUT::GetLastError

::aba::ApiSUT::GetLastError inherits the **::aba::ApiCustom::GetLastError** method.

Purpose

GetLastError displays the text of the last error that occurred in the last called method of the current class instance.

Command

```
::aba::ApiSUT::GetLastError { }
```

Parameters

None

Response

The command returns the text string of the last error that occurred, or an empty string if no errors occurred.

Example

```
...
::aba::ApiSUT sut
...
```

```
set result [sut GetLastError ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSUT::SetChannel

Purpose

SetChannel associates API-class's instance with an Abacus UI instance. If channel wasn't set explicitly, an API class instance will use the Abacus UI instance that was opened last. For more information refer to "*“How Do I Run Multiple Instances of the Abacus UI?” on page 44*.

Command

```
::aba::ApiSUT::SetChannel { NewChannel }
```

Parameters

`NewChannel`
Channel identifier - the value that **::aba::ApiApplication::Enter** method returns upon establishing a new connection.

Response

None

Example

```
::aba::ApiApplication app
::aba::ApiSUT sut
...
set ip localhost
set chan [app Enter $ip]
sut SetChannel chan
```

::aba::ApiSystemInformation Methods

Note: If you are using version 4.10 or later of the Abacus software, you must prefix all classes and procedures with “::aba::” to avoid conflicts with other automation APIs. (Refer to *“Namespace” on page 27* for more information.)

::aba::ApiSystemInformation provides Abacus system controller (SC) configuration management.

::aba::ApiCustom

::aba::ApiSystemInformation

::aba::ApiSystemInformation::ResetCards

Purpose

ResetCards resets all subsystems acquired by the user.

This method should be used in the following circumstances:

- Before starting a test, if the protocol selection configuration of the acquired cards has been changed.
- Any time a subsystem interface has been changed.

Command

```
::aba::ApiSystemInformation::ResetCards { }
```

Parameters

None

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation ResetCards ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::ResetSystem

Purpose

ResetSystem resets the entire system.

This method reboots all connected system controllers. It is not necessary to reestablish connection with the Abacus SC and reacquire cards; this will be done automatically.

This method should be followed by a ***sleep*** procedure, as described in *"Sleep Procedure" on page 68*.

Command

```
::aba::ApiSystemInformation::ResetSystem { }
```

Parameters

None

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation ResetSystem ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::GetEnvSCList

Purpose

GetEnvSCList returns a list of system controllers defined in the active environment.

Command

```
::aba::ApiSystemInformation::GetEnvSCList { }
```

Parameters

None

Response

The command returns a string in XML format (compatible with the **::aba::DatSclist** data holder) with a list of system controllers in the active environment, or an empty string if no SCs were found. For a detailed description, refer to *"::aba::DatScList Data Holder Class" on page 255*.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation GetEnvSCList ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::GetAvailableSCList

Purpose

GetAvailableSCList returns a list of all available system controllers.

Command

```
::aba::ApiSystemInformation::GetAvailableSCList { }
```

Parameters

None

Response

For each system controller found, the command will return a string in the following format:

- SC name
- SC IP address
- Compatibility status (compatibility of the software installed on the system controller with the software installed on your computer)

If there are no available SCs, the command returns an empty string.

Note: An Abacus software instance can be connected to an Abacus SC only if their versions (software and firmware) match in the three first parts separated by dots (e.g., A.B.C.X version is compatible with A.B.C.Y), or compatibility status will be the “INCOMPATIBLE SOFTWARE” string (e.g., A.B.C.X is incompatible with A.D.C.X).

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation GetAvailableSCList ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::GetAvailableSCByAddress

Purpose

GetAvailableSCByAddress searches for a system controller at the specified IP address and returns a list of its attributes.

Command

```
::aba::ApiSystemInformation::GetAvailableSCByAddress { Address
}
```

Parameters

`address`

This string represents the IP address of the system controller to be located. This string must form a valid IP address, such as 23.13.100.5.

For HypermetricAP, a valid IP address should be IP:SLOT, such as 23.13.100.5:1; where 23.13.100.5 represents Spirent TestCenter's IP address, and 1 represents AP's slot number. (Slot number is 1-based.)

Response

The command returns a list of Abacus SC parameters, separated by comma, if the Abacus SC with the given IP address is located, or an empty string otherwise.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation GetAvailableSCByAddress $Address]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::GetConnectionString

Purpose

GetConnectionString returns a string with the current connection configuration.

Command

```
::aba::ApiSystemInformation::GetConnectionString { }
```

Parameters

None

Response

For each connected system controller found, this command returns connection information in the following format:

- SC name
- SC IP address

If no system controllers are connected, the command returns the string "No connection". (This may also mean that the UI is operating in Demo mode.)

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation GetConnectionString ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::SetConnection

Purpose

SetConnection establishes a connection with the system controller at the specified IP address.

Command

```
::aba::ApiSystemInformation::SetConnection { IPAddress Password
Timeout}
```

Parameters

`IPAddress`
This string designates the IP address of the system controller. Abacus will attempt to find the system within the list of available system controllers. If this string is empty, Abacus will connect to Demo mode.

For HypermetricAP, a valid IP address should be IP:SLOT, such as 23.13.100.5:1; where 23.13.100.5 represents Spirent TestCenter's IP address, and 1 represents AP's slot number. (Slot number is 1-based.)

`password`
This string designates the password required to access the system controller. The string can be empty if no password has been assigned to the system controller at the designated IP address.

`Timeout`
This string designates the timeout value when accessing the system controller. This parameter is optional; its default value is 20000.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation SetConnection $IPAddress
$Password]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::RemoveConnection

Purpose

RemoveConnection terminates the connection with the system controller at the specified IP address.

Command

```
::aba::ApiSystemInformation::RemoveConnection { IP }
```

Parameters

`IPAddress`
This string designates the IP address of the system controller to disconnect from the Abacus instance.

For HypermetricAP, a valid IP address should be IP:SLOT, such as 23.13.100.5:1; where 23.13.100.5 represents Spirent TestCenter’s IP address, and 1 represents AP's slot number. (Slot number is 1-based.)

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation RemoveConnection $IP]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::RemoveAllConnections

Purpose

RemoveAllConnections terminates the connection with the current virtual Abacus SC (all physical Abacus SCs connected).

Command

```
::aba::ApiSystemInformation::RemoveAllConnections { }
```

Parameters

None

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation RemoveAllConnections ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::GetSCVersion

Purpose

GetSCVersion retrieves the version of the software currently installed on the system controller at the specified address.

Command

```
::aba::ApiSystemInformation::GetSCVersion { IPAddress }
```

Parameters

`IPAddress`
This string represents the IP address of the system controller for which the version is requested. This string must form a valid IP address, such as 23.13.100.5.

For HypermetricAP, a valid IP address should be IP:SLOT, such as 23.13.100.5:1; where 23.13.100.5 represents Spirent TestCenter’s IP address, and 1 represents AP's slot number. (Slot number is 1-based.)

Response

If successful, the command returns a string in XML format (compatible with the **::aba::DatScversions** data holder) with the version of the software currently installed on the specified SC. If unsuccessful, the command returns an empty string. For a detailed description, refer to *“::aba::DatScversions Data Holder Class” on page 258*.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation GetSCVersion $IPAddress]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::GetSCStatus

Purpose

GetSCStatus retrieves the current status of the system controller at the specified address.

Command

```
::aba::ApiSystemInformation::GetSCStatus { IPAddress }
```

Parameters

`IPAddress`
This string represents the IP address of the system controller for which status is requested. This string must form a valid IP address, such as 23.13.100.5.

For HypermetricAP, a valid IP address should be IP:SLOT, such as 23.13.100.5:1; where 23.13.100.5 represents Spirent TestCenter’s IP address, and 1 represents AP's slot number. (Slot number is 1-based.)

Response

If successful, the command returns a string in XML format (compatible with the **::aba::DatScstatus** data holder) with the status of the specified SC. If unsuccessful, the command returns an empty string. For a detailed description, refer to *“::aba::DatScstatus Data Holder Class” on page 256*.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation GetSCStatus $IPAddress]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::GetEnvCardList

Purpose

GetEnvCardList returns a list of acquired subsystems installed in a specified system in the current environment.

Command

```
::aba::ApiSystemInformation::GetEnvCardList { IPAddress }
```

Parameters

`IPAddress`
This string represents the IP address of the system controller for the requested Abacus system. This string must form a valid IP address, such as 23.13.100.5.

For HypermetricAP, a valid IP address should be IP:SLOT, such as 23.13.100.5:1; where 23.13.100.5 represents Spirent TestCenter's IP address, and 1 represents AP's slot number. (Slot number is 1-based.)

Response

The command returns a string in XML format (compatible with the **::aba::DatSccardlist** data holder) with a list of acquired subsystems installed in the specified system in the current environment, or an empty string if no subsystems were found. For a detailed description, refer to *"::aba::DatSccardlist Data Holder Class" on page 253*.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation GetEnvCardList $IPAddress]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::AddCard

Purpose

AddCard acquires the subsystem installed in the specified physical slot of the Abacus system whose SC has the designated IP address.

Command

```
::aba::ApiSystemInformation::AddCard { IP Slot }
```

Parameters

`IP`
IP address of the SC that controls the specified subsystem.

For HypermetricAP, a valid IP address should be IP:SLOT, such as 23.13.100.5:1; where 23.13.100.5 represents Spirent TestCenter's IP address, and 1 represents AP's slot number. (Slot number is 1-based.)

`slot`
The slot number of the physical location of the subsystem to be added to the current connection configuration. Abacus will automatically assign a *logical slot number* to the subsystem when it is added. Further operations with cards are performed by the logical (rather than physical) slot number when possible.

Response

The command returns the logical slot number assigned to the acquired subsystem on success, or a value of –**1** otherwise.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation AddCard $IP $Slot]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::RemoveCard

Purpose

RemoveCard removes a subsystem assigned to the specified logical slot from the acquired cards set.

Important: The *RemoveCard* command can remove a subsystem while a test is running. You must be careful *not* to use this command while Abacus is running a test.

Command

```
::aba::ApiSystemInformation::RemoveCard { SlotNo }
```

Parameters

`SlotNo`
SlotNo represents the logical slot number assigned to the subsystem to be removed from the current connection configuration, from 1 to 128.

Note: If you enter 0 or a number greater than 128, Abacus will display an error message.

Response

The command returns a value of **1** if the operation was successful (i.e., the specified subsystem no longer belongs to you), or **0** if it was unsuccessful. Possible causes for the return of a **0** value include:

- Using invalid input parameters (e.g., a slot number outside the valid range)
- At the completion of the command, the subsystem still belongs to you.

The command will return **1** if you designate an empty slot, or a subsystem owned by another user, since the subsystem will no longer belong to you.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation RemoveCard $SlotNo]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::GetCardList

Purpose

GetCardList returns the list of all subsystems acquired in connected systems.

Command

```
::aba::ApiSystemInformation::GetCardList { }
```

Parameters

None

Response

For each acquired subsystem, the command returns a string in the following format:

- Card type
- Card logical slot number
- IP address of the system controller for the Abacus system where the subsystem resides

For example:

```
PCG, slot 1, SC 10.1.16.160
```

If no cards are acquired, the command returns an empty string

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation GetCardList ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::GetCardConfig

Purpose

GetCardConfig searches for a subsystem in the specified logical slot; if a subsystem is found, returns its configuration.

Command

```
::aba::ApiSystemInformation::GetCardConfig { Slot }
```

Parameters

`slot`
This string represents the logical slot number assigned to this subsystem.

Response

If a subsystem is found in the specified slot number, the command will return a string in XML format (compatible with **::aba::DatCardconfig** data holder) with the subsystem configuration. If a subsystem is not found, the command returns an empty string. For a detailed description, refer to *“::aba::DatCardconfig Data Holder Class” on page 226*.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation GetCardConfig $Slot]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::ListUsers

Purpose

ListUsers returns a list of users logged onto system controllers connected to the current Abacus Software instance.

Command

```
::aba::ApiSystemInformation::ListUsers { }
```

Parameters

None.

Response

The command returns a string with a list of users logged onto system controllers connected to the current Abacus Software instance; the string is in the following format:

- Number of users on the connected system controllers
- User name
- IP addresses of system controllers the user is logged onto
- System status of system controllers the user is logged onto
- Name of Abacus Software instance.

If there are no other users, the command returns an empty string.

Example

```
...
::aba::ApiSystemInformation systeminformation
```

```
...
set result [systeminformation ListUsers ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::DeleteUsers

Purpose

DeleteUsers deletes the user specified by username from the set of users logged onto Abacus SCs connected to the current Abacus Software instance. User deletion means that all cards acquired by the user are released and their status is set to “Idle.”

Note: A user cannot be deleted while any tests are running on the cards acquired by the user.

Command

```
::aba::ApiSystemInformation::DeleteUsers { UserName }
```

Parameters

`UserName`
The name of the user to be deleted.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiSystemInformation systeminformation
...
set result [systeminformation DeleteUsers $UserName]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::GetCardStatus

Purpose

GetCardStatus returns card status for specified physical slot at the specified IP address.

Command

```
::aba::ApiSystemInformation::GetCardStatus  { IPAddress
PhysSlot Timeout }
```

Parameters

`IPAddress`
This string designates the IP address of the system controller.

For HypermetricAP, valid IP address should be IP:SLOT, such as 23.13.100.5:1, where 23.13.100.5 represents Spirent TestCenter's IP address, and 1 represents AP's slot number. (Slot number is 1-based)

`PhysSlot`
This string designates the physical slot number of card at the system controller.

`Timeout`
This string designates the timeout value when accessing the system controller. This parameter is optional, its default value is 20000.

Return Value

This command returns a string represents specified card status if it was successful, or an empty string if it was unsuccessful. A detailed error description can be retrieved by calling the *GetLastError* method.

Example

```
::aba::ApiSystemInformation  systeminformation


set  result  [systeminformation GetCardStatus $IPAddress
$PhySlot]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiSystemInformation::IsSCReady

Purpose

IsSCReady checks whether the system controller at specified IP address is ready or not.

Command

```
::aba::ApiSystemInformation::IsSCReady  { IP }
```

Parameters

`IP`
This string designates the IP address of the system controller.

For HypermetricAP, valid IP address should be IP:SLOT, such as 23.13.100.5:1, where 23.13.100.5 represents Spirent TestCenter’s IP address, and 1 represents AP’s slot number. (Slot number is 1-based)

Return Value

The command returns a value of 1 if the system controller is ready, or 0 if it is not ready or command failed. A detailed error description can be retrieved by calling the *GetLastError* method.

Example

```
::aba::ApiSystemInformation  systeminformation


set  result  [systeminformation IsSCReady $IPAddress]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiSystemInformation::GetAccumulativeStatusByIP

Purpose

GetAccumulativeStatusByIP returns accumulative status for cards at specified IP.

Command

```
::aba::ApiSystemInformation::GetAccumulativeStatusByIP  {
IPAddress }
```

Parameters

`IPAddress`
This string designates the IP address of the system controller.

For HypermetricAP, valid IP address should be IP:SLOT, such as 23.13.100.5:1,where 23.13.100.5 represents Spirent TestCenter's IP address, and 1 represents AP's slot number. (Slot number is 1-based)

Return Value

The command returns a string represents the status if it was successful, or an empty string if it was unsuccessful.

Example

```
::aba::ApiSystemInformation  systeminformation


set  result  [systeminformation GetAccumulativeStatusByIP
$IPAddress]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiSystemInformation::GetCardMapping

Purpose

GetCardMapping returns card mapping configurations between logical slots and physical slots.

Command

```
::aba::ApiSystemInformation::GetCardMapping  { }
```

Parameters

None

Return Value

This command returns a string in XML format (compatible with the **::aba::DatSclist** data holder class) with data on the current card mapping configuration if it was successful, or an empty string if it was unsuccessful. For a detailed description, refer to the *Abacus Tcl Automation Reference*.

Example

```
::aba::ApiSystemInformation  systeminformation


set  result  [systeminformation GetCardMapping]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiSystemInformation::SetCardMapping

Purpose

SetCardMapping sets the new card mapping configuration for current connected system controllers.

Command

```
::aba::ApiSystemInformation::SetCardMapping  { XMLParam }
```

Parameters

`XMLParam`
This is a string in XML format (compatible with the **::aba::DatSclist** data holder class) with data on the new card mapping configuration. For a detailed description, refer to the *Abacus Tcl Automation Reference*.

Return Value

The command returns a value of 1 if it was successful, or 0 if it was unsuccessful. A detailed error description can be retrieved by calling the *GetLastError* method.

Example

```
::aba::ApiSystemInformation  systeminformation
::aba::DatSclist sclist
sclist ImportFromXMLString [systeminformation GetCardMapping]
......
set  result  [systeminformation SetCardMapping [sclist
ExportToXMLString]]
```

```
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
```

::aba::ApiSystemInformation::GetCompatibleSlots

Purpose

GetCompatibleSlots sets the new card mapping configuration for current connected system controllers.

Command

```
::aba::ApiSystemInformation::GetCompatibleSlots { LogSlot }
```

Parameters

`LogSlot`
This string designates the logical slot number of card at the system controller.

Return Value

This command returns a string in XML format (compatible with the **::aba::DatSclist** data holder class) with data on the current compatible slots configuration if it was successful, or an empty string if it was unsuccessful. For a detailed description, refer to the *Abacus Tcl Automation Reference.*

Example

```
::aba::ApiSystemInformation systeminformation


set result [systeminformation GetCompatibleSlots $LogSlot]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
```

::aba::ApiSystemInformation::GetLastError

::aba::ApiSystemInformation::GetLastError inherits the **::aba::ApiCustom::GetLastError** method.

Purpose

GetLastError displays the text of the last error that occurred in the last called method of the current class instance.

Command

```
::aba::ApiSystemInformation::GetLastError { }
```

Parameters

None

Response

The command returns the text string of the last error that occurred, or an empty string if no errors occurred.

Example

```
...
::aba::ApiSystemInformation sysinfo
...
set result [sysinfo GetLastError ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiSystemInformation::SetChannel

Purpose

SetChannel associates the API class instance with Abacus UI instance. If channel wasn't set explicitly, an API class instance will use Abacus UI instance that was opened last. For more information refer to *“How Do I Run Multiple Instances of the Abacus UI?” on page 44*.

Command

```
::aba::ApiSystemInformation::SetChannel { NewChannel }
```

Parameters

`NewChannel`
Channel identifier - the value that **::aba::ApiApplication::Enter** method returns upon establishing a new connection.

Response

None

Example

```
::aba::ApiApplication app
::aba::ApiSystemInformation systeminformation
...
set ip localhost
set chan [app Enter $ip]
systeminformation SetChannel chan
```

::aba::ApiTerminateSettings Methods

The API class **::aba::ApiTerminateSettings** provides methods for getting and setting the configuration of terminate channels in an Abacus 5000 test environment.

::aba::ApiTerminateSettings::GetCurrentConfig

Purpose

GetCurrentConfig requests the full list of terminate channels configurations for every card type and side.

Command

aba::ApiTerminateSettings::GetCurrentConfig{}

Parameters

None

Response

This method returns the XML string containing descriptions of requested settings on success, or an empty string on failure. A detailed error description can be retrieved by calling the GetLastError method.

Example

```
::aba::ApiTerminateSettings term
```

```
set OUTFILE "GetTerminateCfg.txt"
...
term SetChannel $chan
set result [term GetCurrentConfig]
# Logging result of method execution.
::aba::WriteToFile $OUTFILE
...
```

::aba::ApiTerminateSettings::SetCurrentConfig

Purpose

SetCurrentConfig applies new settings for terminate channels configuration. If some card type/side combination(s) is omitted, corresponding UI settings are left unchanged, and no error is reported.

Command

aba::ApiTerminateSettings::SetCurrentConfig { `XMLData` }

Parameters

`XMLData`

This is a string in XML format (compatible with the **::aba::DatTerminateList** data holder) defining new parameters for the terminate channels configuration. A detailed description of the data holder class is provided in the *Tcl Automation on Abacus Data Holders Reference Guide*.

Response

Returns 1 on success, or 0 on failure. A detailed error description can be retrieved by calling the GetLastError method.

Example

```
TerminateSettings term
set OUTFILE "SetTerminateCfg.txt"
...
set result [term SetCurrentConfig $xml]
# Logging result of method execution.
::aba::WriteToFile $OUTFILE "Apply new cfg: $result"
```

```
...
```

::aba::ApiTest Methods

Note: If you are using version 4.10 or later of the Abacus software, you must prefix all classes and procedures with "::aba::" to avoid conflicts with other automation APIs. (Refer to *"Namespace" on page 27* for more information.)

::aba::ApiTest provides test configuration management.

::aba::ApiCustom

::aba::ApiTest

::aba::ApiTest::Start

Purpose

If a test was not running, *Start* will start a test with the current configuration. If a test was already running, this command causes the test to restart.

Command

```
::aba::ApiTest::Start { XMLParam }
```

Parameters

`XMLParam`

This represents configuration for starting test.
This parameter is optional and its default value is "<start-test/>".

Response

The command returns **OK** if the operation was successful, or an error message if there are compile-time errors.

Example

```
...
::aba::ApiTest test
...
set result [test Start ]
# Logging result of method execution.
```

```
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::Stop

Purpose

Stop stops a test gracefully: all channels are switched off one by one and the response from the SUT for each channel is obtained. Time needed is proportional to the number of active channels.

Command

```
::aba::ApiTest::Stop { XMLParam }
```

Parameters

`XMLParam`

This represents configuration for stoping test. This parameter is optional; its default value is “<stop-test waitUntilDone="true" graceful="true"></stop-test>”.

Response

The command returns **OK** if the operation was successful, or an error message if it was unsuccessful.

Example

```
...
::aba::ApiTest test
...
set result [test Stop ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::StopNow

Purpose

StopNow instructs Abacus to stop a test immediately. This method is for an emergency test stop, and all channels are switched off immediately. The Abacus Software instance may need some time before starting the next test, as the SUT must be ready to undergo testing.

Command

```
::aba::ApiTest::StopNow { }
```

Parameters

None

Response

The command returns **OK** if the operation was successful, or an error message if it was unsuccessful.

Example

```
...
::aba::ApiTest test
...
set result [test StopNow ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::GetStatus

Purpose

GetStatus returns the current test status.

Command

```
::aba::ApiTest::GetStatus { }
```

Parameters

None

Response

If successful, Abacus returns a string in XML format (compatible with the **::aba::DatEnvstatus** data holder) with the current test status; for example:

- *Test: in progress*
- *Test: idle*
- *Test: idle (done)*
- *Test: rebooting*
- *Test: boot*
- *Test: init*
- *Test: loaded*
- *Test: error*
- *Test: unknown*

If unsuccessful, Abacus returns an empty string.

For a detailed description, refer to *“::aba::DatEndpoint Data Holder Class” on page 233*.

Example

```
...
::aba::ApiTest test
...
set result [test GetStatus ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::SetAudioDiagnostics

Purpose

SetAudioDiagnostics configures the **Audio** portion of the **Media Monitor**.

Command

```
::aba::ApiTest::SetAudioDiagnostics { bON Port cType cSide Chan
Dir  Volume }
```

Parameters

`bON`

This represents speaker status. Valid values are 1 for On and 0 for Off. To enable both speakers, set this value to 1; to disable the speakers, set this value to 0. Passing any other value is the same as 0 (Off).

`Port`

This represents the speaker to be configured by the command. Valid values are *A* for left speaker and *B* for right speaker.

`cType`
This is a string code representing the type of channel to be monitored, where type may be:

0 = PRG	13 = SLC96_II	27 = ICG_H323
1 = T1	14 = V5_2_BRI	28 = VRG
2 = E1	15 = AN_CLRCH	29 = ICG_MEGACO
3 = PRI_1544	17 = V5_1_BRI	30 = ICG_SIPT
4 = PRI_2048	18 = SigIUA	31 = ICG_SKINNY
5 = SLC96_I	19 = IUA BRI	32 = T1_BICC
6 = GR303	20 = V5_2_PRI	33 = E1_BICC
7 = BRI	21 = VOIP_CLRCH	34 = IP_BICC
8 = V5_1	22 = SIGTRAN	35 = M2UA
9 = V5_2	23 = T1_SS7	
10 = T1_CLRCH	24 = E1_SS7	
11 = E1_CLRCH	25 = ICG_SIP	
12 = EANALOG	26 = ICG_MGCP	

`cSide`
This is the side of the channel to be monitored:

Sub = subscriber
Ex = exchange
Sw = switch

`Chan`
This is the number of the channel to be monitored.

`Dir`
This string is **Tx** if the direction is transmit, and **Rx** if the direction is receive.

`Volume`
This integer represents the volume of a speaker; valid integer values range from 0 to 10.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiTest test
...
set result [test SetAudioDiagnostics $bON $Port $cType $cSide
$Chan $Dir $Volume]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::StartAudioMonitor

Purpose

StartAudioMonitor begins capturing the audio stream on the specified channel and creates two files in the designated results folder: `left`_filename, and `right`_filename.

Command

```
::aba::ApiTest::StartAudioMonitor { cType cSide Channel Dir }
```

Parameters

`cType`

Stands for *card type*. This is a string code representing the card type to be monitored, where type may be:

0 = PRG	13 = SLC96_II	27 = ICG_H323
1 = T1	14 = V5_2_BRI	28 = VRG

2 = E1	15 = AN_CLRCH	29 = ICG_MEGACO
3 = PRI_1544	17 = V5_1_BRI	30 = ICG_SIPT
4 = PRI_2048	18 = SigIUA	31 = ICG_SKINNY
5 = SLC96_I	19 = IUA BRI	32 = T1_BICC
6 = GR303	20 = V5_2_PRI	33 = E1_BICC
7 = BRI	21 = VOIP_CLRCH	34 = IP_BICC
8 = V5_1	22 = SIGTRAN	35 = M2UA
9 = V5_2	23 = T1_SS7	
10 = T1_CLRCH	24 = E1_SS7	
11 = E1_CLRCH	25 = ICG_SIP	
12 = EANALOG	26 = ICG_MGCP	

`cSide`
This is the side of the channel to be monitored:

Sub = subscriber
Ex = exchange
Sw = switch

`Channel`
This is the number of the channel to be monitored.

`Dir`
This string is **Tx** if the direction is transmit, and **Rx** if the direction is receive.

Response

The command returns XML of the following format:

```
<files>
    <file>left_file_name</file>
    <file>right_file_name</file>
</files>
```

Where `left_file_name` and `right_file_name` are names of the files captured on the corresponding channel.

Example

```
...
::aba::ApiTest test
...
set result [test StartAudioMonitor ICG_SIP Sub 2 Tx]

# Logging result of method execution.

::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::StopAudioMonitor

Purpose

StopAudioMonitor stops the capture of an audio stream on the designated channel that was started with *StartAudioMonitor*.

Command

```
::aba::ApiTest::StopAudioMonitor { cType cSide Channel Dir }
```

Parameters

`cType`

Stands for *card type*. This is a string code representing the card type to be monitored, where type may be:

0 = PRG	13 = SLC96_II	27 = ICG_H323
1 = T1	14 = V5_2_BRI	28 = VRG
2 = E1	15 = AN_CLRCH	29 = ICG_MEGACO
3 = PRI_1544	17 = V5_1_BRI	30 = ICG_SIPT
4 = PRI_2048	18 = SigIUA	31 = ICG_SKINNY
5 = SLC96_I	19 = IUA BRI	32 = T1_BICC
6 = GR303	20 = V5_2_PRI	33 = E1_BICC
7 = BRI	21 = VOIP_CLRCH	34 = IP_BICC

8 = V5_1	22 = SIGTRAN	35 = M2UA
9 = V5_2	23 = T1_SS7	
10 = T1_CLRCH	24 = E1_SS7	
11 = E1_CLRCH	25 = ICG_SIP	
12 = EANALOG	26 = ICG_MGCP	

`cSide`
This is the side of the channel to be monitored:

Sub = subscriber
Ex = exchange
Sw = switch

`Channel`
This is the number of the channel to be monitored.

`Dir`
This string is **Tx** if the direction is transmit, and **Rx** if the direction is receive.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiTest test
...
set result [test StopAudioMonitor ICG_SIP Sub 2 Tx]

# Logging result of method execution.

::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::StartMonitor

Purpose

StartMonitor initializes and starts the **Data Link Monitor**, the **Line Protocol Monitor**, or the **VoIP Signaling Monitor**.

Important: The **StartMonitor** command is not supported for SS7 channel types. Since the **Data Link Monitors** for SS7 display signaling for link and subsystem number, rather than channel number, Abacus may not recognize the command.

Command

```
::aba::ApiTest::StartMonitor { MonitorType A_B cType cSide
Channel  FileName }
```

Parameters

`MonitorType`
This specifies the monitor type:

0 = Line Protocol Monitor
1 = Data Link Monitor
2 = VoIP Signaling Monitor

`A_B`
Speaker to be configured by the command, where **A** is left speaker and **B** is right speaker.

`cType`
This is a string code representing the card type to be monitored, where type may be:

0 = PRG	13 = SLC96_II	27 = ICG_H323
1 = T1	14 = V5_2_BRI	28 = VRG
2 = E1	15 = AN_CLRCH	29 = ICG_MEGACO
3 = PRI_1544	17 = V5_1_BRI	30 = ICG_SIPT
4 = PRI_2048	18 = SigIUA	31 = ICG_SKINNY
5 = SLC96_I	19 = IUA BRI	32 = T1_BICC
6 = GR303	20 = V5_2_PRI	33 = E1_BICC
7 = BRI	21 = VOIP_CLRCH	34 = IP_BICC
8 = V5_1	22 = SIGTRAN	35 = M2UA
9 = V5_2	23 = T1_SS7	
10 = T1_CLRCH	24 = E1_SS7	
11 = E1_CLRCH	25 = ICG_SIP	
12 = EANALOG	26 = ICG_MGCP	

`cSide`
This is the side of the channel to be monitored:

Sub = subscriber
Ex = exchange
Sw = switch

`Channel`
This is the number of the channel to be monitored.

`FileName`
A required field, this string represents the path and name of the file to which Abacus writes the monitor contents. *File* must specify a path that the Abacus server is capable of accessing. If *File* represents a file that already exists, Abacus will append the data to that file.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiTest test
...
set result [test StartMonitor $MonitorType $A_B $cType $cSide
$Channel $FileName]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::StopMonitor

Purpose

StopMonitor stops the **Data Link Monitor**, the **Line Protocol Monitor**, or the **VoIP Signaling Monitor**.

Important: The **StopMonitor** command is not supported for SS7 channel types. Since the **Data Link Monitors** for SS7 display signaling for link and subsystem number, rather than channel number, Abacus may not recognize the command.

Command

```
::aba::ApiTest::StopMonitor { MonitorType  A_B cType cSide
Channel }
```

Parameters

`MonitorType`
This specifies the monitor type:

0 = Line Protocol Monitor
1 = Data Link Monitor
2 = VoIP Signaling Monitor

`A_B`
Speaker to be configured by the command, where **A** is left speaker and **B** is right speaker.

`cType`
This is a string code representing the card type to be monitored, where type may be:

0 = PRG	13 = SLC96_II	27 = ICG_H323
1 = T1	14 = V5_2_BRI	28 = VRG
2 = E1	15 = AN_CLRCH	29 = ICG_MEGACO
3 = PRI_1544	17 = V5_1_BRI	30 = ICG_SIPT
4 = PRI_2048	18 = SigIUA	31 = ICG_SKINNY
5 = SLC96_I	19 = IUA BRI	32 = T1_BICC
6 = GR303	20 = V5_2_PRI	33 = E1_BICC
7 = BRI	21 = VOIP_CLRCH	34 = IP_BICC
8 = V5_1	22 = SIGTRAN	35 = M2UA
9 = V5_2	23 = T1_SS7	
10 = T1_CLRCH	24 = E1_SS7	
11 = E1_CLRCH	25 = ICG_SIP	
12 = EANALOG	26 = ICG_MGCP	

`cSide`
This is the side of the channel being monitored:

Sub = subscriber
Ex = exchange
Sw = switch

`Channel`
This is the number of the channel being monitored.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiTest test
...
set result [test StopMonitor $MonitorType $A_B $cType $cSide
$Channel]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::ActivateManualMode

Purpose

ActivateManualMode controls the initial channel state. If the *On/Off* parameter is TRUE, all channels will be stopped on test start. If the *On/Off* parameter is FALSE, channels will be stopped on test start; otherwise, they will start as usual.

Command

```
::aba::ApiTest::ActivateManualMode { ONOFF }
```

Parameters

`ONOFF`
The mode type number. where **1** means ON, and **0** means OFF.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiTest test
...
set result [test ActivateManualMode $ONOFF]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::RunChannel

Purpose

RunChannel starts specified channels.

Command

```
::aba::ApiTest::RunChannel { XMLData }
```

Parameters

`XMLData`
This string in XML format (compatible with the **::aba::DatSetchannels** data holder) contains a list of channels to be started. For a detailed description, refer to *“::aba::DatSetchannels Data Holder Class” on page 261*.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiTest test
```

```
...
# Configuring dataholder.
::aba::DatSetchannels setchannels
...
set result [test RunChannel  [setchannels ExportToXMLString]]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::PauseChannel

Purpose

PauseChannel pauses specified channel(s).

Command

```
::aba::ApiTest::PauseChannel { XMLData }
```

Parameters

`XMLData`
This string in XML format (compatible with the **::aba::DatSetchannels** data holder) contains a list of channels to be paused. For a detailed description, refer to *“::aba::DatSetchannels Data Holder Class” on page 261*.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiTest test
...
# Configuring dataholder.
::aba::DatSetchannels setchannels
...
set result [test PauseChannel  [setchannels ExportToXMLString]]
# Logging result of method execution.
```

```
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::GetDuration

Purpose

GetDuration is used to request the current settings of the **Test Duration** window.

Command

```
::aba::ApiTest::GetDuration { }
```

Parameters

None

Response

The command returns a string in XML format (compatible with **::aba::DatTestduration** data holder) with the test duration parameters on success, or an empty string otherwise. For a detailed description, refer to *“::aba::DatTestduration Data Holder Class” on page 273*.

Example

```
...
::aba::ApiTest test
...
set result [test GetDuration ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::SetDuration

Purpose

SetDuration is used to configure the execution period of the test. This command provides the same function as the **Test Duration** window; the four duration modes are continuous, interval (days, hours minutes), until (date and time), and total scripts.

Command

```
::aba::ApiTest::SetDuration { ParamRecXML }
```

Parameters

`ParamRecXML`

This is a string in XML format (compatible with **::aba::DatTestduration** data holder) that designates the test duration type (e.g., *Continuous*) and duration (e.g., number of scripts per channel). For a detailed description, refer to *"::aba::DatTestduration Data Holder Class" on page 273*.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiTest test
...
# Configuring dataholder.
::aba::DatTestduration testduration
...
set result [test SetDuration  [testduration ExportToXMLString]]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::GetLastError

::aba::ApiTest::GetLastError inherits the **::aba::ApiCustom::GetLastError** method.

Purpose

GetLastError displays the text of the last error that occurred in the last called method of the current class instance.

Command

```
::aba::ApiTest::GetLastError { }
```

Parameters

None

Response

The command returns the text string of the last error that occurred, or an empty string if no errors occurred.

Example

```
...
::aba::ApiTest test
...
set result [test GetLastError ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiTest::SetChannel

Purpose

SetChannel associates the API class instance with Abacus UI instance. If channel wasn't set explicitly, an API class instance will use Abacus UI instance that was opened last. For more information refer to "*“How Do I Run Multiple Instances of the Abacus UI?” on page 44*".

Command

```
::aba::ApiTest::SetChannel { NewChannel }
```

Parameters

`NewChannel`
Channel identifier - the value that **::aba::ApiApplication::Enter** method returns upon establishing a new connection.

Response

None

Example

```
::aba::ApiApplication app
::aba::ApiTest test
...
set ip localhost
set chan [app Enter $ip]
test SetChannel chan
```

::aba::ApiThresholdsErrors Methods

Note: If you are using version 4.10 or later of the Abacus software, you must prefix all classes and procedures with "::aba::" to avoid conflicts with other automation APIs. (Refer to *"Namespace" on page 27* for more information.)

::aba::ApiThresholdsErrors provides threshold and error configuration settings; corresponds to the Abacus main menu **Configure | Thresholds and Errors** command.

::aba::ApiCustom

::aba::ApiThresholdsErrors

::aba::ApiThresholdsErrors::GetCurrentConfig

Purpose

GetCurrentConfig returns the current threshold and error settings parameters listed in the **Thresholds and Errors** window for a given card type and side.

Command

```
::aba::ApiThresholdsErrors::GetCurrentConfig { }
```

Parameters

None

Response

If successful, the command returns a string in XML format (compatible with the **::aba::DatThresholdserrorslist** data holder) with the thresholds and errors parameters, or an empty string if unsuccessful. For a detailed description, refer to *“::aba::DatThresholdserrorslist Data Holder Class” on page 277*.

Example

```
...
::aba::ApiThresholdsErrors thresholdserrors
...
set result [thresholdserrors GetCurrentConfig]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiThresholdsErrors::SetCurrentConfig

Purpose

SetCurrentConfig configures the parameters listed in the **Thresholds and Errors** window. This table is used to specify thresholds and timeout periods for Abacus error conditions.

Command

```
::aba::ApiThresholdsErrors::SetCurrentConfig { ParamRecXML }
```

Parameters

`ParamRecXML`

A string in XML format compatible with the **::aba::DatThresholdserrors** data holder. For a detailed description, refer to *“::aba::DatThresholdserrorslist Data Holder Class” on page 277*.

Response

The command returns a value of **1** if the operation was successful, or **0** if it was unsuccessful.

Example

```
...
::aba::ApiThresholdsErrors thresholdserrors
...
# Configuring dataholder.
::aba::DatThresholdserrors thresholdserrorsdata
...
set result [thresholdserrors SetCurrentConfig
[thresholdserrorsdata ExportToXMLString]]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiThresholdsErrors::GetLastError

::aba::ApiThresholdsErrors::GetLastError inherits the **::aba::ApiCustom::GetLastError** method.

Purpose

GetLastError displays the text of the last error that occurred in the last called method of the current class instance.

Command

```
::aba::ApiThresholdsErrors::GetLastError { }
```

Parameters

None

Response

The command returns the text string of the last error that occurred, or an empty string if no errors occurred.

Example

```
...
::aba::ApiThresholdsErrors thresholdserrors
...
set result [thresholdserrors GetLastError ]
```

```
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiThresholdsErrors::SetChannel

Purpose

SetChannel associates the APIclass instance with the Abacus UI instance. If channel wasn't set explicitly, an API class instance will use the Abacus UI instance that was opened last. For more information refer to *“How Do I Run Multiple Instances of the Abacus UI?” on page 44*.

Command

```
::aba::ApiThresholdsErrors::SetChannel { NewChannel }
```

Parameters

`NewChannel`
Channel identifier - the value that **::aba::ApiApplication::Enter** method returns upon establishing a new connection.

Response

None

Example

```
::aba::ApiApplication app
::aba::ApiThresholdsErrors thresholdserrors
...
set ip localhost
set chan [app Enter $ip]
thresholdserrors SetChannel chan
```

::aba::ApiToneSettings Methods

The API class **::aba::ApiToneSettings** provides methods for getting and setting the configuration of tones in an Abacus 5000 test environment.

::aba::ApiToneSettings::GetCurrentConfig

Purpose

GetCurrentConfig requests the collection of channel tone settings for every card type and side.

Command

aba::ApiToneSettings::GetCurrentConfig{}

Parameters

None

Response

This method returns the XML string containing descriptions of requested settings on success, or an empty string on failure. A detailed error description can be retrieved by calling the GetLastError method.

Example

```
...
::aba::ApiToneSettings tones
...
set result [tones GetCurrentConfig ]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiToneSettings::SetCurrentConfig

Purpose

SetCurrentConfig applies new settings for channel tones. If some card type/side combination(s) is omitted, corresponding UI settings are left unchanged and no error is reported.

Command

aba::ApiToneSettings::SetCurrentConfig { XMLData}

Parameters

`XMLData`

This is a string in XML format (compatible with the **::aba::DatToneList** data holder) defining new parameters for the channel tones configuration. A detailed description of the data holder class is provided in the *Tcl Automation on Abacus Data Holders Reference Guide*.

Response

Returns 1 on success, or 0 on failure. A detailed error description can be retrieved by calling the GetLastError method.

Example

```
...
::aba::ApiToneSettings tones
...
# Configuring dataholder.
::aba::DatToneList sutparams
...
set result [sut SetCurrentConfig [sutparams
ExportToXMLString]]
# Logging result of method execution.
::aba::WriteToFileEx "log.txt" $result
...
```

::aba::ApiResultRecords Methods

The API class ::aba::ApiResultRecords provides methods for accessing the QoM results shown in the *QoM Monitor* GUI window.

::aba::ApiResultRecords::GetRowCount

Purpose

GetRowCount retrieves the number of records in the result table.

Command

aba::ApiResultRecords::GetRowCount {TableID}

Parameters

`TableID`
This is an integer value which represents the table ID. As of this release, there are only two tables supported: QoM events and Fax events.

TableID	GUI Window
0	Regular events
1	System events
2	SS7 events
3	Internal data, no GUI presentation
4	QoM events
5	Fax events

Response

Returns a non-negative integer on success, or -1 on failure. A detailed error description can be retrieved by calling the GetLastError method.

Example

```
::aba::ApiResultRecords  resrec
set  result  [resrec GetRowCount 4]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiResultRecords::GetRowData

Purpose

GetRowData retrieves an XML string with a list of fields for a particular record in the result table.

Command

aba::ApiResultRecords::GetRowData {TableID RowIdx}

Parameters

`TableID`
This is an integer value which represents the table ID. As of this release, there are only two tables supported: QoM events and Fax events.

TableID	GUI Window
0	Regular events
1	System events
2	SS7 events
3	Internal data, no GUI presentation
4	QoM events
5	Fax events

`RowIdx`
This is an integer value which represents the row index. It is 0-based.

Response

Returns a non-empty string on success, or an empty string on failure. A detailed error description can be retrieved by calling the GetLastError method.

Example

```
::aba::ApiResultRecords  resrec
set result [resrec GetRowData 4 0]
#  Logging  result  of  method  execution.
::aba::WriteToFileEx  "log.txt"  $result
```

::aba::ApiCallTracer Methods

::aba::ApiCallTracer provides methods which can start/stop call tracer and export the captured messages.

There is only one correct order of invoking these methods:

1. CaptureStart
2. ExportStart
3. ExportStop (for continuous ExportStart only)
4. CaptureStop

Steps 2 and 3 can be repeated if necessary.

ExportStart, ExportStop, and CaptureStop should use the CSID obtained from CaptureStart.

::aba::ApiCallTracer::CaptureStart

Purpose

Creates new capturing session associated with a particular channel or a physical port.

Command

```
::aba::ApiCallTracer::CaptureStart  {  XMLParam  }
```

Parameters

`XMLParam`
This represents the configuration for starting capture.

For example:

1) ewc-params

```
    <capture-descr>
      <ewc-params card-number="1" port-number="0" ewc-filter="" stop-after-sec=""/>
      <str-filter></str-filter>
    </capture-descr>
```

2) sig-params

```
    <capture-descr>
      <sig-params type="ICG_SIP" side="Subscriber" channel="1"/>
      <str-filter></str-filter>
    </capture-descr>
```

In case "str-filter" is not used, corresponding XML node can either contain empty text or be omitted entirely.

Response

Returns Capture Session ID (CSID) on success, or 0 on failure. A detailed error description can be retrieved by calling the GetLastError method.

Example

```
::aba::ApiCallTracer tracer
```

```
set conf {
    <capture-descr>
      <sig-params type="ICG_SIP" side="Subscriber" channel="1"/>
      <str-filter></str-filter>
    </capture-descr>
}
set trid [tracer CaptureStart $conf]

#  Logging  result  of  method  execution.
::aba::WriteToFile "log.txt" $trid
```

::aba::ApiCallTracer::CaptureStop

Purpose

Terminates the capturing session.

Command

```
::aba::ApiCallTracer::CaptureStop  {  ID  }
```

Parameters

`ID`
Previously obtained Capture Session ID by CaptureStart.

Response

Returns -1 on success, or 0 on failure. A detailed error description can be retrieved by calling the GetLastError method.

Example

```
::aba::ApiCallTracer tracer

set result [tracer CaptureStop $trid]
#  Logging  result  of  method  execution.
::aba::WriteToFile "log.txt" $result
```

::aba::ApiCallTracer::ExportStart

Purpose

Exports messages received within a capturing session.

Command

```
::aba::ApiCallTracer::ExportStart  {  ID XMLParam  }
```

Parameters

`ID`
Previously obtained Capture Session ID by CaptureStart.

`XMLParam`
This represents the configuration for starting export.

For example:

1) No-continouos export

```
   <capture-export-descr file="messages1.txt" titles-only="no"
continuously="no"/>
```

2) Continouos export

```
   <capture-export-descr file="messages2.txt" titles-only="no"
continuously="yes"/>
```

Response

Returns -1 on success, or 0 on failure.. A detailed error description can be retrieved by calling the GetLastError method.

Example

```
::aba::ApiCallTracer tracer


set  ECont {
<capture-export-descr file="res_continuously.txt" titles-
only="no" continuously="yes"/>
}


set result [tracer ExportStart $trid $ECont]
```

```
# Logging result of method execution.
::aba::WriteToFile "log.txt" $result
```

::aba::ApiCallTracer::ExportStop

Purpose

Stops message export in case of continuous exporting.

Note: This method should not be called if export was not continuous!

Command

```
::aba::ApiCallTracer::ExportStop { ID }
```

Parameters

`ID`
Previously obtained Capture Session ID by CaptureStart.

Response

Returns -1 on success, or 0 on failure. A detailed error description can be retrieved by calling the GetLastError method.

Example

```
::aba::ApiCallTracer tracer

set result [tracer ExportStop $trid]
# Logging result of method execution.
::aba::WriteToFile "log.txt" $result
```

::aba::ApiFileTransfer Methods

The API class ::aba::ApiFileTransfer provides methods for accessing files located on the automation server.

::aba::ApiFileTransfer::GetFileSize

Purpose

GetFileSize retrieves the size (in bytes) of a specified file located on the automation server.

Command

```
aba::ApiFileTransfer::GetFileSize {FolderID FileName}
```

Parameters

`FolderID`
This is a symbolic label (not a path) to determine Abacus folder type. The following IDs are supported (the *Abacus folder* column only contains the default values. The actual values depend on your settings):

FolderID	Abacus folder
::aba::FolderIDs::DR_PATHCONF	PathConf
::aba::FolderIDs::DR_SUT	SUT
::aba::FolderIDs::DR_CHANNELS	Channels
::aba::FolderIDs::DR_FAXIMAGES	Images
::aba::FolderIDs::DR_SCRIPT	SCRIPTS
::aba::FolderIDs::DR_CERTIFICATES	Certificates
::aba::FolderIDs::DR_TEMP	ZTemp
::aba::FolderIDs::DR_SC	Firmware
::aba::FolderIDs::DR_RTP	PCap
::aba::FolderIDs::DR_MANUAL	MANUAL
::aba::FolderIDs::DR_ACTION	ACTIONS
::aba::FolderIDs::DR_INSTALL	Software install path
::aba::FolderIDs::DR_TEMPLATES	Templates
::aba::FolderIDs::DR_AUDIOEVENTS	AUDIO

FolderID	Abacus folder
::aba::FolderIDs::DR_HOME	Software home path
::aba::FolderIDs::DR_LUASCRIPTS	LUA/SIP
::aba::FolderIDs::DR_FTP	FTP
::aba::FolderIDs::DR_PHONE	PHONES
::aba::FolderIDs::DR_VIDEOS	VIDEOS
::aba::FolderIDs::DR_RESULTROOT	Results
::aba::FolderIDs::DR_REPORTSROOT	Reports
::aba::FolderIDs::DR_WCRESULTROOT	WCResults
::aba::FolderIDs::DR_PROTOCOLS	Protocols
::aba::FolderIDs::DR_ENVIRONMENT	Environment
::aba::FolderIDs::DR_PSQMSOUNDS	SOUNDS
::aba::FolderIDs::DR_LICENSES	Folder for licenses
::aba::FolderIDs::DR_LATESTRESULTS	The latest folder in "Results"
::aba::FolderIDs::DR_RESULT	Results
::aba::FolderIDs::DR_XMLCONFIG	XMLConfig
::aba::FolderIDs::DR_CALLLOADPROFILES	CallLoadProfiles
::aba::FolderIDs::DR_WCLATESTRESULTS	The latest folder in "WCResults"

`FileName`
This is a string which represents the file name with extension.

Response

Returns the file size in bytes, or “0” if an error is detected. If the server-side file is actually zero bytes long, GetLastError method returns “no transfer required” message. In other cases, it returns an error message description.

Example

```
::aba::ApiFileTransfer ft
set result [ ft GetFileSize $folderID log.txt ]
```

```
#    Logging    result    of    method    execution.
::aba::WriteToFileEx "log.txt" $result
```

::aba::ApiFileTransfer::Download

Purpose

Download retrieves the specified file from the automation server, and stores it on the local client path.

Command

aba::ApiFileTransfer::Download { FolderID FileName {ClientPath {}} {ChunkSize 10485760} }

Parameters

`FolderID`
This is a symbolic label (not a path) to determine Abacus folder type. The following IDs are supported (the *Abacus folder* column only contains default values. The actual values depend on your settings):

FolderID	Abacus folder
::aba::FolderIDs::DR_PATHCONF	PathConf
::aba::FolderIDs::DR_SUT	SUT
::aba::FolderIDs::DR_CHANNELS	Channels
::aba::FolderIDs::DR_FAXIMAGES	Images
::aba::FolderIDs::DR_SCRIPT	SCRIPTS
::aba::FolderIDs::DR_CERTIFICATES	Certificates
::aba::FolderIDs::DR_TEMP	ZTemp
::aba::FolderIDs::DR_SC	Firmware
::aba::FolderIDs::DR_RTP	PCap
::aba::FolderIDs::DR_MANUAL	MANUAL
::aba::FolderIDs::DR_ACTION	ACTIONS

FolderID	Abacus folder
::aba::FolderIDs::DR_INSTALL	Software install path
::aba::FolderIDs::DR_TEMPLATES	Templates
::aba::FolderIDs::DR_AUDIOEVENTS	AUDIO
::aba::FolderIDs::DR_HOME	Software home path
::aba::FolderIDs::DR_LUASCRIPTS	LUA/SIP
::aba::FolderIDs::DR_FTP	FTP
::aba::FolderIDs::DR_PHONE	PHONES
::aba::FolderIDs::DR_VIDEOS	VIDEOS
::aba::FolderIDs::DR_RESULTROOT	Results
::aba::FolderIDs::DR_REPORTSROOT	Reports
::aba::FolderIDs::DR_WCRESULTROOT	WCResults
::aba::FolderIDs::DR_PROTOCOLS	Protocols
::aba::FolderIDs::DR_ENVIRONMENT	Environment
::aba::FolderIDs::DR_PSQMSOUNDS	SOUNDS
::aba::FolderIDs::DR_LICENSES	Folder for licenses
::aba::FolderIDs::DR_LATESTRESULTS	The latest folder in “Results”
::aba::FolderIDs::DR_RESULT	Results
::aba::FolderIDs::DR_XMLCONFIG	XMLConfig
::aba::FolderIDs::DR_CALLLOADPROFILES	CallLoadProfiles
::aba::FolderIDs::DR_WCLATESTRESULTS	The latest folder in “WCResults”

`FileName`
This is a string which represents the file name with extension.

`ClientPath`
This is a string which represents the client-side destination path; an empty string (means the current folder) by default.

`ChunkSize`
This is an integer value which represents the size of one data block used for each network transaction step; 10485760 bytes (10 Mbytes) by default.

Response

Returns empty string on success, and a local error description on failure.
See the Call GetLastError method to obtain the server-side error description.

Example

```
::aba::ApiFileTransfer ft
set result [ ft Download $folderID log.txt ]
#    Logging    result    of    method    execution.
::aba::WriteToFileEx "log.txt" $result
```

::aba::Api Classes and Features Examples

This section provides Tcl example scripts for every ::aba::Api class. In these examples, Tcl Automation method calls are wrapped into ::aba::WriteToFile “filename” [method call result] sections and ::aba::WriteToFile "filename" “method description” sections. Therefore, sample scripts provide a readable output into a text file. You can find out what methods in what sequence were called and what values were returned. By using parts of these scripts in your own scripts, you can omit such write-to-file sections. However, the style of the sample scripts is preferable.

::aba::ApiActions

```
# Script 1: Change the action HOOK mode


package require tclreadyapi


set OUTFILE "changeaction.txt"


::aba::ApiApplication app


set chan [app Enter localhost]
::aba::WriteToFile $OUTFILE "Channel $chan"


::aba::ApiActions act
::aba::DatAction dat


set name "A calls B, Pulse, confirms for Call Length"
dat ImportFromXMLString [act GetAction $name]


set idx [::aba::FindInObjList [dat GetList_task] "name"
$::aba::DatXstasknames::HOOK]
if { $idx != -1 } {
    set task [dat GetByIndex_task $idx]
    $task.mode SetValue $::aba::DatXstaskmodes::OFF
    set res [act SetAction [dat ExportToXMLString]]
    ::aba::WriteToFile $OUTFILE $res
}


app Exit
```

```
exit
# Script 2: Set all three action SEND STRING parameters

package require tclreadyapi

set OUTFILE "setaction.txt"

::aba::ApiApplication app

set chan [app Enter localhost]
::aba::WriteToFile $OUTFILE "Channel $chan"

::aba::ApiActions act
::aba::DatAction dat
::aba::DatTask task
::aba::DatParam param1
::aba::DatParam param2
::aba::DatParam param3

dat.name SetValue "A calls B, Pulse, confirms for Call Length"

param1.name SetValue $::aba::DatXsparamnames::VALUE
param1.value SetValue "test string"
param2.name SetValue $::aba::DatXsparamnames::TYPE
param2.value SetValue $::aba::DatXsparamvalues::NORMAL
param3.name SetValue $::aba::DatXsparamnames::MODE
param3.value SetValue $::aba::DatXsparamvalues::OVERLAP

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::STRING
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3

dat AddItem_task task
set res [act SetAction [dat ExportToXMLString]]
```

```
::aba::WriteToFile $OUTFILE $res

app Exit
exit
```

::aba::ApiApplication

```
package require tclreadyapi

::aba::ApiApplication app

if { 0 == [app Enter localhost] } {
  ::aba::WriteToFile "_app_.txt" "Unable to establish
connection"
  exit
}

::aba::WriteToFile "_app_.txt" "Is Abacus ready?"
::aba::WriteToFile "_app_.txt" [app IsAbacusReady]

app Load "Demo_A5K_SIP.env"

set caption [app GetAbacusCaption]
::aba::WriteToFileEx "_app_.txt" "Current caption"
::aba::WriteToFile "_app_.txt" $caption

sleep 10

app Save "MyEnv_tmp.env"

set caption [app GetAbacusCaption]
::aba::WriteToFileEx "_app_.txt" "Current caption"
::aba::WriteToFile "_app_.txt" $caption

set phonesdir [app GetDirectoryPath "Phones"]
::aba::WriteToFileEx "_app_.txt" "Current PHONESDIR value is:"
::aba::WriteToFile "_app_.txt" $phonesdir
```

```
app SetDirectoryPath {c:\MyAbacus\NewPhones} "Phones"

set phonesdir [app GetDirectoryPath "Phones"]
::aba::WriteToFileEx "_app_.txt" "New PHONESDIR value is:"
::aba::WriteToFile "_app_.txt" $phonesdir

app Exit

::aba::WriteToFile "_ps_.txt" "Terminated"

exit
```

::aba::ApiCodecs

```
package require tclreadyapi

::aba::ApiApplication app
::aba::ApiCodecs codecs

if { 0 == [app Enter localhost] } {
  ::aba::WriteToFile "_codecs_.txt" "Unable to establish
connection"
  exit
}

set xmlstring [codecs GetCurrentConfig]
::aba::WriteToFile "_codecs_.txt" "Current codecs:"
::aba::WriteToFile "_codecs_.txt" $xmlstring

::aba::DatCodecsconfiguration cfg
cfg ImportFromXMLString $xmlstring

cfg.video.h261.maxpacketsize SetValue 1415

::aba::WriteToFile "_codecs_.txt" "Update maxpacketsize for H-
261:"
::aba::WriteToFile "_codecs_.txt" [cfg ExportToXMLString]
```

```
::aba::WriteToFile "_codecs_.txt" "Try to set new codecs..."
::aba::WriteToFile "_codecs_.txt" [codecs SetCurrentConfig [cfg
ExportToXMLString]]

set xmlstring [codecs GetCurrentConfig]
::aba::WriteToFile "_codecs_.txt" "Current codecs after the
update:"
::aba::WriteToFile "_codecs_.txt" $xmlstring

app Exit

::aba::WriteToFile "_codecs_.txt" "Terminated"

exit
```

::aba::ApiEPDB

```
# Script 1: Configure endpoints and set them in EPDB
::aba::DatEPConfig4Set epcfg

::aba::DatEndpoint ep1
ep1 SetProperty "User Name" "Joe4"
ep1 SetProperty "IP v4" "10.2.16.11"
ep1 SetProperty "Domain IPv4" "10.2.16.1"
ep1 SetProperty "Gateway IPv4" "10.2.16.1"
ep1 SetProperty "Subnet Mask IPv4" "255.255.0.0"
epcfg AddEndpoint ep1

::aba::DatEndpoint ep2
ep2 SetProperty "User Name" "Joe9"
ep2 SetProperty "IP v4" "10.2.16.11"
ep2 SetProperty "Domain IPv4" "10.2.16.1"
ep2 SetProperty "Gateway IPv4" "10.2.16.1"
ep2 SetProperty "Subnet Mask IPv4" "255.255.0.0"
epcfg AddEndpoint ep2

::aba::DatEndpoint ep3
ep3 SetProperty "User Name" "Joe12"
ep3 SetProperty "IP v4" "10.2.16.11"
```

```
ep3 SetProperty "Domain IPv4" "10.2.16.1"
ep3 SetProperty "Gateway IPv4" "10.2.16.1"
ep3 SetProperty "Subnet Mask IPv4" "255.255.0.0"
epcfg AddEndpoint ep3

::aba::ApiEPDB epdb
epdb SetPhoneBookBySet "ICG_SIP" "Subscriber" "1" "Own" [epcfg
ExportToText]

#...

# Script 2: Retrieve endpoints from EPDB
::aba::ApiEPDB epdb2
set phtxt [epdb2 GetPhoneBookBySet "ICG_SIP" "Subscriber" "1"
"External"]
::aba::DatEPConfig4Set epcfg2
epcfg2 ImportFromText $phtxt
epcfg2 SaveToFile {c:\temp\ep_ext.txt}

#...

# Script 3: Import endpoints from a local PhoneBook Txt file.
::aba::DatEPConfig4Set epcfg3
epcfg3 LoadFromFile {c:\temp\ep_ext.txt}
::aba::ApiEPDB epdb3
epdb3 SetPhoneBookBySet [epcfg3 ExportToText]
```

::aba::ApiPathConfirmations

```
package require tclreadyapi

::aba::ApiApplication app
::aba::ApiPathConfirmations pc
::aba::ApiTest test

if { 0 == [app Enter localhost] } {
  ::aba::WriteToFile "_codecs_.txt" "Unable to establish
connection"
  exit
}
```

```
::aba::WriteToFile "_pc_.txt" "Load environment..."
::aba::WriteToFile "_pc_.txt" [app Load "Demo_A5K_SIP.env"]

::aba::DatPathconfirmationparams pathconf

pathconf.toneslope.name SetValue "ToneSlopePC1"
pathconf.toneslope.parameters.f1 SetValue "404"
pathconf.toneslope.parameters.f2 SetValue "1004"
pathconf.toneslope.parameters.f3 SetValue "2804"
pathconf.toneslope.parameters.f1 SetValue "150"

pc CreatePathConfirmation [pathconf ExportToXMLString]

set pcstring [pc GetCurrentConfig]

::aba::WriteToFileEx "_pc_.txt" "The Charecteristics for path
confirmation are...\n\n$pcstring"

::aba::WriteToFile "_pc_.txt" "Start test"
::aba::WriteToFile "_pc_.txt" [test Start]

sleep 60

::aba::WriteToFile "_pc_.txt" "Stop test"
::aba::WriteToFile "_pc_.txt" [test Stop]

pathconf.toneslope.parameters.f1 SetValue "204"
pathconf.toneslope.parameters.f2 SetValue "1000"
pathconf.toneslope.parameters.f3 SetValue "2004"
pathconf.toneslope.parameters.f1 SetValue "230"

pathconf UpdatePathConfirmation [pathconf ExportToXMLString]

::aba::WriteToFileEx "_pc_.txt" "And now they are set
to...\n\n$pcstring"
```

```
::aba::WriteToFile "_pc_.txt" "Start test"
::aba::WriteToFile "_pc_.txt" [test Start]

sleep 60

::aba::WriteToFile "_pc_.txt" "Stop test"
::aba::WriteToFile "_pc_.txt" [test Stop]

pc RemovePathConfirmation [pathconf.toneslope.name GetValue]

app Exit
exit
```

::aba::ApiPnT

```
package require tclreadyapi

::aba::ApiApplication app
::aba::ApiPnT pnt
::aba::ApiSystemInformation sysinfo

if { 0 == [app Enter localhost] } {
  ::aba::WriteToFile "_pnt_.txt" "Unable to establish 
connection"
  exit
}

::aba::WriteToFile "_pnt_.txt" "Load environment..."
::aba::WriteToFile "_pnt_.txt" [app Load "Demo_A5K_SIP.env"]

sleep 10

::aba::WriteToFile "_pnt_.txt" "Cards in env file"
::aba::WriteToFile "_pnt_.txt" [sysinfo GetEnvCardList "DEMO_0"]

set sipsub [pnt GetCurrentConfig ICG_SIP "Subscriber"]
::aba::WriteToFile "_pnt_.txt" $sipsub
```

```
::aba::DatSetlist setlist

setlist ImportFromXMLString $sipsub
set setnum 2
set idx [::aba::FindInObjList [setlist GetList_set] "num"
$setnum]

if { $idx != -1} {

  set tmpset [setlist GetByIndex_set $idx]
  ::aba::WriteToFile "_pnt_.txt" "set $setnum info (idx $idx)"
  ::aba::WriteToFile "_pnt_.txt" [$tmpset ExportToXMLString]

  ::aba::WriteToFile "_pnt_.txt" "Update set ->"
  $tmpset.setassociation.from SetValue 34
  $tmpset.num SetValue 2

  ::aba::WriteToFile "_pnt_.txt" [pnt UpdateSet [$tmpset
ExportToXMLString]]
  ::aba::WriteToFile "_pnt_.txt" "Udate set <-"
  ::aba::WriteToFile "_pnt_.txt" [$tmpset ExportToXMLString]
}

::aba::WriteToFile "_pnt_.txt" "Remove set #1 result (0/1):
[pnt RemoveSet ICG_SIP sub 1]"

app Exit

::aba::WriteToFile "_pnt_.txt" "Terminated"

exit
```

::aba::ApiProtocolSelection

```
package require tclreadyapi
```

```
::aba::ApiApplication app

if { 0 == [app Enter localhost] } {
  ::aba::WriteToFile "_ps_.txt" "Unable to establish
connection"
  exit
}

app Load "Demo_A5K_SIP.env"

sleep 10

app Save "SYSTEM.env"

::aba::ApiProtocolSelection ps_api

set res [ps_api GetCurrentConfig]
::aba::WriteToFile "_ps_.txt" $res

::aba::DatProtocolselection ps_data

ps_data ImportFromXMLString $res

set idx [::aba::FindInObjList [ps_data.card
GetList_slotcardspec] "slot" 1]

::aba::WriteToFile "_ps_.txt" "Index for slot 1 is $idx"

if {$idx != -1} {
  set slotcardspec [ps_data.card GetByIndex_slotcardspec $idx]
  ::aba::WriteToFile "_ps_.txt" "Old signaling:"
  ::aba::WriteToFile "_ps_.txt" [$slotcardspec.signaling
GetValue]
  $slotcardspec.signaling SetValue "SKINNY"
  ::aba::WriteToFile "_ps_.txt" "New signaling:"
  ::aba::WriteToFile "_ps_.txt" [$slotcardspec.signaling
GetValue]
```

```
  set new_conf [ps_data ExportToXMLString]
  ::aba::WriteToFile "_ps_.txt" "New config is:"
  ::aba::WriteToFile "_ps_.txt" $new_conf
  ::aba::WriteToFile "_ps_.txt" "Apply new config:"
  ::aba::WriteToFile "_ps_.txt" [ps_api SetCurrentConfig
$new_conf]
}

app Exit

::aba::WriteToFile "_ps_.txt" "Terminated"

exit
```

::aba::ApiQoMThresholds

```
package require tclreadyapi

set OUTFILE {_qom_.txt}

::aba::ApiApplication app

if { 0 == [app Enter localhost] } {
  ::aba::WriteToFile $OUTFILE "Unable to establish connection"
  exit
}

::aba::ApiQoMThresholds apiQom

set cardtype ICG_SIP
set txtQom [apiQom GetCurrentConfig $cardtype]

::aba::WriteToFile $OUTFILE "$cardtype settings:"
::aba::WriteToFile $OUTFILE $txtQom

::aba::DatThresholds datQom
datQom ImportFromXMLString $txtQom
```

```
datQom.psqmthresholds.threshold SetValue 5
datQom.psqmthresholds.badframethreshold SetValue 3
datQom.psqmthresholds.badframespercent SetValue 17
datQom.psqmthresholds.syncthreshold SetValue 1

set txtQom [datQom ExportToXMLString]

::aba::WriteToFile $OUTFILE "New settings:"
::aba::WriteToFile $OUTFILE $txtQom

::aba::WriteToFile $OUTFILE "Try to update (1/0):"
::aba::WriteToFile $OUTFILE [apiQom SetCurrentConfig $txtQom]

app Exit

::aba::WriteToFile $OUTFILE "Terminated"

exit
```

::aba::ApiReports

```
package require Itcl
package require tdom

package require tclreadyapi

set OUTFILE "rge_html.txt"

::aba::ApiApplication app1

::aba::DatRge RGE
RGE.Section.type SetValue "root"
RGE.Section.name SetValue "Root"

::aba::DatSectionParametersParam RootParam1
RootParam1.name SetValue "report-format-html"
RootParam1 SetValue "Y"
RGE.Section.Parameters AddItem_Param RootParam1
```

```
::aba::DatSectionParametersParam RootParam2
RootParam2.name SetValue "report-file-name-root"
RootParam2 SetValue "rge_html"
RGE.Section.Parameters AddItem_Param RootParam2

::aba::DatSection SectionChannelSelection
SectionChannelSelection.type SetValue "channel-selection"
SectionChannelSelection.name SetValue "Channel Selection"
::aba::DatSectionChannelselection ChanSel
ChanSel.selectionmethod SetValue "All"
ChanSel.includeoriginate SetValue "YES"
ChanSel.includeterminate SetValue "YES"
ChanSel.averaged SetValue "NO"
::aba::DatSectionChannelselectionTypeside TypeSide
TypeSide.averaged SetValue "NO"
TypeSide.enabled SetValue "YES"
TypeSide.typename SetValue "SIP"
TypeSide.sidename SetValue "Sub"
::aba::DatSectionChannelselectionTypesideSet Set1
Set1.averaged SetValue "NO"
Set1.enabled SetValue "YES"
Set1.availablechannels SetValue "1-100"
Set1.number SetValue "1"
Set1.firstphysicalchannel SetValue "1"
TypeSide AddItem_Set Set1
::aba::DatSectionChannelselectionTypesideSet Set2
Set2.averaged SetValue "NO"
Set2.enabled SetValue "NO"
Set2.availablechannels SetValue "201-400"
Set2.number SetValue "2"
Set2.firstphysicalchannel SetValue "201"
TypeSide AddItem_Set Set2
ChanSel AddItem_typeside TypeSide
SectionChannelSelection AddItem_ChannelSelection ChanSel
::aba::DatSection SectionStatistics
SectionStatistics.type SetValue "statistics"
SectionStatistics.name SetValue "Statistics"
SectionChannelSelection AddItem_Section SectionStatistics
```

```
RGE.Section AddItem_Section SectionChannelSelection

::aba::WriteToFile $OUTFILE [RGE ExportToXMLString]

set chan1 [app1 Enter localhost]
::aba::WriteToFile $OUTFILE "1st instance: $chan1"

if { $chan1 != 0 } {

    ::aba::ApiTest test
    test SetChannel $chan1

    ::aba::ApiReports rep
    rep SetChannel $chan1

    test Start
    ::aba::sleep 20
    test Stop

    rep SetRGEConfig [RGE ExportToXMLString]

    ::aba::WriteToFile $OUTFILE "Check applied RGE config:"
    ::aba::WriteToFile $OUTFILE [rep GetRGEConfig]

    ::aba::WriteToFile $OUTFILE "Generate: [rep Generate]"
    rep Download "rge_html"

    app1 Exit
}

WriteToFile $ OUTFILE "Terminated"
exit
```

::aba::ApiResults

```
# Script 1: two iterations of requesting and retrieving
# measurements over time
package req tclreadyapi
```

```
set OUTFILE "motsot.txt"

::aba::ApiApplication app
::aba::ApiResults results
::aba::ApiTest test

# prepare MOT subscription request
::aba::DatMotsotsubscriber subscriber
subscriber.rttype SetValue RT_MEASUREMENT
subscriber.rtid SetValue DELAY_CL_ORIG
subscriber.aggregation SetValue 1
subscriber.granularity SetValue 1000
subscriber.range SetValue {1-200}
subscriber.cardtype SetValue "ICG_SIP"
subscriber.sidetype SetValue "Subscriber"

set chan [app Enter localhost]
if { $chan != 0 } {
  results SetChannel $chan
  test SetChannel $chan

  # subscribe with prepared request
  set id1 [results MOTSOTSubscribe [subscriber 
ExportToXMLString]]
  ::aba::WriteToFile $OUTFILE $id1
  if { 0 >= $id1 } {
    ::aba::WriteToFile $OUTFILE "Error:\n[results 
GetLastError]"
  } else {
    # prepare another MOT subscription
    subscriber.rtid SetValue DELAY_CALL_SETUP
    subscriber.granularity SetValue 800
    # subscribe with prepared request
    set id2 [results MOTSOTSubscribe [subscriber 
ExportToXMLString]]
    ::aba::WriteToFile $OUTFILE $id2
    ::aba::WriteToFile $OUTFILE "Start test: [test Start]"

    for { set i 0 } { $i < 10 } { incr i } {
      ::aba::WriteToFile $OUTFILE "Iteration $i..."
      ::aba::sleep 30
      ::aba::WriteToFile $OUTFILE "Call length: [results
MOTSOTGetData $id1]\n"
      ::aba::WriteToFile $OUTFILE "Call setup:  [results
MOTSOTGetData $id2]\n"
    }

    test StopNow
    ::aba::sleep 10
  }
  read stdin 1
  app Exit
}
exit



# Script 2: Configure and write a report
package require tclreadyapi

::aba::ApiApplication app

if { 0 == [app Enter localhost] } {
  ::aba::WriteToFile "_rge_.txt" "Unable to establish
connection"
  exit
}

::aba::WriteToFile "_results_.txt" "Loading env..."
#::aba::WriteToFile "_results_.txt" [app Load
"Demo_A5K_SIP.env"]

#sleep 10
```

```
::aba::DatRge RGE
::aba::DatSection SectionTestStatus
::aba::DatSection SectionChannelSelection
::aba::DatSection SectionMeasurementCounts
::aba::DatSectionParametersParam Param1
::aba::DatSectionParametersParam Param2

RGE.Section.type SetValue "root"
RGE.Section.name SetValue "Root"

SectionTestStatus.type SetValue "test-status-info"
SectionTestStatus.name SetValue "Test Status Information"
#::aba::WriteToFile "_results_.txt" [SectionTestStatus
ExportToXMLString]

::aba::WriteToFile "_results_.txt" [RGE.Section
GetList_Section]
set v [RGE.Section GetList_Section]
::aba::WriteToFile "_results_.txt" [llength $v]
::aba::WriteToFile "_results_.txt" [llength $[RGE.Section
GetList_Section]]
lappend [RGE.Section GetList_Section] SectionTestStatus
::aba::WriteToFile "_results_.txt" [llength $v]
::aba::WriteToFile "_results_.txt" [llength $[RGE.Section
GetList_Section]]

#::aba::WriteToFile "_results_.txt" [llength [RGE.Section
GetList_Section]]
#lappend [RGE.Section GetList_Section] SectionTestStatus
#::aba::WriteToFile "_results_.txt" [llength [RGE.Section
GetList_Section]]

SectionChannelSelection.type SetValue "channel-selection"
SectionChannelSelection.name SetValue "Channel Selection"
SectionChannelSelection.ChannelSelection.selectionmethod
SetValue "All"
```

```
SectionChannelSelection.ChannelSelection.includeoriginate
SetValue "YES"
SectionChannelSelection.ChannelSelection.includeterminate
SetValue "YES"
SectionChannelSelection.ChannelSelection.averaged SetValue "NO"
lappend RGE.Section.Section SectionChannelSelection

SectionMeasurementCounts.type SetValue "measurement-counts"
SectionMeasurementCounts.name SetValue "Measurement Counts"

Param1.name SetValue "hide-vc-output-config"
Param1 SetValue "YES"
Param2.name SetValue "hide-vc-details"
Param2 SetValue "YES"

lappend SectionMeasurementCounts.Parameters Param1
lappend SectionMeasurementCounts.Parameters Param2

lappend RGE.Section.Section SectionMeasurementCounts

::aba::WriteToFile "_results_.txt" "RGE request:"
::aba::WriteToFile "_results_.txt" [RGE ExportToXMLString]

::aba::ApiTest test

::aba::WriteToFile "_results_.txt" "Starting test..."
set startresult [test Start]
::aba::CheckIfFailed $startresult "OK" app

sleep 60

set teststatus [test GetStatus]
::aba::WriteToFileEx "_results_.txt"" "Test Status:
$teststatus"

results StartEventsLog "_results_events.log"
```

```
sleep 30

set eventsamount [results GetEventsCount]
::aba::WriteToFileEx "_results_.txt" "Amount of events:
$eventsamount"

if { $eventsamount > 0 } {
  set outvalue [results GetEvent [expr $eventsamount - 1]]
  ::aba::WriteToFileEx "_results_.txt" "Last event: $outvalue"
}

results StopEventsLog

set variances [results GetVariances 0 1]
::aba::CheckAndContinue $variances "ERROR" app
::aba::WriteToFileEx "_test1_.txt" "Variances:"
::aba::WriteToFileEx "_test1_.txt" $variances

test Stop

::aba::ApiResults res

::aba::WriteToFile "_results_.txt" "RGE response:"
set responce [results GetResults [RGE ExportToXMLString]]

::aba::DatReport Report

Report ImportFromXMLString $responce
::aba::WriteToFile "_rge_.txt" [Report ExportToXMLString]

app Exit

::aba::WriteToFile "_rge_.txt" "Terminated"

exit
```

::aba::ApiScripts

```
package require tclreadyapi

set OUTFILE "fromscript.txt"

::aba::ApiApplication app

set chan [app Enter localhost]
::aba::WriteToFile $OUTFILE "Channel $chan"

::aba::DatScriptslist lst
::aba::ApiScripts api_scr
::aba::ApiActions api_act
api_scr SetChannel $chan
api_act SetChannel $chan

lst ImportFromXMLString [api_scr GetCurrentConfig]
::aba::WriteToFile $OUTFILE [lst ExportToXMLString]

foreach script [lst GetList_script] {
    $script ImportFromXMLString [api_scr GetActions
[$script.name GetValue]]
    foreach action [$script GetList_action] {
        set xml [api_act GetAction [$action.name GetValue]]
        if { "" != $xml } {
          $action ImportFromXMLString $xml
        }
    }
}

::aba::WriteToFile $OUTFILE [lst ExportToXMLString]

app Exit
exit
```

::aba::ApiSUT

```
package require tclreadyapi
```

```
::aba::ApiApplication app

if { 0 == [app Enter localhost] } {
  ::aba::WriteToFile "sut.txt" "Unable to establish connection"
  exit
}

app Load "Demo_A5K_SIP.env"

::aba::ApiSUT sut

set sut_cfg [sut GetCurrentConfig]
::aba::WriteToFileEx "sut.txt" "Current SUT configuration"
::aba::WriteToFileEx "sut.txt" $sut_cfg

::aba::DatSutlist sut_list
sut_list ImportFromXMLString $sut_cfg

...
#Configurating dataholder named sut_list
...

set result [sut SetCurrentConfig [sut_list ExportToXMLString]]
::aba::WriteToFileEx "sut.txt" "Set SUT configuration result"
::aba::WriteToFileEx "sut.txt" $result

app Exit

exit
```

::aba::ApiSystemInformation

```
package require tclreadyapi

::aba::ApiApplication app

if { 0 == [app Enter localhost] } {
  ::aba::WriteToFile "_sysinfo_.txt" "Unable to establish
connection"
  exit
}

::aba::WriteToFile "_sysinfo_.txt" "Is Abacus ready?"
::aba::WriteToFile "_sysinfo_.txt" [app IsAbacusReady]

::aba::WriteToFile "_sysinfo_.txt" "Last error"
::aba::WriteToFile "_sysinfo_.txt" [app GetLastError]
::aba::WriteToFile "_sysinfo_.txt" ""

::aba::ApiSystemInformation sysinfo

::aba::WriteToFile "_sysinfo_.txt" "Current connection:"
::aba::WriteToFile "_sysinfo_.txt" [sysinfo
GetConnectionString]

::aba::WriteToFile "_sysinfo_.txt" "Last error"
::aba::WriteToFile "_sysinfo_.txt" [app GetLastError]
::aba::WriteToFile "_sysinfo_.txt" ""

::aba::WriteToFile "_sysinfo_.txt" [sysinfo
RemoveAllConnections]

sleep 5

::aba::WriteToFile "_sysinfo_.txt" "SetConnection"
set ip "192.168.10.53"
set password ""
set conres [sysinfo SetConnection $ip $password]
::aba::WriteToFile "_sysinfo_.txt" $conres

::aba::WriteToFile "_sysinfo_.txt" "Last error"
::aba::WriteToFile "_sysinfo_.txt" [app GetLastError]
::aba::WriteToFile "_sysinfo_.txt" ""
```

```
if {1 != $conres} then {
  ::aba::WriteToFile "_sysinfo_.txt" "ERROR: unable to connect"
} else {

  ::aba::WriteToFile "_sysinfo_.txt" "Current connection:"
  ::aba::WriteToFile "_sysinfo_.txt" [sysinfo
GetConnectionString]

  ::aba::WriteToFile "_sysinfo_.txt" "Last error"
  ::aba::WriteToFile "_sysinfo_.txt" [app GetLastError]
  ::aba::WriteToFile "_sysinfo_.txt" ""

  ::aba::WriteToFile "_sysinfo_.txt" "Add card #1"
  set addres1 [sysinfo AddCard $ip 1]
  ::aba::WriteToFile "_sysinfo_.txt" $addres1

  ::aba::WriteToFile "_sysinfo_.txt" "Last error"
  ::aba::WriteToFile "_sysinfo_.txt" [app GetLastError]
  ::aba::WriteToFile "_sysinfo_.txt" ""

  ::aba::WriteToFile "_sysinfo_.txt" "Add card #2"
  set addres2 [sysinfo AddCard $ip 2]
  ::aba::WriteToFile "_sysinfo_.txt" $addres2

  ::aba::WriteToFile "_sysinfo_.txt" "Last error"
  ::aba::WriteToFile "_sysinfo_.txt" [app GetLastError]
  ::aba::WriteToFile "_sysinfo_.txt" ""

  if {-1 == $addres1 || -1 == $addres2} then {
    ::aba::WriteToFile "_sysinfo_.txt" "ERROR: unable to add
card(s)"
  } else {

    ::aba::WriteToFile "_sysinfo_.txt" "Load environment"
    ::aba::WriteToFile "_sysinfo_.txt" [app Load
"Demo_A5K_SIP.env"]

    ::aba::WriteToFile "_sysinfo_.txt" "Current connection:"
    ::aba::WriteToFile "_sysinfo_.txt" [sysinfo
GetConnectionString]

    ::aba::WriteToFile "_sysinfo_.txt" "Last error"
    ::aba::WriteToFile "_sysinfo_.txt" [app GetLastError]
    ::aba::WriteToFile "_sysinfo_.txt" ""

    sleep 5
  }
}

app Exit

::aba::WriteToFile "_sysinfo_.txt" "Terminated"

exit
```

::aba::ApiTerminateSettings

```
# Script 1: GetCurrentConfig

package require tclreadyapi

::aba::ApiApplication app
::aba::ApiTerminateSettings term

set OUTFILE "GetTerminateCfg.txt"

set chan [app Enter $::aba::AUTOMATION_SERVER_ADDRESS]
::aba::WriteToFile $OUTFILE "Enter: $chan"

if {$chan != 0 } then {
  term SetChannel $chan
  set res [term GetCurrentConfig]
  ::aba::WriteToFile $OUTFILE "Raw answer:\n$res"
  ::aba::DatTerminatelist dat
  dat ImportFromXMLString $res
  ::aba::WriteToFile $OUTFILE "Data holder content:\n[dat
ExportToXMLString]"
}
app Exit

::aba::WriteToFile $OUTFILE "abacus completed\nSCRIPT DONE"

exit

# Script 2: SetCurrentConfig

package require tclreadyapi

::aba::ApiApplication app
::aba::ApiTerminateSettings term

set OUTFILE "SetTerminateCfg.txt"

set chan [app Enter $::aba::AUTOMATION_SERVER_ADDRESS]
::aba::WriteToFile $OUTFILE "Enter: $chan"

if {$chan != 0 } then {
  term SetChannel $chan

  set xml {
<terminate-list>
    <terminate-item type="ICG_MGCP" side="Switch">
       <item name="CalledPartyForwardCalledNumber" value="YES"/
>
       <item name="CalledPartyForwardCallingNumber" value="NO"/
>
        <item name="CalledPartyCalledBeforeOne" value="1"/>
        <item name="CalledPartyCalledBeforeTwo" value="2"/>
        <item name="CalledPartyCalledAfterOne" value="3"/>
        <item name="CalledPartyCalledAfterTwo" value="4"/>
        <item name="CalledPartyCallingBeforeOne" value="5"/>
        <item name="CalledPartyCallingBeforeTwo" value="6"/>
        <item name="CalledPartyCallingAfterOne" value="7"/>
        <item name="CalledPartyCallingAfterTwo" value="8"/>
        <item name="CallingPartyCallingMFR2Backward"
value="1..3"/>
        <item name="CallingPartyCallingLength" value="1"/>
        <item name="CallingPartyCalledMFR2Backward"
value="1..5"/>
        <item name="CallingPartyCalledLength" value="4"/>
        <item name="CallingPartyCalledMFR15RequestCallerID" value="NO"/>
        <item name="CallingPartyCalledMFR15Length" value="10"/>
        <item name="CallingPartyCallingMode" value="0"/>
        <item name="CallingPartyCalledMode" value="1"/>
        <item name="CallingDialType" value="2"/>
        <item name="CalledDialType" value="2"/>
        <item name="ResponseOffHook" value="Aux tone"/>
        <item name="ResponseHookFlash" value="Second dial tone"/
>
        <item name="ResponseDialTimeout" value="Aux tone"/>
        <item name="ResponseBusyLine" value="Busy tone"/>
        <item name="ResponsePhoneNotFound" value="Ring back"/>
        <item name="ResponseCalledPartyAlerted" value="Ring
back"/>
        <item name="ResponseOtherSideDisconnected"
value="Silence"/>
    </terminate-item>
    <terminate-item type="T1" side="Subscriber">
        <item name="OffHookRingsBefore" value="5"/>
        <item name="OffHookTimeBefore" value="30"/>
        <item name="AnswerPathConfirmationAttempts" value="2"/>
        <item name="CallerIDCheckPhone" value="YES"/>
        <item name="CallerIDCheckName" value="NO"/>
        <item name="CallerIDCheckDate" value="NO"/>
        <item name="CallerIDDetectThresholdOne" value="17"/>
        <item name="CallerIDDetectThresholdTwo" value="23"/>
        <item name="CallerIDCPEACK" value="DTMF_D"/>
        <item name="AnswerCutThrough" value="0.8"/>
        <item name="OnHookEndOfScript" value="2.5"/>
        <item name="OnHookCallProgressTones" value="1.5"/>
        <item name="OnHookPathConfirmation" value="2.1"/>
        <item name="UnexpectedDisconnect" value="0"/>
    </terminate-item>
    <terminate-item type="T1" side="Exchange">
        <item name="ReceivedDigitsVerifyCalledNumber" 
value="NO"/>
        <item name="ReceivedDigitsEndDialDelay" value="1.5"/>
        <item name="ReceivedDigitsCallingMFR2Backward" 
value="1..3"/>
        <item name="ReceivedDigitsCalledMFR2Backward" 
value="1..5"/>
        <item name="ReceivedDigitsCallingLength" value="2"/>
        <item name="ReceivedDigitsSendRingback" value="0"/>
        <item name="ReceivedDigitsCheckCallerID" value="NO"/>
        <item name="ReceivedDigitsRequestCalledID" value="NO"/>
        <item name="ReceivedDigitsMFR15Length" value="7"/>
        <item name="ReceivedDigitsCalledLength" value="7"/>
        <item name="AnswerPathConfirmationAttempts" value="2"/>
        <item name="AnswerCutThrough" value="0.8"/>
        <item name="OnHookEndOfScript" value="2.5"/>
        <item name="OnHookCallProgressTones" value="5.7"/>
        <item name="OnHookPathConfirmation" value="2.9"/>
        <item name="UnexpectedDisconnect" value="0"/>
        <item name="CallingPartyMode" value="0"/>
        <item name="CalledPartyMode" value="1"/>
        <item name="DialType" value="1"/>
        <item name="ResponseOffHook" value="Aux tone"/>
        <item name="ResponseDialTimeout" value="Busy tone"/>
        <item name="ResponseNumberGood" value="Aux tone"/>
        <item name="ResponseNumberBad" value="Busy tone"/>
    </terminate-item>
    <terminate-item type="T1" side="Switch">
       <item name="CalledPartyForwardCalledNumber" value="YES"/
>
       <item name="CalledPartyForwardCallingNumber" value="NO"/
>
        <item name="CalledPartyCalledBeforeOne" value="1"/>
        <item name="CalledPartyCalledBeforeTwo" value="2"/>
        <item name="CalledPartyCalledAfterOne" value="3"/>
        <item name="CalledPartyCalledAfterTwo" value="4"/>
        <item name="CalledPartyCallingBeforeOne" value="5"/>
        <item name="CalledPartyCallingBeforeTwo" value="6"/>
        <item name="CalledPartyCallingAfterOne" value="7"/>
        <item name="CalledPartyCallingAfterTwo" value="8"/>
        <item name="CallingPartyCallingMFR2Backward"
value="1..3"/>
        <item name="CallingPartyCallingLength" value="1"/>
        <item name="CallingPartyCalledMFR2Backward"
value="1..5"/>
        <item name="CallingPartyCalledLength" value="4"/>
        <item name="CallingPartyCalledMFR15RequestCallerID" value="NO"/>
        <item name="CallingPartyCalledMFR15Length" value="10"/>
        <item name="CallingPartyCallingMode" value="0"/>
        <item name="CallingPartyCalledMode" value="1"/>
        <item name="CallingDialType" value="0"/>
        <item name="CalledDialType" value="0"/>
        <item name="ResponseOffHook" value="Aux tone"/>
        <item name="ResponseHookFlash" value="Second dial tone"/
>
        <item name="ResponseDialTimeout" value="Aux tone"/>
        <item name="ResponseBusyLine" value="Busy tone"/>
        <item name="ResponsePhoneNotFound" value="Ring back"/>
        <item name="ResponseCalledPartyAlerted" value="Ring
back"/>
        <item name="ResponseOtherSideDisconnected"
value="Silence"/>
    </terminate-item>
    <terminate-item type="ICG_SIP" side="Subscriber">
        <item name="OffHookRingsBefore" value="1"/>
        <item name="OffHookTimeBefore" value="5"/>
        <item name="AnswerPathConfirmationAttempts" value="2"/>
        <item name="CallerIDCheckPhone" value="NO"/>
        <item name="CallerIDCheckName" value="NO"/>
        <item name="CallerIDCheckDate" value="NO"/>
        <item name="CallerIDDetectTHresholdOne" value="17"/>
        <item name="CallerIDDetectTHresholdTwo" value="23"/>
```

```
            <item name="CallerIDCPEACK" value="DTMF_D"/>
            <item name="AnswerCutThrough" value="0.8"/>
            <item name="OnHookEndOfScript" value="2.5"/>
            <item name="OnHookCallProgressTones" value="1.5"/>
            <item name="OnHookPathConfirmation" value="2.1"/>
            <item name="UnexpectedDisconnect" value="0"/>
        </terminate-item>
    </terminate-list>
      }

      set res [term SetCurrentConfig $xml]
      ::aba::WriteToFile $OUTFILE "Apply new cfg: $res"
      if { 1 != $res } {
        ::aba::WriteToFile $OUTFILE "Error:\n[term GetLastError]"
      } else {
        set res [term GetCurrentConfig]
        ::aba::WriteToFile $OUTFILE "Retrieved cfg: $res"
      }
    }
    app Exit

    ::aba::WriteToFile $OUTFILE "abacus completed\nSCRIPT DONE"

    Exit
```

::aba::ApiTest

```
package require tclreadyapi

::aba::ApiApplication app
::aba::ApiSystemInformation sysinfo
::aba::ApiTest test

if { 0 == [app Enter localhost] } {
  ::aba::WriteToFile "_test_.txt" "Unable to establish
connection"
  exit
}
```

```
sleep 10

set ip "192.168.10.53"
::aba::WriteToFile "_test_.txt" "Set connection:"
::aba::WriteToFile "_test_.txt" [sysinfo SetConnection $ip ""]

::aba::WriteToFile "_test_.txt" "Current connection:"
::aba::WriteToFile "_test_.txt" [sysinfo GetConnectionString]

::aba::WriteToFile "_test_.txt" "Load environment:"
::aba::WriteToFile "_test_.txt" [app Load "Demo_A5K_SIP.env"]
::aba::WriteToFile "_test_.txt" ""

sleep 10

::aba::WriteToFile "_test_.txt" "Current connection:"
::aba::WriteToFile "_test_.txt" [sysinfo GetConnectionString]

::aba::DatTestduration duration
duration.type SetValue "Interval"
duration.minutes SetValue 1
::aba::WriteToFile "_test_.txt" [duration ExportToXMLString]

::aba::WriteToFile "_test_.txt" "Set duration"
::aba::WriteToFile "_test_.txt" [test SetDuration [duration
ExportToXMLString]]
::aba::WriteToFile "_test_.txt" ""

::aba::WriteToFile "_test_.txt" "Start test:"
::aba::WriteToFile "_test_.txt" [test Start]
::aba::WriteToFile "_test_.txt" ""

sleep 10

::aba::WriteToFile "_test_.txt" "Test status:"
::aba::WriteToFile "_test_.txt" [test GetStatus]
::aba::WriteToFile "_test_.txt" ""
```

```
::aba::WriteToFile "_test_.txt" "Wait for test completion..."
sleep 200

::aba::WriteToFile "_test_.txt" "Test status:"
::aba::WriteToFile "_test_.txt" [test GetStatus]
::aba::WriteToFile "_test_.txt" ""

::aba::WriteToFile "_test_.txt" "Stop test:"
::aba::WriteToFile "_test_.txt" [test Stop]
::aba::WriteToFile "_test_.txt" ""

app Exit

::aba::WriteToFile "_test_.txt" "Terminated"

exit
```

::aba::ApiThresholdsErrors

```
package require tclreadyapi

set OUTFILE {_thresholds_.txt}

::aba::ApiApplication app

if { 0!= [app Enter localhost] }{

  ::aba::ApiThresholdsErrors thresholds

  set CardType T1

  set XmlThresholdsList [thresholds GetCurrentConfig]

  ::aba::DatThresholdserrorslist ClassThresholdslist

  ClassThresholdslist ImportFromXMLString $XmlThresholdsList
  set TempList [ClassThresholdslist GetList_thresholdserrors]


  set idx [::aba::FindInObjList $TempList type $CardType
function {Call generation}]
  ::aba::WriteToFile $OUTFILE "Target \"Thresholds & Errors\"
index is $idx"
  if { $idx != -1 } {
    set ClassThresholdsErrors [ClassThresholdslist
GetByIndex_thresholdserrors $idx]
    set TempList [$ClassThresholdsErrors GetList_errorresponse]
    set idx [::aba::FindInObjList $TempList errorcondition
ERR_NO_RINGBACK]
    ::aba::WriteToFile $OUTFILE "Target \"Error Response\"
index is $idx"
    if { $idx != -1 } {
      set ClassErrorResponse [$ClassThresholdsErrors
GetByIndex_errorresponse $idx]
      ::aba::WriteToFile $OUTFILE "Old config:"
      ::aba::WriteToFile $OUTFILE [$ClassErrorResponse
ExportToXMLString]
      $ClassErrorResponse.threshold SetValue 7
      ::aba::WriteToFile $OUTFILE "New config:"
      ::aba::WriteToFile $OUTFILE [$ClassThresholdsErrors
ExportToXMLString]
      set res [thresholds SetCurrentConfig
[$ClassThresholdsErrors ExportToXMLString]]
      ::aba::WriteToFile $OUTFILE "Try to update (1/0): $res"
      if { $res == 0 } {
        ::aba::WriteToFile $OUTFILE "Last error: [thresholds
GetLastError]"
      }
    }
  }
  app Exit
}

::aba::WriteToFile $OUTFILE "Terminated"

exit
```

::aba::ApiResultRecords

```
package  require  tclreadyapi
set OUTFILE "ResultRecords.txt"

::aba::ApiApplication app
::aba::ApiResultRecords results
::aba::DatProperties datProperties

set chan [app Enter localhost]

if { $chan != 0 } {
  for { set TableID $::aba::ResultsTables::TABLE_QOM_EVENTS } { 
$TableID <= $::aba::ResultsTables::TABLE_FAX_EVENTS } { incr 
TableID } {
    results SetChannel [app GetChannel]
    set RowCount [results GetRowCount $TableID]
    ::aba::WriteToFile $OUTFILE "TableID = $TableID, RowCount = 
$RowCount"
    if { -1 == $RowCount } {
        ::aba::WriteToFile $OUTFILE "Error: [results 
GetLastError]"
    }
    for { set i 0 } { $i < $RowCount } { incr i } {
        datProperties ImportFromXMLString [results GetRowData 
$TableID $i]
        ::aba::WriteToFile $OUTFILE "Row #$i\n[datProperties 
ExportToXMLString]"
    }
  }
  app Exit
}

exit
```

::aba::ApiCallTracer

```
package require Itcl
package require tdom
package require tclreadyapi

set OUTFILE "LOG_CallTracer.txt"

::aba::ApiApplication app
::aba::ApiTest test
::aba::ApiCallTracer tracer

set chan [app Enter localhost]

if { ![regexp {sock\d+} $chan] } { exit }
::aba::WriteToFile $OUTFILE "Abacus Entered Successfully"

::aba::ApiSystemInformation sysinfo
sysinfo SetConnection "192.168.10.53" ""

app Load Demo_SIP.env

set conf {
    <capture-descr>
      <ewc-params card-number="1" port-number="0" ewc-filter=""
stop-after-sec=""/>
       <str-filter></str-filter>
    </capture-descr>
}

set conf2 {
    <capture-descr>
       <sig-params type="ICG_SIP" side="Subscriber" channel="1"/
>
       <str-filter></str-filter>
    </capture-descr>
}
```

```
set  ETNow {
<capture-export-descr file="res_till_now.txt" titles-only="no"
continuously="no"/>
}

set  ECont {
<capture-export-descr file="res_continuously.txt" titles-
only="no" continuously="yes"/>
}

test Start

set trid [tracer CaptureStart $conf2]
::aba::WriteToFile $OUTFILE $trid

tracer ExportStart $trid $ECont
::aba::sleep 100
tracer ExportStop $trid

tracer ExportStart $trid $ETNow
test StopNow

tracer CaptureStop $trid

::aba::sleep 2

app Exit
::aba::WriteToFile $OUTFILE "Abacus Exit"
::aba::WriteToFile $OUTFILE "SCRIPT DONE"

exit
```

Multiple Instance Support feature

```
package require Itcl
package require tdom
package require tclreadyapi

set OUTFILE "multi-user-support.txt"

# create two applications
::aba::ApiApplication app1
::aba::ApiApplication app2
::aba::ApiApplication app3

# specify actual IP addresses of Abacus Automation server
set ip_a localhost
set ip_b 192.168.10.1

# retrieve communication channels associated with each Abacus instance served by Automation server
set chan1 [app1 Enter $ip_a]
set chan2 [app2 Enter $ip_a]
set chan3 [app2 Enter $ip_b]

::aba::WriteToFile $OUTFILE "1st instance: $chan1"
::aba::WriteToFile $OUTFILE "2nd instance: $chan2"
::aba::WriteToFile $OUTFILE "3rd instance: $chan3"

# check all communication channels to be OK
if { $chan1 != 0 && $chan2 != 0 && $chan3 != 0 } {

  # perform a call to both Abacus instances as usual; for example, take a caption
  set caption1 [app1 GetAbacusCaption]
  set caption2 [app2 GetAbacusCaption]
  set caption3 [app3 GetAbacusCaption]

  ::aba::WriteToFile $OUTFILE "1st caption: $caption1"
  ::aba::WriteToFile $OUTFILE "2nd caption: $caption2"
```

```
    ::aba::WriteToFile $OUTFILE "3rd caption: $caption3"

    # create Protocol Selection object
    ::aba::ApiProtocolSelection ps

    # link the object to first communication channel
    ps SetChannel $chan1

    # do any call as usual
    ::aba::WriteToFile $OUTFILE [ps GetCurrentConfig]

    # create PnT object
    ::aba::ApiPnT pnt

    # link the object to second communication channel
    pnt SetChannel $chan2

    # do any ::aba::ApiPnT call as usual
    ::aba::WriteToFile $OUTFILE [pnt GetCurrentConfig]

    # create System Information object
    ::aba::ApiSystemInformation sysinfo

    # link the object to third communication channel
    sysinfo SetChannel $chan3

    # do any ::aba::ApiSystemInformation call as usual
    ::aba::WriteToFile $OUTFILE [sysinfo GetConnectionString]

    # close all Abacus instances and associated communication channels
    app1 Exit
    app2 Exit
    app3 Exit
}

 exit
```

Chapter 4

Data Holders Reference

In this chapter...

Introduction

This chapter provides a description of all data holder classes referenced in *Chapter 3, "API Reference."* A more complete guide to data holder classes is provided separately in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

All data holders belong to the Abacus namespace. For more information refer to *Chapter 2, "Getting Started" "Namespace" on page 27*.

::aba::DatCardconfig Data Holder Class

::aba::DatCardconfig represents the card configuration.

::aba::Dat

::aba::DatCardconfig

Schema

```
<xs:group name="circuit-parameters">
    <xs:sequence>
      <xs:element name="law">
```

Note: For detailed schema, refer to the section on **::aba::DatCircuitparametersLaw** data holder class in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

```
      </xs:element>
      <xs:element name="impedance" type="xs:string"/>
      <xs:element name="frame" type="xs:string"/>
      <xs:element name="line" type="xs:string"/>
      <xs:element name="clock" type="xs:string"/>
    </xs:sequence>
</xs:group>

<xs:element name="card-config">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="circuit-type" type="XS_TypeEnum" minOccurs="0"/>
        <xs:element name="interface" type="XS_PhysicalInterfaceEnum"
minOccurs="0"/>
        <xs:element name="signaling" type="XS_SignalingEnum"" minOccurs="0"/>
        <xs:element name="side" type="XS_SideEnum"/>
        <xs:element name="circuits" minOccurs="0">
```

Note: For detailed schema, refer to the section on **::aba::DatCardconfigCircuits** data holder class in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

```
        </xs:element>
        <xs:choice>
```

```
                <xs:group ref="circuit-parameters"/>
                <xs:element name="codec" type="XS_CodecEnum" minOccurs="0"/>
              </xs:choice>
              <xs:element name="circuit-name" type="xs:string"/>
              <xs:element name="sut-config" type="xs:string" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="log-slot" type="xs:int" use="required"/>
            <xs:attribute name="phys-slot" type="xs:int" use="required"/>
            <xs:attribute name="hdw-type" type="XS_CardHardwareEnum"
use="required"/>
            <xs:attribute name="sc" type="xs:string" use="crequired"/>
          </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatCardconfig {
  inherit ::aba::Dat
  public variable circuittype
  public variable interface
  public variable signaling
  public variable side
  public variable circuits
  public variable law
  public variable impedance
  public variable frame
  public variable line
  public variable clock
  public variable codec
  public variable circuitname
  public variable sutconfig
  public variable logslot
  public variable physslot
  public variable hdwtype
  public variable sc
  constructor {}
  method IsInitialized {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

::aba::DatCardconfigCircuittype circuittype

This field represents the card configuration type (signaling and physical interface types). For a detailed description, see **::aba::DatCardconfigCircuittype** data holder class in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

::aba::DatCardconfigInterface interface

This field represents the card physical interface. For a detailed description, see **::aba::Dat CardconfigInterface** data holder clas in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

::aba::DatCardconfigSignaling signaling

This field represents the card signaling. For a detailed description, see **::aba::DatCardconfig Signaling** data holder class in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

::aba::DatCardconfigSide side

This field represents the card circuit side. For a detailed description, see **::aba::DatCardconfig Side** data holder class in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

::aba::DatCardconfigCircuits circuits

This field represents the card circuits list. For a detailed description, see **::aba::DatCardconfig Circuits** data holder class in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

::aba::DatCircuitparametersLaw law

This field represents the circuits law. For a detailed description, see **::aba::DatCircuit parametersLaw** data holder class in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

::aba::DatString impedance

This field represents the ac impedance of the card channels.

::aba::DatString frame

This field represents the frame format of the PCM signal.

::aba::DatString line

This field represents the line code used for the PCM signal.

::aba::DatString clock

This field represents the card circuits clock source.

::aba::DatCardconfigCodec codec

This field represents the used audio/video codec name. For a detailed description, see *“::aba::DatCardconfigCodec Data Holder Class” on page 230*.

::aba::DatString circuitname

This field represents the card circuits type name (defined by the card signaling and side types).

::aba::DatString sutconfig

This field represents the connected SUT configuration profile name.

::aba::DatString logslot

This field represents the card logical slot number.

::aba::DatString physslot

This field represents the card physical slot number.

::aba::DatXscardhardwareenum hdwtype

This field represents the card hardware type. For a detailed description, see **::aba::DatXscardhardwareenum** data holder class in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

::aba::DatString sc

This field represents the corresponding SC name.

Methods

::aba::DatCardconfig::IsInitialized { }

::aba::DatCardconfig::LoadFromFile { filename }

::aba::DatCardconfig::SaveToFile { filename }

::aba::DatCardconfig::ImportFromXML { node }

::aba::DatCardconfig::ExportToXML { parentNode }

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatCardconfigCodec Data Holder Class

::aba::DatCardconfigCodec represents the card audio/video codec name.

::aba::Dat

::aba::DatXscodecenum

  ::aba::DatCardconfigCodec

Schema

```
<xs:element name="codec" type="XS_CodecEnum" minOccurs="0"/>
```

Definition

```
itcl::class ::aba::DatCardconfigCodec {
  inherit ::aba::DatXscodecenum
  constructor {}
  method IsInitialized {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

No fields.

Methods

::aba::DatCardconfigCodec::IsInitialized { }

::aba::DatCardconfigCodec::GetValue { }

::aba::DatCardconfigCodec::SetValue { value }

::aba::DatCardconfigCodec::LoadFromFile { filename }

::aba::DatCardconfigCodec::SaveToFile { filename }

::aba::DatCardconfigCodec::ImportFromXML { node }

::aba::DatCardconfigCodec::ImportFromXMLString { xml }

::aba::DatCardconfigCodec::ExportToXML { parentNode }

::aba::DatCardconfigCodec::ExportToXMLString {}

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatChannelgroups Data Holder Class

::aba::DatChannelgroups represents the card circuits groups.

::aba::Dat

::aba::DatChannelgroups

Schema

```
<xs:element name="channel-groups">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="isdn-groups" type="GroupListType" minOccurs="0"/>
            <xs:element name="v52-groups" type="GroupListType" minOccurs="0"/>
            <xs:element name="ss7-groups" type="GroupListType" minOccurs="0"/>
            <xs:element name="slc96-groups" type="GroupListType" minOccurs="0"/>
            <xs:element name="gr303-groups" type="GroupListType" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatChannelgroups {
  inherit ::aba::Dat
  public variable isdngroups
  public variable v52groups
  public variable ss7groups
  public variable slc96groups
  public variable gr303groups
  constructor {}
  method IsInitialized {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

::aba::DatChannelgroupsIsdngroups isdngroups

This field represents the ISDN groups list. For a detailed description, refer to *::aba::DatChannelgroupsIsdngroups Data Holder Class* in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

::aba::DatChannelgroupsV52groups v52groups

This field represents the V5.2 groups list. For a detailed description, refer to *::aba::DatChannel-groupsV52groups Data Holder Class* in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

::aba::DatChannelgroupsSs7groups ss7groups

This field represents the SS7 groups list. For a detailed description, refer to *::aba::DatChannel-groupsSs7groups Data Holder Class* in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

::aba::DatChannelgroupsSlc96groups slc96groups

This field represents the SLC-96 groups list. Fora detailed description, refer to ***::aba::DatChannel-groupsSlc96groups*** Data Holder Class in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

::aba::DatChannelgroupsGr303groups gr303groups

This field represents the GR-303 groups list. Fora detailed description, refer to *::aba::DatChannel-groupsGr303groups Data Holder Class* in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

Methods

::aba::DatChannelgroups::IsInitialized { }

::aba::DatChannelgroups::GetValue { }

::aba::DatChannelgroups::SetValue { value }

::aba::DatChannelgroups::LoadFromFile { filename }

::aba::DatChannelgroups::ImportFromXML { node }

::aba::DatChannelgroups::ImportFromXMLString { xml }

::aba::DatChannelgroups::ExportToXML { parentNode }

::aba::DatChannelgroups::ExportToXMLString { xml }

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatEndpoint Data Holder Class

::aba::DatEndpoint represents one endpoint in PhoneBook Txt (EPDB).

Definition

```
itcl::class ::aba::DatEndpoint {
  public variable propertymap
  constructor {}
method SetProperty {name value}
method GetProperty {name}
method RemoveProperty {name}
method GetPropertyNames {}
}
```

Fields

::aba::DatEndpoint propertymap

This field represents the endpoint properties as an array.

Methods

::aba::DatEndpoint::SetProperty { name, value }

::aba::DatEndpoint::GetProperty { name }

::aba::DatEndpoint::RemoveProperty { name }

::aba::DatEndpoint::GetPropertyNames{ }

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatEnvstatus Data Holder Class

::aba::DatEnvstatus represents the test status.

::aba::Dat

::aba::DatXsscstatusenum

  ::aba::DatEnvstatus

Schema

<xs:element xmins:xs="http://www.w3.org/2001/XMLSchema" name="env-status" type="XS_SCStatusEnum"/>

Definition

```
itcl::class ::aba::DatEnvstatus {
  inherit ::aba::DatXsscstatusenum
  method IsInitialized {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

No fields.

Methods

::aba::DatEnvstatus::IsInitialized { }

::aba::DatEnvstatus::GetValue { }

::aba::DatEnvstatus::SetValue { value }

::aba::DatEnvstatus::LoadFromFile { filename }

::aba::DatEnvstatus::SaveToFile { filename }

::aba::DatEnvstatus::ImportFromXML { node }

::aba::DatEnvstatus::ImportFromXMLString { xml }

::aba::DatEnvstatus::ExportToXML { parentNode }

::aba::DatEnvStatus::ExportToXMLString {}

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatEPConfig4Set

::aba::DatEPConfig4Set represents a list of endpoint configurations.

Definition

```
itcl::class ::aba::DatEPConfig4Set {
  public variable eplist
  constructor {} {
method AddEndpoint {ep}
method GetEndpoints {}
method LoadFromFile {filename}
method ImportFromText {strTxt}
method SaveToFile {filename}
method ExportToText {}}
}
```

Fields

::aba::DatEPConfig4Set eplist

This field represents a list of ::aba::DatEndpoints.

Methods

::aba::DatEPConfig4Set::ImportFromText { phonebook txt }

::aba::DatEPConfig4Set::ExportToText { }

::aba::DatEPConfig4Set::LoadFromFile { filename }

::aba::DatEPConfig4Set::SaveToFile{ }

::aba::DatEPConfig4Set::AddEndpoint{ endpoint }

::aba::DatEPConfig4Set::GetEndpoints{ }

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatErrorresponse Data Holder Class

::aba::DatErrorresponse represents information about an error that occurred.

::aba::Dat

::aba::DatErrorresponse

Schema

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" name="error-
response">
      <xs:complexType>
         <c name="error-condition" type="XS_ErrorTypeEnum" use="required"/>
         <xs:attribute name="threshold" type="xs:string" use="required"/>
         <xs:attribute name="error-reaction" type="XS_ErrorReactionsEnum"
use="required"/>
         <xs:attribute name="all-channels" type="xs:boolean"/>
      </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatErrorresponse {
  inherit ::aba::Dat
  public variable errorcondition
  public variable threshold
  public variable errorreaction
  public variable allchannels
  method IsInitialized {}
  method GetValue {}
  method SetValue { value }
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

errorcondition, instance of ::aba::DatXserrortypeenum

This field represents the error type. Refer to *::aba::DatXserrortypeenum Data Holder Class in* the *Tcl Automation on Abacus Data Holder Classes Reference Guide* for a detailed description.

threshold, instance of ::aba::DatString

This field represents the corresponding threshold value. Refer to *::aba::DatString Data Holder Class in* the *Tcl Automation on Abacus Data Holder Classes Reference Guide* for a detailed description.

errorreaction, instance of ::aba::DatXserrorreactionsenum

This field represents the system reaction on the error occured type. Refer to *::aba::DatXserrorreactionsenum Data Holder Class in* the *Tcl Automation on Abacus Data Holder Classes Reference Guide* for a detailed description.

allchannels, instance of ::aba::DatString

This field represents whether the error reaction is actual for all channels. Refer to *::aba::DatString Data Holder Class in* the *Tcl Automation on Abacus Data Holder Classes Reference Guide* for a detailed description.

Methods

::aba::DatErrorresponse::IsInitialized {}

::aba::DatErrorresponse::GetValue {}

::aba::DatErrorresponse::SetValue { value }

::aba::DatErrorresponse::LoadFromFile { filename }

::aba::DatErrorresponse::SaveToFile { filename }

::aba::DatErrorresponse::ImportFromXML { node }

::aba::DatErrorresponse::ImportFromXMLString { xml }

::aba::DatErrorresponse::ExportToXML { parentNode }

::aba::DatErrorresponse::ExportToXMLString {}

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatPathconfirmationlist Data Holder Class

::aba::DatPathconfirmationlist represents the path confirmation parameters list.

::aba::Dat

::aba::DatPathconfirmationlist

Schema

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" name="path-
confirmation-list">
    <xs:complexType>
      <xs:sequence>
          <xs:group ref="PCPathConfirmationParam" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatPathconfirmationlist  {
  inherit ::aba::Dat
  public variable PCPathConfirmationParam
  constructor {}
  method IsInitialized {}
  method GetByIndex_PCPathConfirmationParam { idx }
  method GetList_PCPathConfirmationParam {}
  method AddItem_PCPathConfirmationParam { NewItem }
  method RemoveItem_PCPathConfirmationParam { ItemIndex }
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
```

Fields

PCPathConfirmationParam, list of instances

This field represents the list of parameters. Refer to *::aba::DatPathConfirmationParam Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

Methods

::aba::DatPathconfirmationlist::IsInitialized {}

::aba::DatPathconfirmationlist::GetValue {}

::aba::DatPathconfirmationlist::SetValue { value }

::aba::DatPathconfirmationlist::LoadFromFile { filename }

::aba::DatPathconfirmationlist::SaveToFile { filename }

::aba::DatPathconfirmationlist::ImportFromXML { node }

::aba::DatPathconfirmationlist::ImportFromXMLString { xml }

::aba::DatPathconfirmationlist::ExportToXML { parentNode }

::aba::DatPathconfirmationlist::ExportToXMLString {}

::aba::DatPathconfirmationlist::GetByIndex_PCPathConfirmationParam { idx }

::aba::DatPathconfirmationlist::GetList_PCPathConfirmationParam { }

::aba::DatPathconfirmationlist::AddItem_PCPathConfirmationParam { NewItem }

::aba::DatPathconfirmationlist::RemoveItem_PCPathConfirmationParam { ItemIndex }

See *"Methods" on page 279* for detailed descriptions of each common function.

::aba::DatPathconfirmationparams Data Holder Class

::aba::DatPathconfirmationparams represents the path confirmation parameters configuration.

::aba::Dat

::aba::DatPathconfirmationparams

Schema

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" name="path-
confirmation-params">
      <xs:complexType>
         <xs:sequence>
              <xs:group ref="PCPathConfirmationParam"/>
         </xs:sequence>
      </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatPathconfirmationparams  {
  inherit ::aba::Dat
  public variable callprocessing
  public variable physicaladdress
  public variable tonephone
  public variable toneslope
  public variable toneresilient
  public variable qosvoice
  public variable fax
  public variable digitalprbs
  public variable packet
  public variable data
  method IsInitialized {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

callprocessing, instance of ::aba::DatPcpathconfirmationparamCallprocessing

This field represents the path confirmation call processing configuration. Refer to *::aba::DatPcpathconfirmationparamCallprocessing Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

physicaladdress, instance of ::aba::DatPcpathconfirmationparamPhysicaladdress

This field represents the physical path confirmation. Refer to *::aba::DatPcpathconfirmationparamPhysicaladdress Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

tonephone, instance of ::aba::DatPcpathconfirmationparamTonephone

This field represents the phone path confirmation. Refer to *::aba::DatPcpathconfirmationparamTonephone Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

toneslope, instance of ::aba::DatPcpathconfirmationparamToneslope

This field represents the three-tone path confirmation. Refer to *::aba::DatPcpathconfirmationparamToneslope Data Holder Class in* the *Tcl Automation Data Holders ReferenceGuide* for a detailed description.

toneresilient, instance of ::aba::DatPcpathconfirmationparamTonere-silient

This field represents the resilient path confirmation. Refer to *::aba::DatPcpathconfirmationparamToneresilient Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

qosvoice, instance of ::aba::DatPcpathconfirmationparamQosvoice

This field represents the voice path confirmation quality of service (QoS) configuration. Refer to *::aba::DatPcpathconfirmationparamQosvoice Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

fax, instance of ::aba::DatPcpathconfirmationparamFax

This field represents the path confirmation fax configuration. Refer to *::aba::DatPcpathconfirmationparamFax Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

digitalprbs, instance of ::aba::DatPcpathconfirmationparamDigitalprbs

This field represents the path confirmation digital PRBS configuration. Refer to *::aba::DatPcpathconfirmationparamDigitalprbs Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

packet, instance of ::aba::DatPcpathconfirmationparamPacket

This field represents the packet path confirmation. Refer to *::aba::DatPcpathconfirmationparamPacket Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

data, instance of ::aba::DatPcpathconfirmationparamData

This field represents the data path confirmation type. Refer to *::aba::DatPcpathconfirmationparamData Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

Methods

::aba::DatPathconfirmationparams::IsInitialized {}

::aba::DatPathconfirmationparams::GetValue {}

::aba::DatPathconfirmationparams::SetValue { value }

::aba::DatPathconfirmationparams::LoadFromFile { filename }

::aba::DatPathconfirmationparams::SaveToFile { filename }

::aba::DatPathconfirmationparams::ImportFromXML { node }

::aba::DatPathconfirmationparams::ImportFromXMLString { xml }

::aba::DatPathconfirmationlist::ExportToXML { parentNode }

::aba::DatPathconfirmationparams::ExportToXMLString {}

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatPhonebook Data Holder Class

::aba::DatPhonebook represents the phone book.

Note: This class is deprecated. Please use the GetPhoneBookBySet, SetPhoneBookBySet, and data holder classes ::aba::DatEndpoint and ::aba::DatEPConfig4Set which support PhoneBook Txt.

::aba::Dat

::aba::DatPhonebook

Schema

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" name="phone-book"
type=”PhoneBook Type”/>
</xs:element>
```

Definition

```
itcl::class ::aba::DatPhonebook  {
  inherit ::aba::DatPhonebooktype
  method IsInitialized {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

No fields.

Methods

::aba::DatPhonebook::IsInitialized {}

::aba::DatPhonebook::GetValue {}

::aba::DatPhonebook::SetValue { value }

::aba::DatPhonebook::LoadFromFile { filename }

::aba::DatPhonebook::SaveToFile { filename }

::aba::DatPhonebook::ImportFromXML { node }

::aba::DatPhonebook::ImportFromXMLString { xml }

::aba::DatPhonebook::ExportToXML { parentNode }

::aba::DatPhonebook::ExportToXMLString {}

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatProtocolselection Data Holder Class

::aba::DatProtocolselection represents the protocol selection configuration.

::aba::Dat

::aba::DatProtocolselection

Schema

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" name="protocol-
selection">
       <xs:complexType>
          <xs:sequence>
            <xs:element name="card" minOccurs="0"/>
               <xs:complexType>
                       <xs:sequence>
                          <xs:element name="slot-card-spec" type="slot-parameters"
maxOccurs="unbounded"/>
                          </xs:sequence>
               </xs:complexType>
          </xs:element>
            <xs:element name="channels" minOccurs="0"/>
               <xs:complexType>
                       <xs:sequence>
```

```
                    <xs:element name="slot-channel-spec" maxOccurs="unbounded"/
<xs:complexType>
                      <xs:complexContent>
                      <xs:extension base="protocol-per-card-type">
                      <xs:sequence>
                      <xs:element name="use-mf-r2-protocol" type="xs:boolean"
minOccurs="0"/>
                      </xs:sequence>
                      </xs:extension>
                      </xs:complexContent>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
           </xs:complexType>
         </xs:element>
         <xs:element name="icg" minOccurs="0">
           <xs:complexType>
                <xs:sequence>
                  <xs:element name="slot-icg-spec" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:sequence>
                      <xs:element name="host" maxOccurs="4">
                      <xs:complexType>
                      <xs:choice>
                      <xs:element name="remote-port" minOccurs="0"/>
                      <xs:element name="ip-type" minOccurs="0"/>
                      <xs:element name="remote-address" minOccurs="0"/>
                      <xs:element name="remote-domain-name" minOccurs="0"/>
                      <xs:element name="sctp-protocol" minOccurs="0"/>
                      <xs:element name="signaling" minOccurs="0"/>
                      <xs:element name="ss7-protocol" minOccurs="0"/>
                      <xs:element name="local-port" minOccurs="0"/>
                      <xs:element name="rtp-port" minOccurs="0"/>
                      <xs:element name="port" maxOccurs="2">
                      <xs:complexType>
                      <xs:sequence>
                      <xs:element name="vlan-tagging-enable" minOccurs="0"/>
                      <xs:element name="chan-quantity" minOccurs="0"/>
                      <xs:element name="mac-address" minOccurs="0"/>
                      <xs:element name="ethernet-mode" minOccurs="0"/>
                      <xs:element name="dns" minOccurs="0"/>
                      <xs:element name="gateway" minOccurs="0"/>
                      <xs:element name="vlan-id" minOccurs="0"/>
                      <xs:element name="dhcp-enable" minOccurs="0"/>
                      <xs:element name="subnet-mask" minOccurs="0"/>
                      <xs:element name="local-address" minOccurs="0"/>
                      <xs:element name="prefix-length" minOccurs="0"/>
                      <xs:element name="address-v6" minOccurs="0"/>
                      <xs:element name="gateway-v6" minOccurs="0"/>
                      <xs:element name="dns-v6" minOccurs="0"/>
                      <xs:element name="local-domain-name" minOccurs="0"/>
                      </xs:sequence>
                      <xs:attribute name="num" type="xs:int" use="required"/>
                      </xs:complexType>
                      </xs:element>
                      </xs:choice>
                      <xs:attribute name="num" type="xs:int"/>
                      </xs:complexType>
                      </xs:element>
                      </xs:sequence>
                      <xs:attribute name="slot" type="xs:int"/>
                      <xs:attribute name="type" type="xs:string"/>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
           </xs:complexType>
         </xs:element>
         <xs:element name="voip" minOccurs="0">
           <xs:complexType>
                <xs:sequence>
                  <xs:element name="slot-voip-spec" type="protocol-per-card-type"
maxOccurs="unbounded"/>
                </xs:sequence>
           </xs:complexType>
         </xs:element>
         <xs:element name="sut" minOccurs="0">
           <xs:complexType>
                <xs:sequence>
                  <xs:element name="slot-sut-spec" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:sequence>
                      <xs:element name="profile" type="xs:string"/>
                      </xs:sequence>
                      <xs:attribute name="slot" type="xs:int"/>
                      <xs:attribute name="type" type="xs:string"/>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
           </xs:complexType>
         </xs:element>
        </xs:sequence>
      </xs:complexType>
</xs:element>
</xs:element>
```

Definition

```
itcl::class ::aba::DatProtocolselection {
  inherit ::aba::Dat
  public variable card
  public variable channels
  public variable icg
  public variable voip
  public variable sut
  method IsInitialized {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

card, instance of ::aba::DatProtocolselectionCard

This field represents the protocol for configuring cards. For a detailed description, refer to *::aba::DatProtocolselectionCard Data Holder Class* in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

channels, instance of ::aba::DatProtocolselectionChannels

This field represents the protocol for configuring channels. For a detailed description, refer to *::aba::DatProtocolselectionChannels Data Holder Class* in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

icg, instance of ::aba::DatProtocolselectionIcg

This field represents the protocol for configuring ICG cards. For a detailed description, refer to *::aba::DatProtocolselectionIcg Data Holder Class* in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

voip, instance of ::aba::DatProtocolselectionVoip

This field represents the protocol for configuring VoIP. For a detailed description, refer to *::aba::DatProtocolselectionVoip Data Holder Class* in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

sut, instance of ::aba::DatProtocolselectionSut

This field represents the protocol for configuring SUTs. For a detailed description, refer to *::aba::DatProtocolselectionSut Data Holder Class* in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

Methods

::aba::DatProtocolselection::IsInitialized { }

::aba::DatProtocolselection::GetValue {}

::aba::DatProtocolselection::SetValue { value }

::aba::DatProtocolselection::LoadFromFile { filename }

::aba::DatProtocolselection::SaveToFile { filename }

::aba::DatProtocolselection::ImportFromXML { node }

::aba::DatProtocolselection::ExportToXML { parentNode }

See *"Methods" on page 279* for a detailed description of each function.

::aba::DatReport Data Holder Class

::aba::DatReport represents the report with the test results.

::aba::Dat

::aba::DatReport

Schema

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" name="report">
    <xs:complexType>
      <xs:sequence>
         <xs:choice maxOccurs="unbounded">
            <xs:element ref="sys-info"/>
            <xs:element ref=protocol-settings"/>
            <xs:element ref=configuration"/>
            <xs:element ref=partition-and-timing-settings"/>
            <xs:element ref=scripts-settings"/>
            <xs:element ref="measurements-summary"/>
            <xs:element  ref="errors-summary"/>
            <xs:element ref=variances"/>
            <xs:element ref=errors-vs-time"/>
            <xs:element ref=gen-events"/>
            <xs:element ref=system-events"/>
            <xs:element ref=ss7-events"/>
```

```
                    <xs:element ref=ss7-stats"/>
                    <xs:element  name="channel-data">
                        <xs:complexType>
                           <xs:sequence>
                               <xs:choice maxOccurs="unbounded">
                                   <xs:element ref="channel-groups-config"/>
                                   <xs:element  ref="measurement-value-counts"/>
                                   <xs:element  ref="measurement-vs-time"/>
                                   <xs:element  ref="errors-vs-channels"/>
                                   <xs:element  ref="gen-stats"/>
                                   <xs:element  ref="custom-text"/>
                               </xs:choice>
                           </xs:sequence>
                           </xs:attribute name=”section-num” type=”xs:int”/>
                     </xs:complexType>
                  </xs:element >
                  <xs:element ref="header"/>
                  <xs:element ref="footer"/>
                  <xs:element ref="custom-text"/>
                  <xs:element ref="report-title"/>
                  <xs:element ref="test-status"/>
                </xs:choice>
              </xs:sequence>
              <xs:attribute name="version" type="xs:string" use="required" />
            </xs:complexType>
        </xs:element>
```

Definition

```
itcl::class ::aba::DatReport  {
  inherit ::aba::Dat
  public variable sysinfo
  public variable protocolsettings
  public variable configuration
  public variable partitionandtimingsettings
  public variable scriptsettings
  public variable measurementssummary
  public variable errorssummary
  public variable variances
  public variable errorsvstime
  public variable genevents
  public variable systemevents
  public variable ss7events
  public variable ss7stats
  public variable channeldata
  public variable header
  public variable footer
  public variable customtext
  public variable reporttitle
  public variable teststatus
  public variable version
  method IsInitialized {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

sysinfo, instance of ::aba::DatSysinfo

This field represents the report section with the system information. Refer to *::aba::DatSysInfo Data Holder Class* in the *Tcl Automation Data Holders Reference Guide* for a detailed description.

protocolsettings, instance of ::aba::DatProtocolsettings

This field represents the report section with the protocol settings. Refer to *::aba::DatProtocolsettings Data Holder Class* in the *Tcl Automation Data Holders Reference Guide* for a detailed description.

configuration, instance of ::aba::DatConfiguration

This field represents the report section with the circuits configuration. Refer to *::aba::DatConfiguration Data Holder Class* in the *Tcl Automation Data Holders Reference Guide* for a detailed description.

partitionandtimingsettings, instance of ::aba::DatPartitionandtiming-settings

This field represents the report section with the channels partition and timing settings. Refer to *::aba::DatPartitionandtimingsettings Data Holder Class* in the *Tcl Automation Data Holders Reference Guide* for a detailed description.

scriptsettings, instance of ::aba::DatScriptsettings

This field represents thereport section with the test scripts settings. Refer to *::aba::DatScriptsettings Data Holder Class* in the *Tcl Automation Data Holders Reference Guide* for a detailed description.

measurementssummary, instance of ::aba::DatMeasurementssummary

This field represents the report section with the measurements value-counts summary. Refer to *::aba::DatMeasurementssummary Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

errorssummary, instance of ::aba::DatErrorssummary

This field represents the report section with the occurred errors summary. Refer to *::aba::DatErrorssummary Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

variances, instance of ::aba::DatVariances

This field represents the report section with the test results variances. Refer to *::aba::DatVariances Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

errorsvstime, instance of ::aba::DatErrorsvstime

This field represents the report section with the test results for error vs. time. Refer to *::aba::DatErrorsvstime Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

genevents, instance of ::aba::DatGenevents

This field represents the report section with the test results for events. Refer to *::aba::DatGenevents Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

systemevents, instance of ::aba::DatSystemevents

This field represents the report section with the test results for system events. Refer to *::aba::DatSystemevents Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

ss7events, instance of ::aba::DatSs7events

This field represents the report section with the test results for SS7 system events. Refer to *::aba::DatSs7events Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

ss7stats, instance of ::aba::DatSs7stats

This field represents the report section with the test results for SS7 calls statistics. Refer to *::aba::DatSs7stats Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

channeldata, instance of ::aba::DatReportChanneldata

This field represents the report section with the channels data. Refer to *::aba::DatMReportChanneldata Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

header, instance of ::aba::DatHeader

This field represents the header in the report. Refer to *::aba::DatHeader Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

footer, instance of ::aba::DatFooter

This field represents the footer in the report. Refer to *::aba::DatFooter Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

customtext, instance of ::aba::DatCustomtext

This field represents the custom text section in the report. Refer to *::aba::DatCustomtext Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

reporttitle, instance of ::aba::DatReporttitle

This field represents the report title. Refer to *::aba::DatReporttitle Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

teststatus, instance of ::aba::DatTeststatus

This field represents the current test status. Refer to *::aba::DatTestatus Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

version, instance of ::aba::DatString

This field represents the report version number.

Methods

::aba::DatReport::IsInitialized {}

::aba::DatReport::GetValue {}

::aba::DatReport::SetValue { value }

::aba::DatReport::LoadFromFile { filename }

::aba::DatReport::SaveToFile { filename }

::aba::DatReport::ImportFromXML { node }

::aba::DatReport::ImportFromXMLString { xml }

::aba::DatReport::ExportToXML { parentNode }

::aba::DatReport::ExportToXMLString {}

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatRge Data Holder Class

::aba::DatRge represents the RGE configuration to set the results report format.

::aba::Dat

::aba::DatRge

Schema

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" name="RGE">
    <xs:complexType>
      <xs:sequence>
          <xs:element ref="Section"/>
      </xs:sequence>
      <xs:attribute name=”averaged” type=”xs:string”/>
      <xs:attribute name=”enabled” type=”xs:string”/>
      <xs:attribute name=”available-channels” type=”xs:string”/>
      <xs:attribute name=”selected-channels” type=”xs:string”/>
      <xs:attribute name=”number” type=”xs:int”/>
      <xs:attribute name=”first-physical-channel” type=”xs:int”/>
    </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatRge  {
  inherit ::aba::Dat
  public variable Section
  constructor {}
  method IsInitialized {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
```

Fields

Section, instance of ::aba::DatSection

This field represents the parent section to be included in the report. Refer to *::aba::DatSection Data Holder Class* in the *Tcl Automation Data Holders Reference Guide* for a detailed description.

Methods

::aba::DatSection::IsInitialized {}

::aba::DatSection::GetValue {}

::aba::DatSection::SetValue { value }

::aba::DatSection::LoadFromFile { filename }

::aba::DatSection::SaveToFile { filename }

::aba::DatSection::ImportFromXML { node }

::aba::DatSection::ImportFromXMLString { xml }

::aba::DatSection::ExportToXML { parentNode }

::aba::DatSection::ExportToXMLString {}

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatSccardlist Data Holder Class

::aba::DatSccardlist represents the card circuits groups.

::aba::Dat

::aba::DatSccardlist

Schema

```
<xs:element name="sc-card-list">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="card" maxOccurs="unbounded"/
                <xs:complexType>
```

```
                        <xs:attribute name="log-slot" type="xs:int" use="required"/>
                        <xs:attribute name="phys-slot" type="xs:int" use="required"/>
                        <xs:attribute name="hdw-type" type="XS_CardHardwareEnum"
use="required"/>
                   </xs:complexType>
                 </xs:element>
               </xs:sequence>
               xs:attribute name="address" type="xs:string"/
           </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatSccardlist {
  inherit ::aba::Dat
  public variable card
  public variable address
  constructor {}
  method IsInitialized {}
  method GetByIndex_card { idx }
  method GetList_card {}
  method AddItem_card { NewItem }
  method RemoveItem_card { ItemIndex }
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

card as list of ::aba::DatSccardlistCard data holder objects.

This field represents the SC cards list. For a detailed description, refer to *::aba::DatSccardlistCard Data Holder Class* in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

address, instance of ::aba::DatString

This field represents the IP adress of the SC where the card is located.

Methods

::aba::DatSccardlist::IsInitialized {}

::aba::DatSccardlist::GetValue {}

::aba::DatSccardlist::SetValue { value }

::aba::DatSccardlist::GetByIndex_card { idx }

::aba::DatSccardlist::GetList_card { }

::aba::DatSccardlist::AddItem_card { NewItem }

::aba::DatSccardlist::RemoveItem_card { ItemIndex }

::aba::DatSccardlist::LoadFromFile { filename }

::aba::DatSccardlist::SaveToFile { filename }

::aba::DatSccardlist::ImportFromXML { node }

::aba::DatSccardlist::ImportFromXMLString { xml }

::aba::DatSccardlist::ExportToXML { parentNode }

::aba::DatSccardlist::ExportToXMLString {}

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatScList Data Holder Class

::aba::DatSclist represents the list of SCs available in the active environment.

::aba::Dat

::aba::DatSclist

Schema

```
<xs:element name="sc-card-list">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="sc" type="BasicSC"/
         </xs:sequence>
      </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatSclist {
   inherit ::aba::Dat
```

```
  public variable sc
  method IsInitialized {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

sc, instance of ::aba::DatSclistSc

This field represents the SCs list. For a detailed description, refer to *::aba::DatSclistSc Data Holder Class* in the *Tcl Automation on Abacus Data Holder Classes Reference Guide*.

Methods

::aba::DatSclist::IsInitialized {}

::aba::DatSclist::GetValue {}

::aba::DatSclist::SetValue { value }

::aba::DatSclist::LoadFromFile { filename }

::aba::DatSclist::SaveToFile { filename }

::aba::DatSclist::ImportFromXML { node }

::aba::DatSclist::ImportFromXMLString { xml }

::aba::DatSclist::ExportToXML { parentNode }

::aba::DatSclist::ExportToXMLString {}

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatScstatus Data Holder Class

::aba::DatScstatus represents the SC status. The **::aba::DatScstatus** instance is a field of the **::aba::DatSysinfoGeneralsysinfoScinfo** class.

::aba::Dat

::aba::DatXsscstatusenum

  ::aba::DatScstatus

Schema

```
<xs:element name="sc-status">
      <xs:complexType>
         <xs:simpleContent>
           <xs:extension base="XS_SCStatusEnum">
         xs:attribute name="address" type="xs:string"use="required"/>
           </xs:extension>
         <xs:simpleContent>
      </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatScstatus {
  inherit ::aba::DatXsscstatusenum
  public variable address
  private variable retvalue ""
  constructor {}
  method IsInitialized {}
  method GetValue {}
  method SetValue {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

::aba::DatString address

This field represents the SC IP address.

Methods

::aba::DatScstatus::IsInitialized { }

::aba::DatScstatus::GetValue { }

::aba::DatScstatus::SetValue { value }

::aba::DatScstatus::LoadFromFile { filename }

::aba::DatScstatus::SaveToFile { filename }

::aba::DatScstatus::ImportFromXML { node }

::aba::DatScstatus::ExportToXML { parentNode }

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatScversions Data Holder Class

::aba::DatScversions represents the SC version and serial numbers. The **::aba::DatScversions** instance is a field of the **::aba::DatSysinfoGeneralsysinfoScinfo** class.

::aba::Dat

::aba::DatScversions

Schema

```
<xs:element name="sc-versions">
      <xs:complexType>
         <xs:attribute name="mac" type="xs:string"use="required"/>
         <xs:attribute name="sn" type="xs:string"use="required"/>
         <xs:attribute name="boot" type="xs:string"use="required"/>
         <xs:attribute name="hardware" type="xs:string"/>
         <xs:attribute name="software" type="xs:string"use="required"/>
         <xs:attribute name="software64" type="xs:string"/>
         <xs:attribute name="sys-boot-line" type="xs:string"use="required"/>
      </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatScversions {
  inherit ::aba::Dat
  public variable mac
  public variable sn
  public variable boot
  public variable hardware
  public variable software
  public variable software64
  public variable sysbootline
  private variable retvalue ""
  constructor {}
  method IsInitialized {}
  method GetValue {}
  method SetValue {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

::aba::DatString mac

This field represents the SC mac address.

::aba::DatString sn

This field epresents the SC serial number.

::aba::DatString boot

This field epresents the SC boot version.

::aba::DatString hardware

This field epresents the SC hardware release date.

::aba::DatString software

This field epresents the SC software version.

::aba::DatString software64

This field epresents the SC software version.

::aba::DatString sysbootline

This field epresents the SC boot initialization string.

Methods

::aba::DatScversions::IsInitialized { }

::aba::DatScversions::GetValue { }

::aba::DatScversions::SetValue { value }

::aba::DatScversions::LoadFromFile { filename }

::aba::DatScversions::SaveToFile { filename }

::aba::DatScversions::ImportFromXML { node }

::aba::DatScversions::ExportToXML { parentNode }

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatSet Data Holder Class

::aba::DataSet represents the channels set.

::aba::DatSetlistitemtype

::aba::DatSet

Schema

```
<xs:element xmins:xs="http://www.w3.org/2001/XMLSchema" name="set" >
     <xs:complexType>
       <xs:complexContent>
         <xs:extension base="SetListItemType">
           <xs:attribute name="type" type="XS_TypeEnum"use="required"/>
           <xs:attribute name="side" type="XS_SideEnum"use="required"/>
         </xs:extension>
       <xs:complexContent>
     </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatSet {
  inherit ::aba::DatSetlistitemtype
  public variable type
  public variable side
  method IsInitialized {}
  method GetValue {}
  method SetValue { value }
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

type, instance of ::aba::DatXstypeenum

This field represents the card configuration type (signaling and physical interface types) the set is associated with. Refer to *::aba::DatXstypeenum Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

side, instance of ::aba::DatXssideenum

This field represents the side of the card circuit the set is associated with. Refer to *::aba::DatXssideenum Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

Methods

::aba::DatSet::IsInitialized { }

::aba::DatSet::GetValue { }

::aba::DatSet::SetValue { value }

::aba::DatSet::LoadFromFile { filename }

::aba::DatSet::SaveToFile { filename }

::aba::DatSet::ImportFromXML { node }

::aba::DatSet::ImportFromXMLString { xml }

::aba::DatSet::ExportToXML { parentNode }

::aba::DatSet::ExportToXMLString {}

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatSetchannels Data Holder Class

::aba::DataSetchannels represents the channels set.

::aba::Dat

::aba::DatSetchannels

Schema

```
<xs:element xmins:xs="http://www.w3.org/2001/XMLSchema" name="set-channels">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
```

```
            <xs:attribute name="type" type="XS_TypeEnum" use="required"/>
            <xs:attribute name="side" type="XS_SideEnum" use="required"/>
          </xs:extension>
        <xs:simpleContent>
      </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatSetchannels {
  inherit ::aba::Dat
  public variable type
  public variable side
  method IsInitialized {}
  method GetValue {}
  method SetValue {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

type, instance of ::aba::DatXstypeenum

This field represents the card configuration type (signaling and physical interface types) the set is associated with. Refer to *::aba::DatXstypeenum Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

side, instance of ::aba::DatXssideenum

This field represents the side of the card circuit the set is associated with. Refer to *::aba::DatXssideenum Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

Methods

::aba::DatSetchannels::IsInitialized { }

::aba::DatSetchannels::GetValue { }

::aba::DatSetchannels::SetValue { value }

::aba::DatSetchannels::LoadFromFile { filename }

::aba::DatSetchannels::SaveToFile { filename }

::aba::DatSetchannels::ImportFromXML { node }

::aba::DatSetchannels::ImportFromXMLString { xml }

::aba::DatSetchannels::ExportToXML { parentNode }

::aba::DatSetchannels::ExportToXMLString {}

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatSetList Data Holder Class

::aba::DatSetList represents the channels sets list.

::aba::Dat

::aba::DatSetlist

Schema

```
<xs:element name="set-list">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="set" maxOccurs="unbounded"/>
         <xs:sequence>
            <xs:attribute name="type" type="XS_TypeEnum"use="required"/>
            <xs:attribute name="side" type="XS_SideEnum"use="required"/>
      </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatSetlist {
  inherit ::aba::Dat
  public variable set
  public variable type
  public variable side
  constructor {}
  method IsInitialized {}
  method GetByIndex_set { idx }
  method GetList_set {}
  method AddItem_set { NewItem }
  method RemoveItem_set { ItemIndex }
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

set as list of ::aba::DatSet data holder objects.

This field represents the channels sets list. For a detailed description, refer to *“::aba::DatSet Data Holder Class” on page 260*.

::aba::DatXstypeenum type

This field represents the configuration type (signaling and physical interface types) of the card the set is associated with.

::aba::DatXssideenum side

This field represents the side of the card circuit the set is associated with.

Methods

::aba::DatSetlist::IsInitialized { }

::aba::DatSetlist::GetList_set { }

::aba::DatSetlist::GetByIndex_set { idx }

::aba::DatSetlist::AddItem_set { NewItem }

::aba::DatSetlist::RemoveItem_set { ItemIndex }

::aba::DatSetlist::LoadFromFile { filename }

::aba::DatSetlist::SaveToFile { filename }

::aba::DatSetlist::ImportFromXML { node }

::aba::DatSetlist::ExportToXML { parentNode }

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatSetup Data Holder Class

::aba::DatSetup represents the global setup information from the Abacus software application (accessed through the UI by **File** | **Setup**).

::aba::Dat

::aba::DatSetup

Schema

```
<xs:element name="setup">
        <xs:complexType>
            <xs:sequence>
                <xs:element  name="general" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element  name="viewer" minOccurs="0" 
maxOccurs="unbounded">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element  name="use-default" type="xs:boolean"/
>
                                        <xs:element  name="custom-viewer-name" 
type="xs:string"/>
                                    </xs:sequence>
                                    <xs:attribute name="viewer-type" type="xs:string" 
use="required"/>
                                </xs:complexType>
                            </xs:element >
                            <xs:element  name="save-windows" type="xs:boolean"/>
                            <xs:element  name="load-windows" type="xs:boolean"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element >
                <xs:element  name="test" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element  name="on-start">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element  name="wait-mode" 
type="WaitLinksType"/>
                                        <xs:element  name="wait-time" type="xs:int"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element >
                            <xs:element  name="incremental-folders">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element  name="create" type="xs:boolean"/>
                                        <xs:element  name="append-instance"
type="xs:boolean"/>
                                        <xs:element  name="delete-old" type="xs:boolean"/>
                                        <xs:element  name="delete-after" type="xs:int"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element >
                            <xs:element  name="misc">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element  name="ignore-warnings"
type="xs:boolean"/>
                                        <xs:element  name="called-time" type="xs:boolean"/
>
                                        <xs:element  name="show-status"
type="xs:boolean"/>
                                        <xs:element  name="graceful-limit"
type="xs:boolean"/>
                                        <xs:element  name="graceful-time" type="xs:int"/>
                                        <xs:element  name="reset-in-batch"
type="xs:boolean"/>
                                        <xs:element  name="space-check"
type="xs:boolean"/>
                                        <xs:element  name="space-limit" type="xs:int"/>
                                        <xs:element  name="manual-start"
type="xs:boolean"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element >
                        </xs:sequence>
                    </xs:complexType>
                </xs:element >
                <xs:element  name="events" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element  name="elapsed-time"/>
                            <xs:element  name="real-time"/>
                            <xs:element  name="event-type"/>
                            <xs:element  name="comment"/>
                            <xs:element  name="phone"/>
                            <xs:element  name="cause"/>
                            <xs:element  name="alarm-on-error">
                                <xs:complexType>
                                    <xs:sequence>
```

```
                                        <xs:element  name="mode"
type="AlarmOnErrorMode"/>
                                        <xs:element  name="use-default" type="xs:boolean"/
>
                                        <xs:element  name="event" minOccurs="0"
maxOccurs="unbounded">
                                            <xs:complexType>
                                                <xs:attribute name="name"
type="XS_ErrortypeEnum" use="required"/>
                                                <xs:attribute name="play" type="xs:boolean"
use="optional"/>
                                            </xs:complexType>
                                        </xs:element >
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element >
                        </xs:sequence>
                    </xs:complexType>
                </xs:element >
                <xs:element  name="environments" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element  name="connection-mode"
type="ConnectionModeType"/>
                            <xs:element  name="recently-used-files-list" type="xs:int"/>
                            <xs:element  name="release-cards-on-exit" type="xs:boolean"/
>
                            <xs:element  name="save-on-exit"
type="SaveEnvChangesType"/>
                            <xs:element  name="override-env-mode"
type="XS_YesNoAsk"/>
                            <xs:element  ref="overwrite-on-load"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element >
                <xs:element  name="report" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element  name="generation">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element  name="plain"
type="ReportGenerationType"/>
                                        <xs:element  name="formatted"
type="ReportGenerationType"/>
                                        <xs:element  name="source"
type="ReportOptionsSourceType"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element >
                            <xs:element  name="measurements-over-time">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element  name="enable" type="xs:boolean"/>
                                        <xs:element  name="granularity" type="xs:int"/>
                                    </xs:sequence>
                                </xs:complexType>
                            </xs:element >
                        </xs:sequence>
                    </xs:complexType>
                </xs:element >
                <xs:element  name="directories" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element  name="scripts" type="xs:string"/>
                            <xs:element  name="path-confirmation" type="xs:string"/>
                            <xs:element  name="actions" type="xs:string"/>
                            <xs:element  name="phones" type="xs:string"/>
                            <xs:element  name="protocols" type="xs:string"/>
                            <xs:element  name="channels" type="xs:string"/>
                            <xs:element  name="results" type="xs:string"/>
                            <xs:element  name="reports" type="xs:string"/>
                            <xs:element  name="batch" type="xs:string"/>
                            <xs:element  name="environment" type="xs:string"/>
                            <xs:element  name="sounds" type="xs:string"/>
                            <xs:element  name="videos" type="xs:string"/>
                            <xs:element  name="images" type="xs:string"/>
                            <xs:element  name="audio" type="xs:string"/>
                            <xs:element  name="manual" type="xs:string"/>
                            <xs:element  name="ftp" type="xs:string"/>
                            <xs:element  name="templates" type="xs:string"/>
                            <xs:element  name="sut" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element >
            </xs:sequence>
            </xs:complexType>
            </xs:element >
```

Definition

```
itcl::class ::aba::DatSetup {
 inherit ::aba::Dat
 public variable general
 public variable test
 public variable events
 public variable environments
 public variable report
 public variable directories
 constructor {}
 method IsInitialized {} {
 method LoadFromFile { filename } {
 method SaveToFile { filename } {
 method ImportFromXML { node } {
 method ExportToXML { parentNode } {
}
```

Fields

general, instance of ::aba::DatSetupGeneral

This field represents the the file viewer and window parameters of global setup data (*General* panel of UI **Setup** dialog).

test, instance of ::aba::DatSetupTest

This field represents global test parameters for *On Start*, *Incremental folders* and *Miscellaneous* setup data (*Test* panel of UI **Setup** dialog).

events, instance of ::aba::DatSetupEvents

This field represents *Events Settings* and *Alarm on Error (PC)* setup data (*Events* panel of UI **Setup** dialog).

environments, instance of ::aba::DatSetupEnvironments

This field represents global settings pertaining to environments (*Environments* panel of UI **Setup** dialog).

report, instance of ::aba::DatSetupReport

This field represents the enable flag and time granularity parameter of the *Measurements Vs. Time* report (*Report* panel of UI **Setup** dialog).

directories, list of ::aba::DatSetupDirectories instances

This field represents the list of subdirectory names that the Abacus software uses when it compiles and runs test environments.

Methods

::aba::DatSetup::IsInitialized { }

::aba::DatSetup::LoadFromFile { filename }

::aba::DatSetup::SaveToFile { filename }

::aba::DatSetup::ImportFromXML { node }

::aba::DatSetup::ExportToXML { parentNode }

See *"Methods" on page 279* for a detailed description of each function.

::aba::DatSutlist Data Holder Class

::aba::DatSutList represents the list with the configurations of SUTs categorized by the corresponding card types.

::aba::Dat

::aba::DatSutlist

Schema

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" name="sut-list">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="sut-params" maxOccurs="unbounded"/>
      <xs:sequence>
    </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatSutlist {
  inherit ::aba::Dat
  public variable sutparams
  constructor {}
  method IsInitialized {}
  method GetValue {}
  method SetValue {}
  method GetList_sutparams {}
  method GetByIndex_sutparams { idx }
  method AddItem_sutparams { NewItem }
  method RemoveItem_sutparams { ItemIndex }
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

sutparams, list of ::aba::DatSutparams instances

This field represents the SUTs configurations list. For a detailed description, refer to *"::aba::DatSutparams Data Holder Class" on page 271*.

Methods

::aba::DatSutlist::IsInitialized { }

::aba::DatSutlist::GetValue { }

::aba::DatSutlist::SetValue { value }

::aba::DatSutlist::LoadFromFile { filename }

::aba::DatSutlist::SaveToFile { filename }

::aba::DatSutlist::ImportFromXML { node }

::aba::DatSutlist::ImportFromXMLString { xml }

::aba::DatSutlist::ExportToXML { parentNode }

::aba::DatSutlist::ExportToXMLString {}

::aba::DatSutlist::GetList_sutparams { }

::aba::DatSutlist::GetByIndex_sutparams { idx }

::aba::DatSutlist::AddItem_set { NewItem }

::aba::DatSutlist::RemoveItem_set { ItemIndex }

See *"Methods" on page 279* for detailed descriptions of each common function.

::aba::DatSutparams Data Holder Class

::aba::DatSutparams represents the configuration of the SUT connected with a specific type of card.

::aba::Dat

::aba::DatSutparams

Schema

```
<xs:element xmins:xs="http://www.w3.org/2001/XMLSchema"> name="sut-params"
      <xs:complexType>
         <xs:sequence>
           <xs:element ref="sut-settihgs">
         </xs:sequence>
         <xs:attribute name="type" type="XS_TypeEnum"use="required"/>
         <xs:attribute name="profile" type="xs.string"use="optional"/>
      </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatSutparams {
  inherit ::aba::Dat
  public variable sutsettings
  public variable type
  public variable profile
  method IsInitialized {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

sutsettings, instance of ::aba::DatSutsettings

This field represents the SUT settings. Refer to *::aba::DatSutsettings Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

type, instance of ::aba::DatXstypeenum

This field represents the card configuration type (signaling and physical interface types) the set is associated with. Refer to *::aba::DatXstypeenum Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

profile, instance of ::aba::DatString

This field represents the SUT configuration profile name.

Methods

::aba::DatSutparams::IsInitialized { }

::aba::DatSutparams::GetValue { }

::aba::DatSutparams::SetValue { value }

::aba::DatSutparams::LoadFromFile { filename }

::aba::DatSutparams::SaveToFile { filename }

::aba::DatSutparams::ImportFromXML { node }

::aba::DatSutparams::ImportFromXMLString { xml }

::aba::DatSutparams::ExportToXML { parentNode }

::aba::DatSutparams::ExportToXMLString {}

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatTestduration Data Holder Class

::aba::DatTestduration represents the test duration configuration type.

::aba::Dat

::aba::DatTestduration

Schema

```
<xs:element xmins:xs="http://www.w3.org/2001/XMLSchema"> name="test-
duration">
      <xs:complexType>
         <xs:sequence>
           <xs:element name="days" type="xs:long" minOccurs="0"/>
           <xs:element name="hours" type="xs:long" minOccurs="0"/>
           <xs:element name="minutes" type="xs:long" minOccurs="0"/>
           <xs:element name="date" type="xs:string" minOccurs="0"/>
           <xs:element name="time" type="xs:string" minOccurs="0"/>
           <xs:element name="scripts-per-channel" type="xs:long" minOccurs="0"/>
         </xs:sequence>
         <xs:attribute name="type" type="XS_TestDurationEnum"/>
      </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatTestduration {
  inherit ::aba::Dat
  public variable days
  public variable hours
  public variable minutes
  public variable date
  public variable time
  public variable scriptsperchannel
  public variable type
  method IsInitialized {}
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

days, instance of ::aba::DatString

This field represents the number of days the test will run for the “for specified time” test duration mode.

hours, instance of ::aba::DatString

This field represents the number of hours the test will run for the “for specified time” test duration mode.

minutes, instance of ::aba::DatString

This field represents the number of minutes the test will run for the “for specified time” test duration mode.

date, instance of ::aba::DatString

This field represents the date on which the test stops running, as specified in the “until” test duration mode.

time, instance of ::aba::DatString

This field represents the time at which the test stops running, as specified in the “until” test duration mode

scriptsperchannel, instance of ::aba::DatString

This field represents the number of days the test is running for the specified test duration mode.

type, instance of ::aba::DatXstestdurationenum

This field represents the the test duration configuration type. Refer to *::aba::DatXstestdurationenum Data Holder Class in* the *Tcl Automation Data Holders Reference Guide* for a detailed description.

Methods

::aba::DatTestduration::IsInitialized { }

::aba::DatTestduration::GetValue { }

::aba::DatTestduration::SetValue { value }

::aba::DatTestduration::LoadFromFile { filename }

::aba::DatTestduration::SaveToFile { filename }

::aba::DatTestduration::ImportFromXML { node }

::aba::DatTestduration::ExportToXML { parentNode }

See *“Methods” on page 279* for a detailed description of each function.

::aba::DatThresholdserrors Data Holder Class

::aba::DatThresholdserrors represents the thresholds and errors for the certain circuit type.

::aba::Dat

::aba::DatThresholdserrors

Schema

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" name="thresholds-
errors">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="error-response" minOccurs="0"
maxOccurs="unbounded"/>
         <xs:sequence>
            <xs:attribute name="type" type="XS_TypeEnum"use="required"/>
            <xs:attribute name="function" type="XS_ThresholdsErrorsFunctionEnum"/>
      </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatThresholdserrors {
  inherit ::aba::Dat
  public variable errorresponse
  public variable type
  public variable function
  constructor {}
  method IsInitialized {}
  method GetByIndex_errorresponse { idx }
  method GetList_errorresponse {}
  method AddItem_errorresponse { NewItem }
  method RemoveItem_errorresponse { ItemIndex }
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

errorresponse, list of ::aba::DatErrorresponse instances

This field represents the information about the error that occurred. Refer to *“::aba::DatErrorresponse Data Holder Class” on page 236* for a detailed description.

type, instance of ::aba::DatXstypeenum

This field represents the card signaling and physical interface types. Refer to *::aba::DatXstypeenum Data Holder Class* in the *Tcl Automation on Abacus Data Holder Classes Reference Guide* for a detailed description.

function, instance of ::aba::DatXsthresholdserrorsfunctionenum

This field represents the function for errors and timeouts specification type. Refer to *::aba::DatXsthresholdserrorsfunctionenum Data Holder Class* in the *Tcl Automation on Abacus Data Holder Classes Reference Guide* for a detailed description.

Methods

::aba::DatThresholdserrors::IsInitialized {}

::aba::DatThresholdserrors::GetValue {}

::aba::DatThresholdserrors::SetValue { value }

::aba::DatThresholdserrors::GetByIndex_errorresponse { idx }

::aba::DatThresholdserrors::GetList_errorresponse {}

::aba::DatThresholdserrors::AddItem_errorresponse { NewItem }

::aba::DatThresholdserrors::RemoveItem_errorresponse { ItemIndex }

::aba::DatThresholdserrors::LoadFromFile { filename }

::aba::DatThresholdserrors::SaveToFile { filename }

::aba::DatThresholdserrors::ImportFromXML { node }

::aba::DatThresholdserrors::ImportFromXMLString { xml }

::aba::DatThresholdserrors::ExportToXML { parentNode }

::aba::DatThresholdserrors::ExportToXMLString {}

See *"Methods" on page 279* for a detailed description of each function.

::aba::DatThresholdserrorslist Data Holder Class

::aba::DatThrsholdserrorslist represents the thresholds and errors list.

::aba::Dat

::aba::DatThresholdserrorslist

.Schema

```
<xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema" name="thresholds-
errors-list">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="thresholds-errors" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:sequence>
    </xs:complexType>
</xs:element>
```

Definition

```
itcl::class ::aba::DatThresholdserrorslist {
  inherit ::aba::Dat
  public variable thresholdserrors
  constructor {}
  method IsInitialized {}
  method GetByIndex_thresholdserrors { idx }
  method GetList_thresholdserrors {}
  method AddItem_thresholdserrors { NewItem }
  method RemoveItem_thresholdserrors { ItemIndex }
  method LoadFromFile { filename }
  method SaveToFile { filename }
  method ImportFromXML { node }
  method ExportToXML { parentNode }
}
```

Fields

thresholdserrors, list of ::aba::DatThresholdserrors instances

This field represents the thresholds and errors list. Refer to *"::aba::DatThresholdserrors Data Holder Class" on page 275* for a detailed description.

Methods

::aba::DatThresholdserrorslist::IsInitialized {}

::aba::DatThresholdserrorslist::GetValue {}

::aba::DatThresholdserrorslist::SetValue { value }

::aba::DatThresholdserrorslist::GetByIndex_thresholdserrors { idx }

::aba::DatThresholdserrorslist::GetList_thresholdserrors {}

::aba::DatThresholdserrorslist::AddItem_thresholdserrors { NewItem }

::aba::DatThresholdserrorslist::RemoveItem_thresholdserrors { ItemIndex }

::aba::DatThresholdserrorslist::LoadFromFile { filename }

::aba::DatThresholdserrorslist::SaveToFile { filename }

::aba::DatThresholdserrorslist::ImportFromXML { node }

::aba::DatThresholdserrorslist::ImportFromXMLString { xml }

::aba::DatThresholdserrorslist::ExportToXML { parentNode }

::aba::DatThresholdserrorslist::ExportToXMLString {}

See *"Methods" on page 279* for a detailed description of each function.

Methods

All data holder classes provide **::aba::DatXXX::ImportFromXMLString** methods for object initialization from an XML-marked-up string, and **::aba::DatXXX::ExportToXMLString** methods for the serialization of an object inner state to an XML-marked-up string.

> **Note:** XXX represents the name of the data holder, for example, **::aba::DatCardConfig::ImportFromXMLString** and **::aba::DatCardConfig::ExportToXMLString**.

Data holder classes that contain only variable fields provide **::aba::DatXXX::SetValue** and **::aba::DatXXX::GetValue** methods for setting and retrieving their values, respectively.

This section provides a description of methods referenced in this chapter, in alphabetical order.

::aba::DatXXX::AddItem_set { NewItem }

This method appends the specified **::aba::DatXXX** data holder element to the **::aba::DatXXX::xxx** list. For example, for the **::aba::DatSutlist** data holder class, this method appends the specified **::aba::DatSutlist** data holder element to the **::aba::DatSutlist::sutparams** list.

Parameters

`NewItem`

New list element string as an instance of the **::aba::DatXXX** data holder.

Response

No response.

::aba::DatXXX::ExportToXML { parentNode }

This method serializes the **::aba::DatXXX** data holder object to a specified string in XML format; overrides the **::aba::Dat::ExportToXML** method.

Parameters

`parentNode`

`XML DOM node.`

Response

The method returns a string in XML format compatible with the **::aba::DatXXX** data holder.

::aba::DatXXX::ExportToXMLString {}

This method serializes the **::aba::DatXXX** data holder object into an XML-marked-up string.

Parameters

No parameters.

Response

The method returns an XML-marked-up string.

::aba::DatXXX::GetByIndex_xxx { idx }

This method returns the **::aba::DatXXX** data holder element from the **::aba::DatXXX::xxx** list by the specified index. For example, for the **::aba::DatSutlist** data holder class, this method returns the **::aba::DatSutparams** data holder element from the **::aba::DatSutlist::sutparams** list by the specified index.

Parameters

`idx`

Zero-base index of the element to be returned.

Response

The method returns a string in XML format (compatible with the **::aba::DatXXX** data holder) on success, or -1 otherwise. For a detailed description, refer to the ::aba::DatXXX Data holder class.

::aba::DatXXX::GetList_xxx { }

This method of a parent data holder instance returns the corresponding object list (that is, it returns the **::aba::DatXXX::xxx** list).

Parameters

No parameters.

Response

The method returns a list of **::aba::DatXXX** data holder objects. For example, for the **::aba::DatSutparams** data holder class, this method returns the **::aba::DatSutlist::sutparams** list.

::aba::DatXXX::GetValue { }

This method returns the string with the object value. Overrides the **::aba::Dat::GetValue** method.

Parameters

No parameters.

Response

The method returns a string with the object value if the object is initialized, or an empty string otherwise.

::aba::DatXXX::ImportFromXML { node }

This method initializes the **::aba::DatXXX** data holder object from the specified string in XML format; overrides the **::aba::Dat::ImportFromXML** method.

Parameters

`node`
XML DOM node.

Response

No response.

::aba::DatXXX::ImportFromXMLString { xml }

This method initializes the **::aba::DatXXX** data holder object from the specified string in XML format.

Parameters

`node`

This is the source XML-marked-up string.

Response

No response.

::aba::DatXXX::IsInitialized { }

This method determines whether the object within all its fields is initialized.

Parameters

No parameters.

Response

The method returns **1** if the object is initialized, or **0** otherwise.

::aba::DatXXX::LoadFromFile { filename }

This method initializes the **::aba::DatXXX** data holder object from the file with the specified filename; inherits the **::aba::Dat::LoadFromFile** method.

Parameters

`filename`
Source file name.

Response

No response.

::aba::DatXXX::RemoveItem_set { ItemIndex }

This method removes an element from the **::aba::DatXXX::xxx** list by the specified index.

Parameters

`ItemIndex`

Zero-based index of the element to be removed.

Response

The method returns **1** on success, or **0** otherwise.

::aba::DatXXX::SaveToFile { filename }

This method serializes the **::aba::DatXXX** data holder object to the file with the specified filename; inherits the **::aba::Dat::SaveToFile** method.

Parameters

`filename`
Destination file name.

Response

No response.

::aba::DatXXX::SetValue { value }

This method assigns the specified value to the object. Overrides the **::aba::Dat::SetValue** method.

Parameters

`value`

This is a string with the enumeration legitimate value.

Response

No response.

Appendix A

RGE Configuration Interface

In this appendix...

General Information

An Abacus report consists of a number of report sections. Each section has its own configuration stored in XML. The sub-set of sections and section parameters included in a report are configured by the user before report generation.

Sections have a hierarchical structure. The top-level section is called the **Root** section; this section produces no data. All other sections are located within the root section.

Abacus reports can be generated in PDF, HTML and XML formats.

For information about how to generate a report based on a report template and sample script, refer to *“How Do I Generate a Report Using a Template?” on page 40*.

Report Sections

The available report sections are:

- **General Information Sections**
  - Report Title
  - Header
  - Footer
  - Test Status Info
  - Custom Text
- **Results Sections**
  - Summary Data Sections
    - Measurements Summary (Variances)
    - Errors vs. Time
    - Events
    - System Events
    - SS7 Events
    - SS7 Statistics
  - Channels Data Sections - these sections are associated with a channel selection and show data related to the configured channels and channel groups
    - Measurement Counts
    - Measurements vs. Time
    - Errors vs. Channels
    - Statistics
- **Test Configuration Sections**
  - System Information
  - Protocol Settings

- Configuration
- Partition and Timing
- Scripts

Recurring Sections

All sections except Report Title, Header, and Footer can be included into the same report several times. For example, a user can include several *Channel Selections* in one report and configure a set of channels and measurements for each of the included sections.

Order of Sections in Report

There are the following restrictions on the order of report sections:

- All channel data sections can be re-ordered only within the corresponding channel selection
- No other report sections can be placed between the channel data sections that are associated with the same *channel selection* (i.e. when moving the top level report sections, channel data sections are treated as sub-sections of one consolidated *channel selection* section)
- The order of sections in the XML defines the order of sections in the report.

Section Configuration Format

Section configuration parameters should be in the following format:

```
<Section type="<type>" name="<name>">
   <Parameters>
   section parameters (... repeat for all necessary params
   for the sections)
   </Parameters>
   <Section type="<type>" name="<name>"> (...repeat for all
   nested sections, if any)
</Section>
```

```
(...repeat for all sections that should be present in the
report.)
```

- where <**type**> (mandatory) is one of the report sections *type* constants.
- **<name>** (mandatory) is the name of the report section under which it will appear in the HTML frame. By default this name equals to the name of the section plus section's ordinal number in brackets (i.e. "Header (1)").

The “channel-selection” section has additional configuration elements, described *“Channel Selection Section Parameters” on page 307*.

Section-Type Constants

root

The root section contains all other sections. No data is defined.

report-title

This section contains the title of the report only if it is included in the report (see *"Report Parameter Types" on page 291*).This section includes configurable parameters described in *"Report Title Section Parameters" on page 298*.

In the printable report formats the section can be printed either on the first report page only or at the top of each report page.

test-status-info

This section contains general test parameters (see *"Report Parameter Types" on page 291*).

channel-selection

Refer to *"Channel Selection Section Parameters" on page 307*.

variances

This section includes the count, minimum, average, and maximum for selected measurements (same data as presented in "Variances" window on GUI). All measurements are calculated across all channels.

errors-vs-time

This section includes graphs showing how the number of errors depends on time (similar data as presented in the "Errors vs Time" window on GUI), where errors are calculated through all channels that took part in the test.

events

This section lists, in chronological order, events (errors received for: all active channels, debug messages, stopping of a channel that has reached a maximum number of errors) that occured during the test (same data as presented in "Events" window on GUI).

system-events

This section lists, in chronological order, system events (problems that indirectly affect call generation and call switching) that occured during the test (same data as presented in "System Events" window on GUI).

ss7-events

This section lists, in chronological order, SS7 events (exceptions produced by an SS7 data link) that occured during the test (same data as presented in "SS7 Events" window on GUI).

ss7-statistics

This section includes cumulative SS7 statistics for calls over virtual trunks (same data as presented in "Enhanced SS7 Statistics" window on GUI).

measurement-counts

This section includes a set of value/count graphs and tables named by the measurement they show. The graphs and tables are shown for channels that are selected by the user. Some subsets of these channels may be united into groups.

If a graph/table represents some individual channel, it shows values of the corresponding measurement for this channel. If a graph/table represents some group of channels, it shows values of the corresponding measurement averaged across all channels in the group.

measurements-vs-time

This section includes a set of measurement-over-time graphs and tables named by the measurement they show. The graphs and tables are shown for channels that are selected by the user. Some subsets of these channels may be united into groups.

errors-vs-channels

This section contains graphs and tables showing how many errors of particular types have occured on individual channels during the test. Which error types to show is configured in Errors vs. Channels section parameters. Which channel errors to show is configured in Channel Selection parameters (refer to *“Channel Selection Section Parameters” on page 307*).

statistics

This section includes tables with statistics for calls and scripts on a per channel basis, broken down by type-sides or cards (same data are presented in the "Statistics" window of the GUI). Which channels to show is configured in Channel Selection parameters (refer to *“Channel Selection Section Parameters” on page 307*).

system-info

This section contains the configuration of physical subsystems installed in Abacus.

protocols-cards

This section contains information about protocol settings made for each card installed into each System Controller (these settings are defined on the *Protocol Selection* dialog of the GUI).

configuration

This section includes settings of the System Under Test, broken down by circuit type.

partition-timing

This section includes the partitioning and linking of sets (configured in the *Partition and Timing* window of the GUI).

scripts

This section includes the contents of each script selected to run in the *Partition and Timing* window (as it is shown in the *Scripts* panel of the GUI).

header

This section contains the page header repeated on each page of a printable report. Parameters are described in *"Header Section Parameters" on page 295*.

footer

This section contains page footer repeated on each page of a printable report. Parameters are described in *"Footer Section Parameters" on page 296*

custom-text

This section contains custom text. Parameters are described in *"Custom Text Section Parameters" on page 297*.

Section Parameter Format

Single section parameter must be in the following format:

```
<Param name="<name>"><value><Param>
```

where:

<name> (mandatory) is the name of the parameter

<value> (mandatory) is the value of the parameter

Report Parameter Types

Report parameter values can be of the following types:

- boolean - "Y", "yes", "true" means true, "N", "no", "false" means false. "Y"|"N" variant is preferred.
- integer - integer number is expected
- float - floating-point number is expected
- string - string value is expected
- character - single character value is expected

Parameters Common for All Sections

If there is more than one section of a single kind in the report, then the following parameter must be added to each of these sections:

- **Parameter name:** "section-ordinal-nr"
- **Parameter value:** integer ordinal number of the section among the sections of the same kind

General Parameters of Report Generation

General parameters of report generation are stored as parameters within the root section.

Report Formats

These parameters are of boolean type (see *"Report Parameter Types"*). If the value of the parameter is true then report is corresponding format will be generated. At least one of report formats should be turned on. Only values of true are mandatory. Parameters with values of false can be omitted.

The following parameter toggles report generation in PDF format:

- **Parameter name:** "report-format-pdf"

The following parameter toggles report generation in HTML format:

- **Parameter name:** "report-format-html"

The following parameter toggles report generation in XML format:

- **Parameter name:** "report-format-xml"

Target File Name

Target report file name is controlled via the following parameter:

- **Parameter name:** "report-file-name-root"
- **Parameter value:** string (see "Report parameter types", section 4) defining the name of report file name (without directory part, without auto-generated suffix)
- **Parameter kind**: mandatory

Depending on the selected report formats and enabling/disabling the auto-generated suffix, full target report file name will be:

```
<Report directory>\<Target file name><Autogenerated
suffix>.<Extension>
```

where:

- **<Report directory> -** the directory with reports for the last/current test (equals to test results directory by default)
- **<Autogenerated suffix> -** automatically generated suffix if it's turned on.
- **<Extension> -** extension of report file depending on the selected report formats and compression options:
  - for PDF: "pdf"
  - for HTML: "html"
  - for XML : "xml"
  - for compressed PDF: "pdf.zip"
  - for compressed HTML: "html.zip"
  - for compressed XML: "xml.zip"

Auto-Generated Suffix Status

Auto-generated suffix presence in the report file name is controlled via the following parameter:

- **Parameter name:** "report-file-nr-mark"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is false

Template for Printable Report Formats

Template for printable report formats is controlled via the following parameter:

- **Parameter name:** "report-template"
- **Parameter value:** character (see *"Report Parameter Types" on page 291*), "C" for colorful reports, "B" for B&W reports
- **Parameter kind:** mandatory if any of printable report formats is selected

Automatic Generation at the End of Test

Automatic generation at the end of test is controlled via the following parameter:

- **Parameter name:** "report-auto-gen-end-of-test"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is false

Automatic Generation While Test is Running Status

Automatic generation while test is running status is controlled via the following parameter:

- **Parameter name:** "report-auto-gen-periodic"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is false

Automatic Generation While Test is Running Period

Automatic generation while test is running period is controlled via the following parameter:

- **Parameter name:** "report-auto-gen-interval-value"
- **Parameter value:** integer (see *"Report Parameter Types" on page 291*) number of minutes in the interval between automatic report generations
- **Parameter kind:** mandatory if automatic generation is enabled (see *"Automatic Generation While Test is Running Status"*)

Use Current Configuration in Batch Mode Option

Use current configuration in batch mode option is controlled via the following parameter:

- **Parameter name:** "report-use-current-config-in-batch"
- **Parameter value:** boolean (see "Report parameter types", section 4)
- **Parameter kind:** optional, default is false

Report Compression Options

Report compression parameters are of boolean type (see *"Report Parameter Types" on page 291*). If the value of the parameter is true then report in corresponding format will be compressed.

PDF Format

- **Parameter name:** "report-compress-pdf"
- **Parameter kind:** optional, default is false

HTML Format

- **Parameter name:** "report-compress-html"
- **Parameter kind:** optional, default is false

XML Format

- **Parameter name:** "report-compress-xml"
- **Parameter kind:** optional, default is false

Report Threshold Warning Status

Report threshold warning status are of boolean type (see *"Report Parameter Types" on page 291*). If the value of the parameter is true then warning for the corresponding format is enabled.

PDF Format

- **Parameter name:** "report-threshold-warn-pdf"
- **Parameter kind:** optional, default is false

HTML Format

- **Parameter name:** "report-threshold-warn-html"
- **Parameter kind:** optional, default is false

XML Format

- **Parameter name:** "report-threshold-warn-xml"
- **Parameter kind:** optional, default is false

Report Warning Threshold Values

PDF Format

- **Parameter name:** "report-threshold-value-pdf"

- **Parameter value:** integer (see *“Report Parameter Types” on page 291*) threshold number of pages
- **Parameter kind:** mandatory if report threshold is enabled for the PDF format

HTML Format

- **Parameter name:** "report-threshold-value-html"
- **Parameter value:** integer (see *“Report Parameter Types” on page 291*) threshold number of pages
- **Parameter kind:** mandatory if report threshold is enabled for the HTML format

XML Format

Parameter name: "report-threshold-value-xml"

Parameter value: float (see *“Report Parameter Types” on page 291*) threshold number of megabytes

Parameter kind: mandatory if report threshold is enabled for the XML format

Configure for the Next Test Checkbox Status

Configure for the next test checkbox status is controlled via the following parameter:

- **Parameter name:** "report-configuration-source"
- **Parameter value:** boolean (see *“Report Parameter Types” on page 291*)
- **Parameter kind:** optional, default is false

Header Section Parameters

Header Text Source

Header text source is controlled via the following parameter:

- **Parameter name:** "choose"
- **Parameter value:** character (see *“Report Parameter Types” on page 291*), "F" for file, "T" for text
- **Parameter kind:** mandatory

Header Text File Path

Header text file path is controlled via the following parameter:

- **Parameter name:** "file-name"
- **Parameter value:** string (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** mandatory if file source for header text is selected

Header Text Value

Header text value is controlled via the following parameter:

- **Parameter name:** "text"
- **Parameter value:** string (see *"Report Parameter Types" on page 291*4)
- **Parameter kind:** mandatory if user-provided text source for header text is selected

Footer Section Parameters

Footer Text Source

Footer text source is controlled via the following parameter:

- **Parameter name:** "choose"
- **Parameter value:** character (see *"Report Parameter Types" on page 291*), "F" for file, "T" for text
- **Parameter kind:** mandatory

Footer Text File Path

Footer text file path is controlled via the following parameter:

- **Parameter name:** "file-name"
- **Parameter value:** string (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** mandatory if file source for footer text is selected

Footer Text Value

Footer text value is controlled via the following parameter:

- **Parameter name:** "text"
- **Parameter value:** string (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** mandatory if user-provided text source for footer text is selected

Custom Text Section Parameters

Custom Text Source

Custom text source is controlled via the following parameter:

- **Parameter name:** "choose"
- **Parameter value:** character (see *"Report Parameter Types" on page 291*), "F" for file, "T" for text
- **Parameter kind:** mandatory

Custom Text File Path

Custom text file path is controlled via the following parameter:

- **Parameter name:** "file-name"
- **Parameter value:** string (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** mandatory if file source for custom text is selected

Custom Text Value

Custom text value is controlled via the following parameter:

- **Parameter name:** "text"
- **Parameter value:** string (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** mandatory if user-provided source for custom text is selected

Insert Page Break Before Custom Text Option

Insert page break before custom text option is controlled via the following parameter:

- **Parameter name:** "break-before"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is false

Insert Page Break After Custom Text Option

Insert page break after custom text option is controlled via the following parameter:

- **Parameter name:** "break-after"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is false

Report Title Section Parameters

Company

Company value is controlled via the following parameter:

- **Parameter name:** "title-company"
- Parameter value: string (see *"Report Parameter Types" on page 291*)
- Parameter kind: optional, default is empty

Test Title

Test title is controlled via the following parameter:

- **Parameter name:** "title-test-title"
- **Parameter value:** string (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is empty

User Name

User name is controlled via the following parameter:

- **Parameter name:** "title-user-name"
- **Parameter value:** string (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is empty

Logo Type

Logo type is controlled via the following parameter:

- **Parameter name:** "logo-choose"
- **Parameter value:** character (see *"Report Parameter Types" on page 291*), "S" for Spirent logo, "C" for custom logo
- **Parameter kind:** optional, default is "S"

Logo File Path

Logo file path is controlled via the following parameter:

- **Parameter name:** "logo-file-name"
- **Parameter value**: string (see *"Report Parameter Types" on page 291*), full path to custom logo
- **Parameter kind:** mandatory if custom logo is selected

The end-user may specify a logo file for custom reports in one of following ways:

- Using the full path from a drive root.

  For example, **C:\My Logos\ColoredLogo.bmp**

  Note: That means “drive C:” on the same PC on which the Abacus software is installed (server), not a Tcl-running PC (the client may be running Windows or Linux, as well).

- Using a relative path:

  ..\My Logos\ColoredLogo.bmp

  Note: This also means the same PC on which the Abacus software is installed (server). The difference is: the relative path is applied to a folder where Abacus is installed. So, if Abacus is installed into the folder named “D:\Abacus 5000\5.20\”, the logo file is intended to be located at “**D:\Abacus 5000\My Logos\ColoredLogo.bmp**”

There is no need to specify any extra parameters to let Abacus recognize relative or full paths used for the logos. The recognition is performed automatically.

Print Date of Report Generation Option

Print date of report generation option is controlled via the following parameter:

- **Parameter name:** "date-mark"
- **Parameter value:** boolean (see *“Report Parameter Types” on page 291*)
- **Parameter kind:** optional, default is false

Report Title Print Place

Report title print place is controlled via the following parameter:

- **Parameter name:** "print-place"
- **Parameter value:** character (see *“Report Parameter Types” on page 291*), "1" (one) for print on top, "E" for print on every page
- **Parameter kind:** optional, default is "1"

System Information Section Parameters

Detail Level

Detail level is controlled via the following parameter:

- **Parameter name:** "detail-level"

- **Parameter value:** string (see *“Report Parameter Types” on page 291*), "brief" for brief, "full" for full
- **Parameter kind:** optional, default is "brief"

Configuration Section Parameters

At least one of section's parameters described below should be provided or the section won't be included into the report.

Include SUT settings subsection option

Include SUT settings subsection option is controlled via the following parameter:

- **Parameter name:** "sut-settings"
- **Parameter value:** boolean (see *“Report Parameter Types” on page 291*4)
- **Parameter kind:** optional, default is false

Include Channels Subsection Option

Include channels subsection option is controlled via the following parameter:

- **Parameter name:** "channels"
- **Parameter value:** boolean (see *“Report Parameter Types” on page 291*)
- **Parameter kind:** optional, default is false

Include Phone Books Subsection Option

Include phone books subsection option is controlled via the following parameter:

- **Parameter name:** "phone-books"
- **Parameter value:** boolean (see *“Report Parameter Types” on page 291*)
- **Parameter kind:** optional, default is false

Variances Section Parameters

Measurements to Print

The following parameter controls measurement's presence in the report for the section:

- **Parameter name:** "variance-print-<measurement-num>"
- **Parameter value:** boolean (see *“Report Parameter Types” on page 291*)

- **Parameter kind:** optional, default is false

where

<measurement-num> is the ordinal number of TMeasurementType (remember that LEVEL_FIRST_EC_ENABLED, LEVEL_FIRST_EC_DISABLED, LEVEL_SECOND_EC_ENABLED, LEVEL_SECOND_EC_DISABLED, LEVEL_THIRD_EC_ENABLED, LEVEL_THIRD_EC_DISABLED are never included in a report).

*Table A-1*shows ordinal numbers and corresponding **TMeasurementType** constants.

Table A-1. Measurement Types and Ordinal Numbers

measurement-num	TMeasurementType
0	DELAY_DIALTONE
1	DELAY_TONE
2	DELAY_SILENCE
3	DELAY_ENERGY
4	DELAY_PATHCONF
5	DELAY_STRING
6	DELAY_ROUNDTRIP
7	DELAY_HIT_CNFR
8	DELAY_CLIPPING
9	DELAY_CL_TERM
10	DELAY_CL_ORIG
11	DELAY_USER_TIMER
12	DELAY_RTP_JITTER
13	DELAY_RESPONSE_TIME
14	DELAY_CALL_SETUP
15	DELAY_TEAR_DOWN
16	DELAY_POST_DIAL
17	GRAPH_BYTE_ERR_RATE
18	GUI_DELAY_MOS

Table A-1. Measurement Types and Ordinal Numbers (continued)

measurement-num	TMeasurementType
19	GRAPH_PSQM
20	GRAPH_PESQ
21	GRAPH_PESQ_MOS_LQO
22	GRAPH_RFACTOR
23	GRAPH_RFACTORG107
24	GRAPH_JMOS
25	CG_GRAPH_PCPK
26	GRAPH_ONE_WAY_DELAY
27	GRAPH_JITTER
28	GRAPH_RTP_PACKET_LOSS
29	GRAPH_RTP_PACKETS_OUT_OF_ORDER
30	GRAPH_RTP_PACKETS_TOO_LATE
31	DELAY_FAX_CONNECT_RATE
32	DELAY_FAX_TRANSFER_RATE
33	GRAPH_DAT_MODEM_CONNECT_SPD_CLI_TX
34	GRAPH_DAT_MODEM_CONNECT_SPD_CLI_RX
35	GRAPH_DAT_MODEM_CONNECT_SPD_SRV_TX
36	GRAPH_DAT_MODEM_CONNECT_SPD_SRV_RX
37	GRAPH_DAT_MODEM_THROUGHPUT_SPD_CLI
38	GRAPH_DAT_MODEM_THROUGHPUT_SPD_SRV
39	DELAY_FAX_LINE_ERR_RATE
40	DELAY_MODEM_BYTE_ERR_RATE
41	DELAY_PING_ROUNDTRIP
42	DELAY_PING_LOSS_RATE

Table A-1. Measurement Types and Ordinal Numbers (continued)

measurement-num	TMeasurementType
43	GRAPH_DAT_FTP_DOWNLOAD_THROUGHPUT_CLI
44	GRAPH_DAT_FTP_UPLOAD_THROUGHPUT_CLI
45	GRAPH_DAT_FTP_DOWNLOAD_THROUGHPUT_SRV
46	GRAPH_DAT_FTP_UPLOAD_THROUGHPUT_SRV
47	DELAY_MODEM_CONNECT_TIME
48	DELAY_FAX_CONNECT_TIME
49	DELAY_RRQ_RESPONSE_TIME
50	DELAY_FAX_PAGE_RECEIVED
51	DELAY_T38_PKT_SENT
52	DELAY_T38_PKT_RCVD
53	DELAY_T38_SESSION_DUR
54	DELAY_T38_AVG_TX_RATE
55	DELAY_T38_AVG_RX_RATE
56	DELAY_FIRST_EC_ENABLED
57	LEVEL_FIRST_EC_ENABLED *
58	DELAY_FIRST_EC_DISABLED
59	LEVEL_FIRST_EC_DISABLED *
60	GRAPH_FIRST_ERL_EC_ENABLED
61	GRAPH_FIRST_ERL_EC_DISABLED
62	GRAPH_FIRST_ERLE
63	GRAPH_FIRST_TELR
64	DELAY_SECOND_EC_ENABLED
65	LEVEL_SECOND_EC_ENABLED *
66	GRAPH_SECOND_ERL_EC_ENABLED

Table A-1. Measurement Types and Ordinal Numbers (continued)

measurement-num	TMeasurementType
67	DELAY_SECOND_EC_DISABLED
68	LEVEL_SECOND_EC_DISABLED *
69	GRAPH_SECOND_ERL_EC_DISABLED
70	DELAY_THIRD_EC_ENABLED
71	LEVEL_THIRD_EC_ENABLED*
72	GRAPH_THIRD_ERL_EC_ENABLED
73	DELAY_THIRD_EC_DISABLED
74	LEVEL_THIRD_EC_DISABLED *
75	GRAPH_THIRD_ERL_EC_DISABLED
76	DELAY_DISCONNECT
77	GRAPH_RTP_PACKETS_RX
78	GRAPH_RTP_PACKETS_TX
79	GRAPH_RTP_PACKETS_TX_DROP_OOM
80	GRAPH_RTP_PACKETS_TX_DROP_OLOAD
81	DELAY_REG_4XX_RESPONSE_TIME
82	DELAY_REG_200_RESPONSE_TIME
83	DELAY_REG_SUCCESS_TIME
84	GRAPH_RTP_PACKET_LOSS_RTCP
85	DELAY_RTP_JITTER_RTCP
86	GRAPH_RTP_LOSS_RATE
87	DELAY_LUA_CUSTOM

* These measurement types cannot be used in reports.

Confidence Limits

The following parameter controls presence of confidence limits for the measurement:

- **Parameter name:** "variance-limit-<measurement-num>"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is false

where

<measurement-num> is the ordinal number of TMeasurementType type defined in *Table A-1 on page 301* (remember that LEVEL_FIRST_EC_ENABLED, LEVEL_FIRST_EC_DISABLED, LEVEL_SECOND_EC_ENABLED, LEVEL_SECOND_EC_DISABLED, LEVEL_THIRD_EC_ENABLED, LEVEL_THIRD_EC_DISABLED are never included in a report).

Errors vs. Time Section Parameters

Graph Style

The following parameter controls graph style:

- **Parameter name:** "graph-style"
- **Parameter value:** string (see *"Report Parameter Types" on page 291*) "Line" for line graph, "Bar" for bar graph, "Area" for area
- **Parameter kind:** mandatory

Time Resolution

The following parameter controls time resolution value:

- **Parameter name:** "time-resolution-seconds"
- **Parameter value:** integer (see *"Report Parameter Types" on page 291*) number of seconds
- **Parameter kind:** mandatory

Report Time Range Start

The following parameter controls report time range start:

- **Parameter name:** "start-time"
- **Parameter value:** float (see *"Report Parameter Types" on page 291*) TDateTime Delphi value, absolute time of time range start
- **Parameter kind:** mandatory

Report Time Range End

The following parameter controls report time range end:

- **Parameter name:** "end-time"
- **Parameter value:** float (see *"Report Parameter Types" on page 291*) TDateTime Delphi value, absolute time of time range end
- **Parameter kind:** optional, default is test stop time if test is finished or time of report generation start if test is running

Events Section Parameters

Important: At least one of "Show elapsed time" and "Show real time" options should be turned on.

Show Elapsed Time in Section Option

Show elapsed time in section option is controlled via the following parameter:

- **Parameter name:** "show-elapsed-time"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** default is false

Show Real Time in Section Option

Show real time in section option is controlled via the following parameter:

- **Parameter name:** "show-real-time"
- **Parameter value:** boolean (see "*"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is false

Show Event Type Option

Show event type option is controlled via the following parameter:

- **Parameter name:** "show-event-type"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is false

Show Event Comment Option

Show event comment option is controlled via the following parameter:

- **Parameter name:** "show-comment"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is false

Show Event Phone Option

Show event phone option is controlled via the following parameter:

- **Parameter name:** "show-phone"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is false

Show Event Cause Option

Show event cause option is controlled via the following parameter:

- **Parameter name:** "show-cause"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*, section 4)
- **Parameter kind:** optional, default is false

Channel Selection Section Parameters

Channel selection configuration has the following format:

```
    <Section name="channel-selection" name="any string">
        <ChannelSelection averaged="<averaged>" include-
originate="<include-originate>"
            include-terminate="<include-terminate>" selection-
method="<selection method>"
         channel selection configuration (depend on <selection-
method> attribute).
        </ChannelSelection>
        ...nested sections
    </Section>
```

where:

- **<averaged>** If <selection method> is "All" this attribute specifies whether to aggregate over all channels in channel selection. Otherwise it has no meaning and is optional. Possible values: "YES" or "NO" (case-insensitive)

- **<include-originate>** Specifies if originate channels should be included in the channel selection. Possible values: "YES" or "NO" (case insensitive). Default value is "NO".
- **<include-terminate>** Specifies if terminate channels should be included in the channel selection. Possible values: "YES" or "NO" (case insensitive). Default value is "NO".
- **<selection method>** Specifies channel selection method. Possible values: "All", "Typesides", "Cards". The attribute is mandatory. If attribute value is "All" then all inner elements of <ChannelSelection> can be omitted.

Set of Channels Description

In all channel selection attributes where set of channels should be specified, it is done in the following format:

```
    <number>[-<number>]{,<number>[-<number>]}*
```

Valid examples: "1-5", "1-5,6-8,9-12", "1-5,6,12,18".

By-Type-Side Channel Selection Parameters

If selection-method is "Typesides" the following structure is expected:

```
    <type-side averaged="<averaged>" enabled="<enabled>" side-
name="<side-name>"
                  type-name="<type-name>">
        <Set available-channels="<available-channels>"
averaged="<set-averaged>" enabled="<set-enabled>"
                first-physical-channel="<first-phys-channel>"
number="<number>"
                selected-channels="<selected-channels>"/>
            ..... repeat for every set ...
        <sc-ref averaged="<sc-averaged>" enabled="<sc-enabled>"
ip-address="<ip-address>"/>
            ..... repeat for every SC ...
    </type-side>
        ...repeat for every typeside.
```

Type-Side Element

- **<averaged>** Specifies if data should be aggregated over all channels in this typeside. Possible values: "YES" or "NO" (case insensitive). The attribute is mandatory.
- **<enabled>** Should be "YES", mandatory
- **<side-name>** Mandatory. Should be one of "Sub", "Ex", "Sw" which is a shortcut for XS_SideEnum enumeration of SharedSchema.xsd (refer to ::aba::DatXssideenum Data Holder Class in the *Tcl Automation on Abacus Data Holders Reference Guide)*.

- **<type-name>** Mandatory. Should be one of 'T1', 'E1', 'PRI T1', 'PRI E1', 'SLC-96 I', 'GR-303', 'BRI', 'V5.1', 'V5.2','T1 ClrCh', 'E1 ClrCh', 'Analog', 'SLC-96 II', 'V5.2 BRI', 'N/A', 'E1 Q.50', 'V5.1 BRI', 'T1 SS5', 'E1 SS5','V5.2 PRI', 'VoIP ClrCh', 'SigM3UA', 'T1 SS7', 'E1 SS7','SIP', 'MGCP', 'H323', 'VRG', 'MEGACO','SIP-T', 'SKINNY' which are the aliases for XS_TypeEnum enumeration of SharedSchema.xsd (refer to ::aba::DatXstypeenum Data Holder Class in the *Tcl Automation on Abacus Data Holders Reference Guide*).

Set Element

- **<available-channels>** Mandatory. Available channels in the set (see "Set of channels description", section 16.1). Before generating a report an intersection between available and selected channels is computed.
- **<selected-channels>** Mandatory. Channels in the set that are selected to be included in the report (see *"Set of Channels Description" on page 308*)
- **<set-averaged>** Mandatory. Specifies if data should be aggregated over all channels in this set. Possible values: "YES" or "NO" (case insensitive).
- **<set-enabled>** Specifies if the set is included in the report. Possible values are "YES" or "NO" (case insensitive).
- **<first-phys-channel>** Specifies the number first physical channel for the set. Mandatory. It is used to determine the system controller for a set. If this attribute is omitted whole set is excluded from channel selection as invalid.
- **<number>** Mandatory. Integer ordinal number of set.

Sc-Ref Element

- **<sc-averaged>** Mandatory. Specifies if data should be aggregated over all channels in this SC. Possible values: "YES" or "NO" (case insensitive).
- **<sc-enabled>** Specifies if the SC is included in the report. Possible values are "YES" or "NO" (case insensitive).
- **<ip-address>** Mandatory. IP address of the SC.

By-Cards Channel Selection Parameters

If selection-method is "Cards" the following structure is expected:

```
    <system-controller averaged="<sc-averaged>"
enabled="<sc-enabled>" ip-address="<ip-address>"
        system-name="<sc-system-name>">
        <card averaged="<card-averaged>"
enabled="<card-enabled>"
```

```
                available-channels="<available-channels>"
selected-channels="<selected-channels>"
                card-name="<card-name>" logical-slot="<log-
slot>" physical-slot="<phys-slot>"
                channels-count="<chan-count>" start-
channel="<start-channel>"
                end-channel="<end-channel>" signaling-
type="<sig-type>"/>
        ...repeat for every card that resides in the SC
    </system-controller>
    ...repeat for every SC
```

System-Controller Element

- **<sc-averaged>** Mandatory. Specifies if data should be aggregated over all channels in this SC. Possible values: "YES" or "NO" (case insensitive).
- **<sc-enabled>** Specifies if the SC is included in the report. Possible values are "YES" or "NO" (case insensitive).
- **<ip-address>** Mandatory. IP address of the SC.
- **<system-name>** Optional. If omitted or empty, system name will be set to 'ip-address' everywhere it is required.

Card Element

- **<card-averaged>** Mandatory. Specifies if data should be aggregated over all channels on this card. Possible values: "YES" or "NO" (case insensitive).
- **<card-enabled>** Specifies if the card is included in the report. Possible values are "YES" or "NO" (case insensitive).
- **<available-channels>** Mandatory. Available channels in the set (see "*"Set of Channels Description" on page 308*). Before generating a report an intersection between available and selected channels is computed.
- **<selected-channels>** Mandatory. Channels in the set that are selected to be included in the report (see *"Set of Channels Description" on page 308*)
- **<card-name>** Mandatory. One of XS_CardHardwareEnum from SharedSchema.xsd. Refer to ::aba::DatXscardhardwareenum Data Holder Class in the *Tcl Automation on Abacus Data Holders Reference Guide.*
- **<log-slot>** Mandatory. Integer number of logical slot for the card.
- **<phys-slot>** Mandatory. Integer number of physical slot for the card.
- **<chan-count>** Mandatory. Integer number of available channels on the card.

- **<start-channel>** Mandatory. Integer number of starting channel of the card.
- **<end-channel>** Mandatory. Integer number of end channel of the card.
- **<sig-type>** Mandatory. Ordinal number of corresponding PCardType (signaling) type value.

Table A-2 shows the ordinal numbers for PCardTypes .

Table A-2. Signaling Types and Ordinal Numbers

sig-type	PCardType
0	ctPRG
1	ctT1
2	ctE1
3	ctPRI1544
4	ctPRI2048
5	ctSLC96_I
6	ctGR303
7	ctBRI
8	ctV5_1
9	ctV5_2
10	ctT1ClrCh
11	ctE1ClrCh
12	ctEAnalog
13	ctSLC96_II
14	ctV5_2_BRI
15	ctEAnalogClrCh
16	ctNTD
17	ctV5_1_BRI
18	ctSIGIUA
19	ctNotUsed19

Table A-2. Signaling Types and Ordinal Numbers (continued)

sig-type	PCardType
20	ctV5_2_PRI
21	ctVoIPClrCh
22	ctSIGM3UA
23	ctT1_SS7
24	ctE1_SS7
25	ctICG_SIP
26	ctICG_MGCP
27	ctICG_H323
28	ctVRG
29	ctICG_MEGACO
30	ctICG_SIP_T
31	ctICG_SKINNY
32	ctT1_BICC
33	ctE1_BICC
34	ctBICC_over_IP

Measurement Counts Section Parameters

This section must be a subsection to "Channels Selection" (see *“Channel Selection Section Parameters” on page 307*).

Single Measurement Parameters

- **Parameter name:** "meas-<measurement-num>"
- **Parameter value:** <value>
- **Parameter kind:** optional, measurement is not present in the report if the parameter is omitted

where

- **<measurement-num>** is the ordinal number of TMeasurementType type defined in *Table A-1 on page 301* (remember that LEVEL_FIRST_EC_ENABLED, LEVEL_FIRST_EC_DISABLED, LEVEL_SECOND_EC_ENABLED, LEVEL_SECOND_EC_DISABLED, LEVEL_THIRD_EC_ENABLED, LEVEL_THIRD_EC_DISABLED are never included in a report).

- **<value>** is a string in the form of "1,0,1,0,1" (five digits separated by commas)
    - first digit specifies whether to include graph for this measurement (1 - yes, 0 - no)
    - second digit specifies whether to include table for this measurement (1 - yes, 0 - no)
    - third digit specifies whether to include graph mark for this measurement (1 - yes, 0 - no)
    - fourth digit specifies mark value (floating-point). is ignored if third digit is 0.
    - fifth digit specifies graph type for the graph (-1 - none, 0 - line, 1 - bar).

Hide Output Config Option

The following parameter controls absence of vc-output-config element in the report:

- **Parameter name:** "hide-vc-output-config"
- **Parameter value:** boolean (see "Report parameter types", section 4)
- **Parameter kind:** optional, default is false (vc-output-config element is present)

Do not enable this parameter if any of the printable formats is to be generated.

Hide Details Option

The following parameter controls absence of vc details in the report:

- **Parameter name:** "hide-vc-details"
- **Parameter value:** boolean (see "Report parameter types", section 4)
- **Parameter kind:** optional, default is false (vc details are present)

Do not enable this parameter if any of the printable formats is to be generated.

Measurement vs. Time Section Parameters

This section must be a subsection to "Channels Selection" (see *"Channel Selection Section Parameters" on page 307*).

Single Measurement Parameters

- **Parameter name:** "row-<measurement-num>"
- **Parameter value:** <value>
- **Parameter kind:** optional, measurement is not present in the report if the parameter is omitted

where

- **<measurement-num>** is the index of measurement: Refer to *Table A-1 on page 301* .
- **<value>** is a string in the form of "1,0,1,0,1" (five digits separated by commas)
  - 1st digit specifies where to include table for this measurement (1 - yes, 0 - no)
  - 2nd digit specifies where to include graph for this measurement (1 - yes, 0 - no)
  - 3rd digit specifies where to show minimum for this measurement (1 - yes, 0 - no)
  - 4th digit specifies where to show maximum for this measurement (1 - yes, 0 - no)
  - 5th digit specifies graph type for the graph (0 - line, 1 - area)

Report Time Range Start

The following parameter controls report time range start:

- **Parameter name:** "start-time"
- **Parameter value:** float (see *"Report Parameter Types" on page 291*) TDateTime Delphi value, absolute time of time range start
- **Parameter kind:** mandatory

Report Time Range End

The following parameter controls report time range end:

- **Parameter name:** "end-time"
- **Parameter value:** float (see *"Report Parameter Types" on page 291*) TDateTime Delphi value, absolute time of time range end
- **Parameter kind:** optional, default is test stop time if test is finished or time of report generation start if test is running

Errors vs. Channels Section Parameters

This section must be a subsection to "Channels Selection" (see *“Channel Selection Section Parameters” on page 307*).

Print Tables Option

Print tables option is controlled via the following parameter:

- **Parameter name:** "print-tables"
- **Parameter value:** boolean (see *“Report Parameter Types” on page 291*)
- **Parameter kind:** optional, default is false

Print Graphs Option

Print graphs option is controlled via the following parameter:

- **Parameter name:** "print-graphs"
- **Parameter value:** boolean (see *“Report Parameter Types” on page 291*)
- **Parameter kind:** optional, default is false

Print Color Legend Option

Print color legend option is controlled via the following parameter:

- **Parameter name:** "print-color-legend"
- **Parameter value:** boolean (see *“Report Parameter Types” on page 291*)
- **Parameter kind:** optional, default is false

Graph Type

Graph type is controlled via the following parameter:

- **Parameter name:** "graph-type"
- **Parameter value**: integer (see *“Report Parameter Types” on page 291*) "0" for totals, "1" for stack bars, "2" for side bars
- **Parameter kind:** mandatory

Errors to Print

The following parameter controls the presence of errors in the report for the section:

- **Parameter name:** "print-error-<error-num>"
- **Parameter value:** boolean (see *“Report Parameter Types” on page 291*)
- **Parameter kind:** optional, default is false

where

<error-num> is the ordinal number of corresponding error. ("1" for EP_NO_DIAL_TONE etc.). Refer to *Table* for valid numbers and error types.

Table A-3. Error Types and Ordinal Numbers

error-num	Error Type
1	EP_NO_DIAL_TONE
2	EP_CONT_DIAL_TONE
3	EP_NO_RINGBACK
4	EP_NO_ANSWER
5	EP_NO_PC_FIRST
6	EP_NO_PC_SUBSEQ
7	EP_UNEXPECT_BUSY
8	EP_UNEXPECT_CONG
9	EP_NO_EXPECT_BUSY
10	EP_NO_TONE
11	EP_NO_ENERGY
12	EP_NO_SILENCE
13	EP_NO_STRING_FIRST
14	EP_NO_STRING_SUBSEQ
15	EP_CONN_FAILED
16	EP_UNEXP_DISCONN
17	EP_ABNORMAL_DISC
18	EP_BER_THRESHOLD_EXCEEDED
19	EP_DIAL_NO_NOT_FOUND
20	EP_FIRST_DIGIT_TIMEOUT
21	EP_INT_DIGIT_TIMEOUT
22	EP_INSUFF_DIGITS_DIALED

Table A-3. Error Types and Ordinal Numbers (continued)

error-num	Error Type
23	EP_NO_DIAL_NO
24	EP_NO_RESPONSE
25	EP_CHAN_BUSY
26	EP_NO_MFR2_BACK_DIG_FIRST
27	EP_NO_MFR2_BACK_DIG_SUBS
28	EP_WRONG_MFR2_BACK_DIG
29	EP_NO_MFR2_SILENCE
30	EP_SCRIPT_HOLDOFF
31	EP_ASSIGNMENT_FAILED
32	EP_EQBUSY
33	EP_NO_METER_TONE
34	EP_V5
35	EP_KMS
36	EP_NO_INCOMING_CALL
37	EP_STOPPED_CHNL
38	EP_DBG_MSG_EV
39	EP_CALLERID_TIMEOUT
40	EP_CALLERID_PHONE_ERR
41	EP_CALLERID_PHONE_ABS
42	EP_CALLERID_NAME_ERR
43	EP_CALLERID_NAME_ABS
44	EP_CALLERID_DATE_ABS
45	EP_CALLERID_RX_err
46	EP_METER_PULSE_TOO_EARLY

Table A-3. Error Types and Ordinal Numbers (continued)

error-num	Error Type
47	EP_METER_PULSE_TOO_LATE
48	EP_METER_PULSE_MISSED
49	EP_METER_PULSE_TOO_SHORT
50	EP_METER_PULSE_TOO_LONG
51	EP_METER_PULSE_NOT_TURNED_OFF
52	EP_PSQM_OVER_THRESHOLD
53	EP_PSQM_TOO_MANY_BADS
54	EP_PSQM_NO_DATA
55	EP_PESQ_UNDER_THRESHOLD
56	EP_PESQ_NO_RESOURCE
57	EP_PESQ_NO_DATA
58	EP_FAXMOD_NO_RESOURCE
59	EP_CODEC_NO_RESOURCE
60	EP_REGR_FAILED
61	EP_NO_PRBS
62	EP_DOMAIN_NOT_FOUND
63	EP_NEGOTIATION_TIMEOUT
64	EP_PAGE_DELIVERY_TIMEOUT
65	EP_NO_PPP_CHNL_CONNECTED
66	EP_SS7_COT_FAIL
67	EP_NO_MFR15_BACK_DIG
68	EP_NO_MFR15_FORWARD_DIG
69	EP_WRONG_MFR15_DIG
70	EP_NO_MF_R15_SILENCE

Table A-3. Error Types and Ordinal Numbers (continued)

error-num	Error Type
71	EP_ERR_FAX_N4
72	EP_ERR_FAX_NS
73	EP_ERR_FAX_N23
74	EP_ERL
75	EP_ERLE
76	EP_ECHO_DELAY
77	EP_ECHO_TELR
78	EP_NO_DISCONNECT
79	EP_PACKETS_MINIMUM_NUMBER
80	EP_G107_UNDER_THRESHOLD
81	EP_PACKETS_LOST_THRESHOLD
82	EP_REG_4XX_RESPONSE_TIME
83	EP_REG_200_RESPONSE_TIME
84	EP_REG_SUCCESS_TIME
85	EP_NO_STUTTER_DIAL_TONE
86	EP_UNEXPECTED_CID
87	EP_UNEXPECTED_CWT
88	EP_NO_FSK_ALERTING_DETECTED
89	EP_NO_FSK_DATA_RECEIVED
90	EP_INVALID_FSK_ALERTING_DETECTED

Error Color

Colors of errors are defined globally for all Errors vs. Channels sections and are saved as parameters of root section. The following parameter controls error color for all Errors vs. Channels sections (should be the parameter of root section):

- **Parameter name:** "error-color-<error-num>"
- **Parameter value:** integer (see *"Report Parameter Types" on page 291*) decimal RGB color value
- **Parameter kind:** optional, black color is default

where

<error-num> is the ordinal number of the corresponding error ("1" for EP_NO_DIAL_TONE etc.). Refer to *Table A-3, "Error Types and Ordinal Numbers," on page 316*.

Errors to Show on Stack/Sidebars

The following parameter controls error's presence on stack- and sidebars:

- **Parameter name:** "graph-error-<error-num>"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is false

where

<error-num>is the ordinal number of the corresponding error ("1" for EP_NO_DIAL_TONE etc.). Refer to *Table A-3, "Error Types and Ordinal Numbers," on page 316*.

This parameter is ignored when "print-error-" for the same error is not specified

Hide Output Config Option

The following parameter controls absence of evc-output-config element in the report:

- **Parameter name:** "hide-evc-output-config"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is false (hide-evc-output-config element is present)

Do not enable this parameter if any of the printable formats is to be generated.

Hide Graph-Related Attributes of Errors-vs-Channels Element Option

The following parameter controls absence of graph-related attributes of errors-vs-channels element in the report:

- **Parameter name:** "hide-evc-graph-attribs"
- **Parameter value:** boolean (see *"Report Parameter Types" on page 291*)
- **Parameter kind:** optional, default is false (graph-related attributes are present)

Do not enable this parameter if any of the printable formats is to be generated.

Sections that Have No Configurable Parameters

The following sections have no configurable parameters:

- Test Status Info
- System Events
- SS7 Events
- SS7 Statistics
- Statistics
- Protocol Settings
- Partition and Timing
- Scripts
- Statistics should be a subsection of the *Channel Selection* section (see *"Channel Selection Section Parameters" on page 307*).

Appendix B

Acronyms

Abacus SU	Abacus Server Unit
Abacus CU	Abacus Client Unit
Abacus UI	Abacus User Interface
API	Application Programming Interface
BRI	Basic Rate Interface
CAS	Channel Associated Signaling
CG	Circuit Generator
ClrCh	Clear Channel signaling
CU	Client Unit
DCME	Digital Circuit Multiplication Equipment
E1	Digital signal, level 1, specified by ITU-T (2.048 Mbps)
E3	Digital signal, level 3
FTP	File Transfer Protocol
FZ	Freeze
IP	Internet Protocol
ISDN	Integrated Services Digital Network
MEGACO	Media Gateway Control
MGCP	Media Gateway Control Protocol
MOS	Mean Opinion Score
PESQ	Perceptual Evaluation of Speech Quality
PESQ-LQ	Perceptual Evaluation of Speech Quality-Listening Quality
PRI	Primary Rate Interface
PSQM	Perceptual Speech Quality Measurement
PSQM+	Enhanced PSQM
PSTN	Public Switched Telephone Network
RGE	Report Generation Engine
RRQ	Registration Request

RTP	Real-Time Transport Protocol
Rx	Receive
SC	System Controller
SIGTRAN	Signaling Transport
SIP	Session Initiation Protocol
SIP-T	Session Initiation Protocol for Telephones
SLC	Subscriber Line Carrier (SLC-96)
SLC	Signaling Link Code (SS7)
SS7	Signaling System 7
SU	Server Unit
T1	T-carrier for Digital Signal Level 1 (1.544 Mbps)
T3	T-carrier for Digital Signal Level 3
Tcl	Tool command language
Tx	Transmit
UI	User Interface
VoIP	Voice over Internet Protocol
XML	eXtensible Markup Language

Appendix C
Action Management Tasks

This appendix lists the six tasks that are part of every call sequence action and includes all possible modes and mode parameters. Tcl script examples showing how to set and change these task modes and parameters are also included.

In this appendix...

BEFORE

NOTHING

Parameters

There are no parameters for this task mode.

Example of Usage

```
::aba::ApiActions act
::aba::DatAction dat
::aba::DatTask task

task.name SetValue $::aba::DatXstasknames::BEFORE
task.mode SetValue $::aba::DatXstaskmodes::NOTHING
dat AddItem_task task
dat.name SetValue "A calls B, Pulse, confirms for Call Length"
act SetAction [dat ExportToXMLString]
```

TIMER

Parameters

There are no parameters for this task mode.

Example of Usage

```
task.name SetValue $::aba::DatXstasknames::BEFORE
task.mode SetValue $::aba::DatXstaskmodes::TIMER
```

TRIGGER

Parameters

There is one required parameter: **VALUE**

Value type is an integer from 0 to 255.

Example of Usage

```
::aba::ApiActions act
::aba::DatAction dat
::aba::DatTask task
::aba::DatParam param1

param1.name SetValue $::aba::DatXsparamnames::VALUE
param1.value SetValue "10"

task.name SetValue $::aba::DatXstasknames::BEFORE
task.mode SetValue $::aba::DatXstaskmodes::TRIGGER
task AddItem_param param1
dat AddItem_task task
dat.name SetValue "A calls B, Pulse, confirms for Call Length"
act SetAction [dat ExportToXMLString]
```

HOOK

ON

Parameters

There are no parameters for this task mode.

Example of Usage

```
task.name SetValue $::aba::DatXstasknames::HOOK
task.mode SetValue $::aba::DatXstaskmodes::ON
```

OFF

Parameters

There are no parameters for this task mode.

Example of Usage

```
task.name SetValue $::aba::DatXstasknames::HOOK
task.mode SetValue $::aba::DatXstaskmodes::OFF
```

FLASH

Parameters

There are no parameters for this task mode.

Example of Usage

```
task.name SetValue $::aba::DatXstasknames::HOOK
task.mode SetValue $::aba::DatXstaskmodes::OFF
```

WAIT

NOWAIT

Parameters

There are no parameters for this task mode.

Example of Usage

```
task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::NOWAIT
```

DIALTONE

Parameters

There is one required parameter: **VALUE**.

Value must be one of the following values:

- **DIALTONE**
- **SECOND**
- **STUTTER**

Example of Usage

```
task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::DIALTONE

param1.name SetValue $::aba::DatXsparamnames::VALUE
```

```
param1.value SetValue $::aba::DatXsparamvalues::SECOND

task AddItem_param param1
```

BUSY

Parameters

There are no parameters for this task mode.

Example of Usage

```
task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::BUSY
```

STRING

Parameters

There are four required parameters:

- **VALUE** -- any string value, max 20 characters
- **INCREMENT --** any string with length less than or equal to length of VALUE string
- **TYPE** --one of the following values:
  - **NORMAL**
  - **ADDCH**
  - **FILE**
- **MODE**--one of the following values:
  - **DTMF**
  - **MFR1**
  - **MFR2**
  - **TONE1**
  - **TONE2**
  - **TONES**
  - **OVERLAP**

Example of Usage

```
param1.name SetValue $::aba::DatXsparamnames::VALUE
param1.value SetValue "TestString"
```

```
param2.name SetValue $::aba::DatXsparamvalues::INCREMENT
param2.value SetValue "123"
param3.name SetValue $::aba::DatXsparamnames::TYPE
param3.value SetValue $::aba::DatXsparamvalues::NORMAL
param4.name SetValue $::aba::DatXsparamnames::MODE
param4.value SetValue $::aba::DatXsparamvalues::OVERLAP

task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::STRING
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
task AddItem_param param4
```

TIME

Parameters

There are two required parameters:

- **VALUE** -- any float value from 0 to 9999
- **RANDOM** -- one of the following values:
  - **FIXED**
  - **UNIFORM**
  - **GAUSSIAN**

  When RANDOM is **UNIFORM** or **GAUSSIAN**, a third parameter is required:
- **VARIANCE** -- any float value from 0 to 9999

Example of Usage

#Example 1:

```
param1.name SetValue $::aba::DatXsparamnames::VALUE
param1.value SetValue "10.5"
param2.name SetValue $::aba::DatXsparamnames::RANDOM
param2.value SetValue $::aba::DatXsparamvalues::GAUSSIAN
param3.name SetValue $::aba::DatXsparamnames::VARIANCE
param3.value SetValue "0.45"
```

```
task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::TIME
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
```

#Example 2:

```
param1.name SetValue $::aba::DatXsparamnames::VALUE
param1.value SetValue "10.5"
param2.name SetValue $::aba::DatXsparamnames::RANDOM
param2.value SetValue $::aba::DatXsparamvalues::FIXED

task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::TIME
task AddItem_param param1
task AddItem_param param2
```

TONE

Parameters

There are three required parameters:

- **TIME** -- any float value from 0 to 9999
- **FREQ1** -- any integer value from 0 to 9999
- **FREQ2** -- any integer value from 0 to 9999

Example of Usage

```
param1.name SetValue $::aba::DatXsparamnames:: TIME
param1.value SetValue "0.8"
param2.name SetValue $::aba::DatXsparamnames::FREQ1
param2.value SetValue "123"
param3.name SetValue $::aba::DatXsparamnames:: FREQ2
param3.value SetValue "1.23"

task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::TONE
```

```
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
```

SILENCE

Parameters

There is one required parameter: **TIME**.

Value type is a Float from 0 to 9999.

Example of Usage

```
param1.name SetValue $::aba::DatXsparamnames:: TIME
param1.value SetValue "0.8"

task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::SILENCE
task AddItem_param param1
```

ENERGY

Parameters

There is one required parameter: **TIME**.

Value type is a Float from 0 to 9999.

Example of Usage

```
param1.name SetValue $::aba::DatXsparamnames:: TIME
param1.value SetValue "0.8"

task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::ENERGY
task AddItem_param param1
```

PRBS

Parameters

There are three required parameters:

- **TIME --** any float value from 0 to 9999.
- **WORD --** one of two predetermined values **- PRBS7** or **PRBS8**.
- **SIGNAL_TYPE --** One of two predetermined values **- PRBS11** or **PRBS15**.

Example of Usage

```
param1.name SetValue $::aba::DatXsparamnames:: TIME
param1.value SetValue "0.8"
param2.name SetValue $::aba::DatXsparamnames::WORD
param2.value SetValue $::aba::DatXsparamvalues:: PRBS8
param3.name SetValue $::aba::DatXsparamnames:: SIGNAL_TYPE
param3.value SetValue $::aba::DatXsparamvalues:: PRBS11

task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::PRBS
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
```

DISCONNECT

Parameters

There are no parameters for this task mode.

Example of Usage

```
task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::DISCONNECT
```

INCOMING_CALL

Parameters

There is one required parameter: **YIELD_NEXT_TASK**.

Value is a Boolean and must be **YES** or **NO**.

Example of Usage

```
param1.name SetValue $::aba::DatXsparamnames::
YIELD_INCOMING_CALL
param1.value SetValue $::aba::DatXsparamvalues::YES

task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::INCOMING_CALL
task AddItem_param param1
```

FAXMODEM

Parameters

There is one required parameter: **PAGES**.

Value type is an integer from 0 to 9999.

Example of Usage

```
param1.name SetValue $::aba::DatXsparamnames::PAGES
param1.value SetValue "10"

task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::FAXMODEM
task AddItem_param param1
```

DATAMODEM

There are two required parameters.

- **TYPE --** reserved for future use. Should be always equal **BYTE_SEQUENCE**.
- **DURATION --** any float value from 0 to 9999 or fixed value **FOREVER**.

Example of Usage

#Example 1:

```
param1.name SetValue $::aba::DatXsparamnames::TYPE
param1.value SetValue $::aba::DatXsparamvalues::BYTE_SEQUENCE
param2.name SetValue $::aba::DatXsparamnames::DURATION
param2.value SetValue $::aba::DatXsparamvalues::FOREVER

task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::DATAMODEM
task AddItem_param param1
task AddItem_param param2
```

#Example 2:

```
param1.name SetValue $::aba::DatXsparamnames::TYPE
param1.value SetValue $::aba::DatXsparamvalues::BYTE_SEQUENCE
param2.name SetValue $::aba::DatXsparamnames::DURATION
param2.value SetValue "10"

task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::DATAMODEM
task AddItem_param param1
task AddItem_param param2
```

VOICE

Parameters

There are two required parameters:.

- **QUALITY --** must be one of the following values:
  - **PSQM**
  - **PSQM_PLUS**
  - **PESQ**
- **FILE1** and/or **FILE2** and/or **FILE3** and/or **FILE4-** contains string value (file name)

Example of Usage

#Example 1:

```
param1.name SetValue $::aba::DatXsparamnames::QUALITY
param1.value SetValue $::aba::DatXsparamvalues::PSQM_PLUS

task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::VOICE
task AddItem_param param1
```

#Example 2:

```
param1.name SetValue $::aba::DatXsparamnames::QUALITY
param1.value SetValue $::aba::DatXsparamvalues::PSQM_PLUS
param2.name SetValue $::aba::DatXsparamnames::FILE3
param2.value SetValue "longs-mu"

task.name SetValue $::aba::DatXstasknames::WAIT
task.mode SetValue $::aba::DatXstaskmodes::VOICE
task AddItem_param param1
task AddItem_param param2
```

SEND

NOSEND

Parameters

There are no parameters for this task mode.

Example of Usage

```
task.name SetValue $::aba::DatXstasknames::SEND
task.mode NOSEND
```

DIAL

Parameters

There is one required parameter: **VALUE**.

Value must be one of the following values:

- **DTMF**
- **MFR1**
- **MFR15**
- **MFR2**
- **PULSE**
- **OVERLAP**

Example of Usage

```
param1.name SetValue $::aba::DatXsparamnames::VALUE
param1.value SetValue $::aba::DatXsparamvalues::DTMF

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::DIAL
task AddItem_param param1
```

STRING

Parameters

There are three required parameters:

- **VALUE -** any string value, max **-** 20 characters.
- **TYPE -** must be one of the following values:
    - **NORMAL**
    - **INCREMENT**
    - **INTERPRET**
    - **ADDCH**
    - **FILE**
    - **RANDOM**

  When TYPE is **RANDOM**, the following parameters are required:
    - **HEX -** Boolean value (**YES** / **NO**)

- **LENGTH -** Any integer from 0 to 9999 for fixed length or **UNFIXED**
- **FROM -** Any integer from 0 to 9999
- **TO -** Any integer from 0 to 9999

- **MODE --** one of the following values:
  - **DTMF**
  - **MFR1**
  - **MFR2**
  - **TONE1**
  - **TONE2**
  - **TONES**
  - **OVERLAP**
  - **PULSE**
  - **SIGNALING**

Example of Usage

#Example 1:

```
param1.name SetValue $::aba::DatXsparamnames::VALUE
param1.value SetValue "test string"
param2.name SetValue $::aba::DatXsparamnames::TYPE
param2.value SetValue $::aba::DatXsparamvalues::NORMAL
param3.name SetValue $::aba::DatXsparamnames::MODE
param3.value SetValue $::aba::DatXsparamvalues::OVERLAP

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::STRING
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
```

#Example 2:

```
param1.name SetValue $::aba::DatXsparamnames::VALUE
param1.value SetValue "test string"
param2.name SetValue $::aba::DatXsparamnames::TYPE
param2.value SetValue $::aba::DatXsparamvalues::RANDOM
```

```
param3.name SetValue $::aba::DatXsparamnames::MODE
param3.value SetValue $::aba::DatXsparamvalues::OVERLAP
param4.name SetValue $::aba::DatXsparamnames::HEX
param4.value SetValue $::aba::DatXsparamvalues::YES
param5.name SetValue $::aba::DatXsparamnames::LENGTH
param5.value SetValue $::aba::DatXsparamvalues::UNFIXED
param6.name SetValue $::aba::DatXsparamnames::FROM
param6.value SetValue "0"
param7.name SetValue $::aba::DatXsparamnames::TO
param7.value SetValue "100"

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::STRING
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
task AddItem_param param4
task AddItem_param param5
task AddItem_param param6
task AddItem_param param7
```

VSC

Parameters

There are three required parameters:

- **VALUE** -- one of the following values:
  - **CALLBACK**
  - **FORWARD**
  - **CIDBLOCK**
  - **CIDDELIVERY**
  - **HOLD**
  - **E911**
- **MODE** -- one of the following values:
  - **DTMF**
  - **MFR1**

- **MFR2**
- **TONE1**
- **TONE2**
- **TONES**
- **OVERLAP**
- **PULSE**
- **SIGNALING**

- **DO --** one of the following values:
  - **ACTIVATE**
  - **DEACTIVATE**

Example of Usage

```
param1.name SetValue $::aba::DatXsparamnames::VALUE
param1.value SetValue $::aba::DatXsparamvalues::CALLBACK
param2.name SetValue $::aba::DatXsparamnames::MODE
param2.value SetValue $::aba::DatXsparamvalues::OVERLAP
param3.name SetValue $::aba::DatXsparamnames::DO
param3.value SetValue $::aba::DatXsparamvalues::ACTIVATE

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::VSC
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
```

TONE

Parameters

There are four required parameters:

- **FREQ1 --** any integer value from 0 to 9999
- **FREQ2 --** any integer value from 0 to 9999
- **TIME --** any float value from 0 to 9999
- **RANDOM --** one of the following values:
  - **FIXED**
  - **UNIFORM**

- **GAUSSIAN**

When RANDOM is **UNIFORM** or **GAUSSIAN**, a third parameter is required:

- **VARIANCE -** any float value from 0 to 9999

Example of Usage

#Example 1:

```
param1.name SetValue $::aba::DatXsparamnames::FREQ1
param1.value SetValue "123"
param2.name SetValue $::aba::DatXsparamnames::FREQ2
param2.value SetValue "1.23"
param3.name SetValue $::aba::DatXsparamnames::TIME
param3.value SetValue "10.5"
param4.name SetValue $::aba::DatXsparamnames::RANDOM
param4.value SetValue $::aba::DatXsparamvalues::GAUSSIAN
param5.name SetValue $::aba::DatXsparamnames::VARIANCE
param5.value SetValue "0.45"

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::TONE
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
task AddItem_param param4
task AddItem_param param5
```

#Example 2:

```
param1.name SetValue $::aba::DatXsparamnames::FREQ1
param1.value SetValue "123"
param2.name SetValue $::aba::DatXsparamnames::FREQ2
param2.value SetValue "1.23"
param3.name SetValue $::aba::DatXsparamnames::TIME
param3.value SetValue "10.5"
param4.name SetValue $::aba::DatXsparamnames::RANDOM
```

```
param4.value SetValue $::aba::DatXsparamvalues::FIXED

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::TONE
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
task AddItem_param param4
```

NOISE

Parameters

There are two required parameters:

- **TIME --** any float value from 0 to 9999.
- **RANDOM --** one of the following values:
  - **FIXED**
  - **UNIFORM**
  - **GAUSSIAN**

  When RANDOM is **UNIFORM** or **GAUSSIAN**, a third parameter is required:
- **VARIANCE -** any float value from 0 to 9999

Example of Usage

```
param1.name SetValue $::aba::DatXsparamnames::TIME
param1.value SetValue "10.5"
param2.name SetValue $::aba::DatXsparamnames::RANDOM
param2.value SetValue $::aba::DatXsparamvalues::GAUSSIAN
param3.name SetValue $::aba::DatXsparamnames::VARIANCE
param3.value SetValue "0.45"

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::NOISE
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
```

#Example 2:

```
param1.name SetValue $::aba::DatXsparamnames::TIME
param1.value SetValue "10.5"
param2.name SetValue $::aba::DatXsparamnames::RANDOM
param2.value SetValue $::aba::DatXsparamvalues::FIXED

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::NOISE
task AddItem_param param1
task AddItem_param param2
```

PRBS

Parameters

There are four required parameters:

- **WORD --** one of two predetermined values - PRBS7 or PRBS8.
- **SIGNAL_TYPE --** One of two predetermined values - PRBS11 or PRBS15.
- **TIME --** any float value from 0 to 9999.
- **RANDOM --** one of the following values:
  - **FIXED**
  - **UNIFORM**
  - **GAUSSIAN**

  When RANDOM is **UNIFORM** or **GAUSSIAN**, a fifth parameter is required:
- **VARIANCE --** any float value from 0 to 9999

Example of Usage

#Example 1:

```
param1.name SetValue $::aba::DatXsparamnames::WORD
param1.value SetValue $::aba::DatXsparamvalues:: PRBS8
param2.name SetValue $::aba::DatXsparamnames::SIGNAL
param2.value SetValue $::aba::DatXsparamvalues:: PRBS11
param3.name SetValue $::aba::DatXsparamnames::TIME
```

```
param3.value SetValue "10.5"
param4.name SetValue $::aba::DatXsparamnames::RANDOM
param4.value SetValue $::aba::DatXsparamvalues::GAUSSIAN
param5.name SetValue $::aba::DatXsparamnames::VARIANCE
param5.value SetValue "0.45"

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::PRBS
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
task AddItem_param param4
task AddItem_param param5
```

#Example 2:

```
param1.name SetValue $::aba::DatXsparamnames::WORD
param1.value SetValue $::aba::DatXsparamvalues:: PRBS8
param2.name SetValue $::aba::DatXsparamnames::SIGNAL
param2.value SetValue $::aba::DatXsparamvalues:: PRBS11
param3.name SetValue $::aba::DatXsparamnames::TIME
param3.value SetValue "10.5"
param4.name SetValue $::aba::DatXsparamnames::RANDOM
param4.value SetValue $::aba::DatXsparamvalues::FIXED

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::PRBS
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
task AddItem_param param4
```

PLAY

Parameters

There are four required parameters:

- **VALUE -- AUDIO** or **VIDEO**
- **PILOT**
- **CPD -- YES** or **NO**
- **FILE1** and/or **FILE2** and/or **FILE3** and/or **FILE4-** contains string value (file name)

Example of Usage

#Example 1:

```
param1.name SetValue $::aba::DatXsparamnames::VALUE
param1.value SetValue $::aba::DatXsparamvalues::AUDIO
param2.name SetValue $::aba::DatXsparamnames::PILOT
param2.value SetValue $::aba::DatXsparamvalues::NO
param3.name SetValue $::aba::DatXsparamnames::CPD
param3.value SetValue $::aba::DatXsparamvalues::YES

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::PLAY
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
```

#Example 2:

```
param1.name SetValue $::aba::DatXsparamnames::VALUE
param1.value SetValue $::aba::DatXsparamvalues::AUDIO
param2.name SetValue $::aba::DatXsparamnames::PILOT
param2.value SetValue $::aba::DatXsparamvalues::NO
param3.name SetValue $::aba::DatXsparamnames::CPD
param3.value SetValue $::aba::DatXsparamvalues::YES
param4.name SetValue $::aba::DatXsparamnames::FILE2
```

```
param4.value SetValue "Longs-mu"

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::PLAY
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
task AddItem_param param4
```

FAXMODEM

Parameters

There are two required parameters:

- **PAGES --** an integer from 0 to 9999.
- **TYPE -- SIMPLE** or **COMPLEX**
  When TYPE is **COMPLEX**, a third parameter is required:
- **FILE --** any string value.

Example of Usage

#Example 1:

```
param1.name SetValue $::aba::DatXsparamnames::PAGES
param1.value SetValue "10"
param2.name SetValue $::aba::DatXsparamnames::TYPE
param2.value SetValue $::aba::DatXsparamvalues::SIMPLE

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::FAXMODEM
task AddItem_param param1
task AddItem_param param2
```

#Example 2:

```
param1.name SetValue $::aba::DatXsparamnames::PAGES
param1.value SetValue "10"
```

```
param2.name SetValue $::aba::DatXsparamnames::TYPE
param2.value SetValue $::aba::DatXsparamvalues::COMPLEX
param3.name SetValue $::aba::DatXsparamnames::FILE
param3.value SetValue "TARGET"

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::FAXMODEM
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
```

DATAMODEM

Parameters

There are two required parameters:

- **TYPE --** reserved for future use. Should always equal **BYTE_SEQUENCE**.
- **DURATION --** any float value from 0 to 9999

Example of Usage

```
param1.name SetValue $::aba::DatXsparamnames::TYPE
param1.value SetValue $::aba::DatXsparamvalues::BYTE_SEQUENCE
param2.name SetValue $::aba::DatXsparamnames::DURATION
param2.value SetValue "37"

task.name SetValue $::aba::DatXstasknames::SEND
task.mode SetValue $::aba::DatXstaskmodes::DATAMODEM
task AddItem_param param1
task AddItem_param param2
```

DO

NOTHING

Parameters

There are no parameters for this task mode.

Example of Usage

```
task.name SetValue $::aba::DatXstasknames::DO
task.mode SetValue $::aba::DatXstaskmodes::NOTHING
```

DO

Parameters

There are three required parameters:

- **PATH_CONFIRM --** one of the following values:
  - **NONE**
  - **ONCE**
  - **FOR**
  - **FOREVER**
  - **CL_FIXED**
  - **CL_UNIFORM**
  - **CL_GAUSSIAN**
  - **CL_POISSON**

  When PATH_CONFIRM is **FOR**, the following parameter is also required:
  - **TIME -** any integer value from 0 to 9999

  When PATH_CONFIRM value is **CL_UNIFORM**, **CL_GAUSSIAN** or **CL_POISSON**, the following parameter is required:
  - **VARIANCE -** any float value from 0 to 9999.
- **CPD --** Boolean value (**YES** / **NO**)
- **PATH_ECHO --** one of the following values:
  - **NONE**
  - **CANCELLER_ENABLED**
  - **CANCELLER_DISABLED**

Example of Usage

#Example 1:

```
param1.name SetValue $::aba::DatXsparamnames::PATH_CONFIRM
param1.value SetValue $::aba::DatXsparamvalues::FOR
param2.name SetValue $::aba::DatXsparamnames::TIME
param2.value SetValue "10"
param3.name SetValue $::aba::DatXsparamnames::CPD
param3.value SetValue $::aba::DatXsparamvalues::YES
param4.name SetValue $::aba::DatXsparamnames::PATH_ECHO
param4.value SetValue $::aba::DatXsparamvalues::NONE

task.name SetValue $::aba::DatXstasknames::DO
task.mode SetValue $::aba::DatXstaskmodes::DO
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
task AddItem_param param4
```

#Example 2:

```
param1.name SetValue $::aba::DatXsparamnames::PATH_CONFIRM
param1.value SetValue $::aba::DatXsparamvalues::NONE
param2.name SetValue $::aba::DatXsparamnames::NO
param2.value SetValue $::aba::DatXsparamvalues::YES
param3.name SetValue $::aba::DatXsparamnames::PATH_ECHO
param3.value SetValue
$::aba::DatXsparamvalues::CANCELLER_DISABLED

task.name SetValue $::aba::DatXstasknames::DO
task.mode SetValue $::aba::DatXstaskmodes::DO
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
```

```
#Example 3:
param1.name SetValue $::aba::DatXsparamnames::PATH_CONFIRM
param1.value SetValue $::aba::DatXsparamvalues::CL_GAUSSIAN
param2.name SetValue $::aba::DatXsparamnames::VARIANCE
param2.value SetValue "10.5"
param3.name SetValue $::aba::DatXsparamnames::CPD
param3.value SetValue $::aba::DatXsparamvalues::YES
param4.name SetValue $::aba::DatXsparamnames::PATH_ECHO
param4.value SetValue
$::aba::DatXsparamvalues::CANCELLER_ENABLED

task.name SetValue $::aba::DatXstasknames::DO
task.mode SetValue $::aba::DatXstaskmodes::DO
task AddItem_param param1
task AddItem_param param2
task AddItem_param param3
task AddItem_param param4
```

AFTER

NOTHING

Parameters

There are no parameters for this task mode.

Example of Usage

```
task.name SetValue $::aba::DatXstasknames::AFTER
task.mode SetValue $::aba::DatXstaskmodes::NOTHING
```

TIMER

Parameters

There are no parameters for this task mode.

Example of Usage:

```
task.name SetValue $::aba::DatXstasknames::AFTER
task.mode SetValue $::aba::DatXstaskmodes::TIMER
```

TRIGGER

Parameters

One required parameter: **VALUE.**

Value type is an integer from 0 to 255.

Example of Usage

```
param1.name SetValue $::aba::DatXsparamnames:: VALUE
param1.value SetValue "10"

task.name SetValue $::aba::DatXstasknames::AFTER
task.mode SetValue $::aba::DatXstaskmodes::TRIGGER
```

SPIRENT
Communications