ValiCert® Validator Toolkit™

Programmer's Guide

Version 3.2

ValiCert, Inc.
1215 Terra Bella Avenue
Mountain View, CA 94043

Part Number: DCU-W-TKPG-0301E

Revision: 0100-1

Contents

3 Toolkit Reference

Preface

This guide describes how an application developer can use the ValiCert Validator Toolkit to integrate digital certificate validation into their own application.

Audience

We assume that the developer is familiar with the following:

- C programming language
- Fundamentals of digital certificates and validation

Organization of This Guide

Section	Description
Introduction	Provides an overview of the Toolkit
Using the Toolkit	Describes how to use the various functions to perform validation. This chapter includes a sample program
Toolkit Reference	Provides reference information about the data structures and functions provides as part of the Toolkit.
Error and Status Codes	List all the error codes and status codes that can be returned.

Typographical Conventions

The following typographical conventions are used in this guide to help you locate and identify information:

Italic text is used for emphasis and book titles.

Bold text identifies menu names, menu options, items you can click on the screen, and keyboard keys.

`Courier font` identifies commands you enter at the command line, file names, folder names, and text that either appears on the screen or that you are required to type in.

NOTE: Notes provide significant, helpful information about a feature, operation, or procedure.

ValiCert Documentation

- ValiCert Enterprise VA™ Installation and Administration Guide
- ValiCert VA Publisher™ Installation and Administration Guide
- ValiCert Validator Suite™ Installation and Configuration Guide
- ValiCert Validator Toolkit™ Programmer's Guide

Technical Support

ValiCert provides integration assistance and general customer support. Please contact us through one of the following methods:

- Email: support@valicert.com
- Telephone: +1.650.567.5469
- Fax: +1.650.254.2148

When you contact us, we would appreciate your sending us as much detailed information as possible regarding your:

- Network
- Platform
- Specific problem and how to reproduce it.

Credits

This product includes portions of SSLeay software written by Eric Young (eay@mincom.oz.au). Copyright (C) 1995-1997 Eric Young. All rights reserved. This product includes software written by Tim Hudson (tjh@mincom.oz.au).

This product includes software from Netscape Communications Corp. Copyright (C) 1997 Netscape Communications Corp. All rights reserved.

CHAPTER
1

Introduction

The ValiCert Validator Toolkit™ allows third-party developers to integrate digital certificate validation into their client applications. It also allows client applications to access the ValiCert Global VA Service℠ which aggregates certificate status information from many public CAs.

The Toolkit provides an API that can be used to integrate validation into the application. It also enables applications to validate all types of digital certificates. The ValiCert Validator Toolkit consists of three major components:

- A set of functions that encapsulate x.509 certificate validation
- A set of functions that enable certificate parsing
- Source code application examples that demonstrate how to use the API

The ValiCert Validator Toolkit saves developers time, and guides them quickly through the implementation process. Consider the following challenges:

- Different Certification Authorities (CAs) use different types of validation mechanisms. There are already three major mechanisms that we have discussed, and many more are in prototype stages. Each has its own syntax, cryptography, messaging and communication mechanisms. The ValiCert Validator Toolkit accommodates all available validation mechanisms.
- Even within a single mechanism such as Certificate Revocation Lists (CRLs), different CA vendors use different encoding mechanisms to understand and retrieve certificate “hot list” data. ValiCert Validator Toolkit-enabled applications can receive aggregated CRL information from the ValiCert Global VA Service, saving developers effort even compared to implementing a simple mechanism.

However, the Toolkit does not address the Implementation of public key cryptography. The Toolkit does not include, for example, an RSA license.

Product Architecture

It is important to understand the relationship of the application to the Toolkit and other components. This will help you to better understand the tasks of your application and the functions that help perform them.

Your application can use the Toolkit to communicate with the VA or can communicate with it directly.

The Toolkit supports several third-party crypto libraries. The Toolkit crypto layer combined with a third-party crypto library is treated as a single lower level component of the Toolkit architecture. Each single-lower component interchangeable to meet the needs of your environment. For information about integrating a crypto library into the Toolkit architecture, see “Integrating Crypto Libraries” on page 10.

Figure 1 shows the relationship of your application to a VA, CA, ValiCert VA Publisher, ValiCert Validator Toolkit, and third-party crypto libraries.

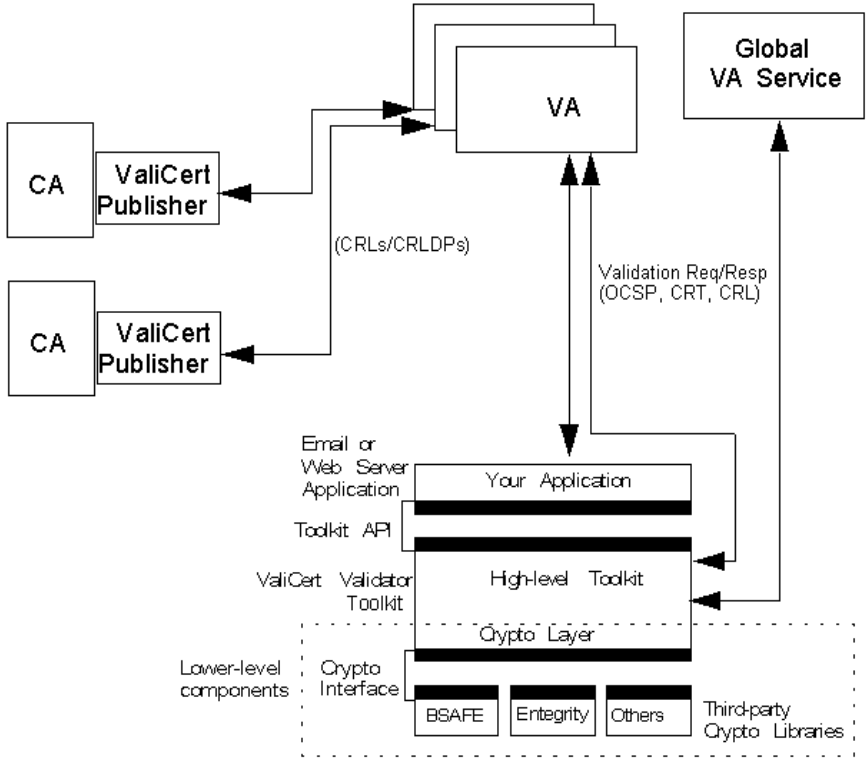


Figure 1. Product Architecture

Supported Validation Mechanisms

The Toolkit enables applications to validate all types of digital certificates and to access the ValiCert Global VA Service. The Toolkit supports the following validation mechanisms:

- Certificate Revocation Lists (CRLs)
- Online Certificate Status Protocol (OCSP)
- Certificate Revocation Trees™ (CRTs)

Certificate Revocation Lists (CRLs)

CRLs are the traditional method of certificate validation. Each CA publishes signed lists of revoked certificates. The verifier downloads these lists, checks the signature on the list, makes sure the list is recent, verifies the date of the list, and searches the list to make sure that the serial number of the certificate in question is not on the list.

CRLs are ill-suited to many applications because downloading the lists becomes impractical as the number of certificates in circulation and on the lists increases. Further, verifiers may have to collect lists from multiple CAs. In short, the network bandwidth, reliability, latency and processing effort of handling CRLs directly are likely to be—or to become—unacceptably large.

For more detailed technical information on CRLs, see RFC 2459 at:

http://www.ietf.org/html.charters/pkix-charter.html

Online Certificate Status Protocol (OCSP)

OCSP defines a mechanism for online certificate status checking, in which a certificate recipient contacts a server (called an OCSP responder) each time it needs to check a certificate's status. OCSP is one of a broader class of approaches that call for a recipient of a message to check some server to ascertain certificate status.

In its pure form, this type of approach has the advantage of providing access to the most up-to-date certificate status information. However, it has the disadvantage of being cumbersome from a communications standpoint, because every secure communication involving n certificates requires n other network connections to ascertain the status of the certificates. Thus, OCSP is expected to be the most appropriate protocol for high-security applications requiring the most up to date certificate status information.

For more detailed information on OCSP, see RFC 2560:

http://www.ietf.org/html.charters/pkix-charter.html

Certificate Revocation Trees™ (CRTs)

CRTs are a high performance technique developed by ValiCert to allow applications to validate certificates efficiently. This technique, based on a cryptographic data structure called a certificate revocation tree, achieves two goals:

1 **Amortize certificate validation costs over many transactions**: A user, having gained short-term proof that its certificate is valid, can use this proof over many secure transactions. More importantly, if the user encloses such proof in a message, recipients can be assured the certificate is valid without making external network requests.

2 **Structure certificate validity assurance data efficiently**: The CRT data structure reduces the data transfers required to create certificate status information and disseminate it by *several orders of magnitude*. This increases the speed of validation, reduces bandwidth consumption, and increases scalability to a virtually unlimited number of users.

CRTs work with existing protocols and X.509 certificates. CRT validation may be used in applications transparently by means of the Toolkit API.

System Requirements

Table 1 lists the system requirements for using the Toolkit.

Table 1. System Requirements

Requirement	Minimum	Recommended
Hardware	For Windows: ❖ Intel Pentium-based or compatible systems For Solaris: ❖ Sun SPARCstation 5	For Windows: ❖ Intel Pentium-II based or compatible systems For Solaris: ❖ SunUltra or compatible
Memory	32 MB	64 MB
Disk Space	20 MB	20 MB

Table 1. System Requirements (Continued)

Requirement	Minimum	Recommended
Operating Systems	❖ Microsoft Windows (x86 based systems) ❖ Windows NT Workstation /Server 4.0 or later ❖ Windows 95/98 **or** ❖ Solaris (SPARC Platform Edition): ❖ Solaris 2.6/2.7	Not applicable
Compilers	❖ Microsoft Visual Studio 97 (Windows Platform) ❖ Sun Visual Workshop 4.2 (Sun Solaris Platform)	Not applicable
Crypto libraries	BSAFE 3.0 or later	Not applicable

Other Considerations

- ❖ The Toolkit APIs in this release are not compatible with previous releases.
- ❖ The Toolkit is a DLL on the Windows platform and a shared object on the Solaris platform.
- ❖ The Toolkit's support for crypto libraries adds libraries to the dependency list for Toolkit users.

CHAPTER
2

Using the Toolkit

This chapter provides information about how you can use the Toolkit API to integrate certificate validation into your application.

Before you begin using the Toolkit functions, familiarize yourself with the following:

- X.509 Certificate Format
- Toolkit Memory Model
- Integrating Crypto Libraries

Once you understand these concepts, you will better understand the following information:

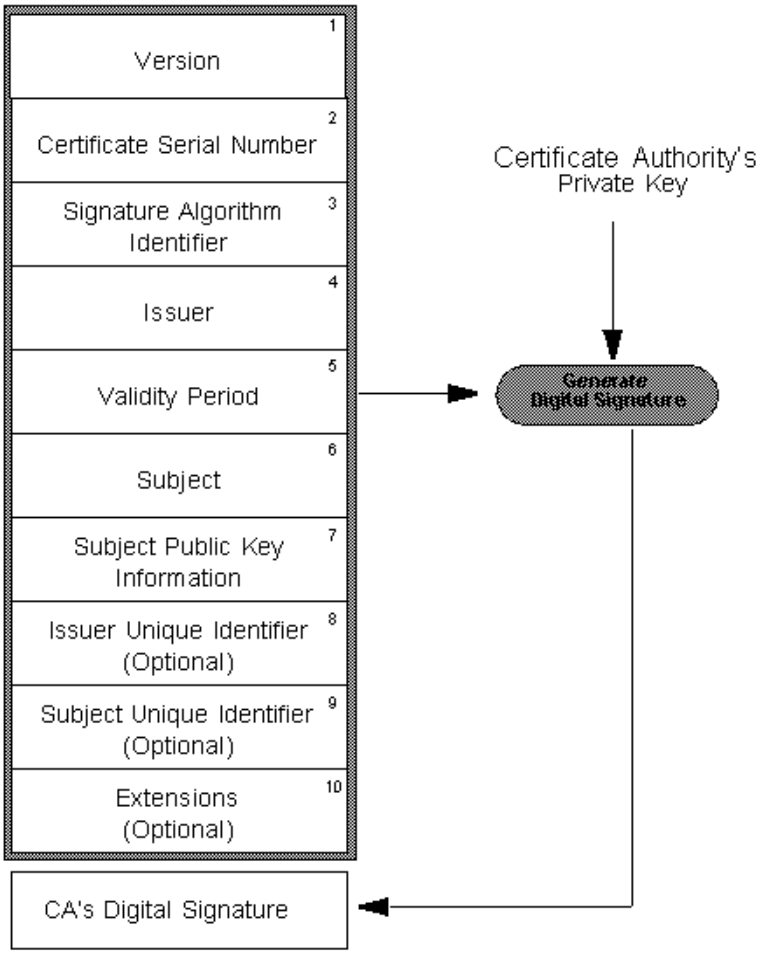
- Implementing Basic Validation in an Application
- Extending Validation in Your Application
- Implementing Specialized Validation Processing

X.509 Certificate Format

The X.509 certificate uses a standardized format to allow interoperability between applications, Certificate Authorities, and Validation Authorities.

Figure 2 shows the format of a X.509 v3 certificate.

NOTE: X.509 v1 certificates do not support extensions.



Legend:

1 version number of encoded certificate
2 unique number assigned by the CA
3 algorithm used by CA to sign the certificate, e.g. RSA
4 CA who has signed and issued the certificate
5 time interval (start date-end date) for which the certificate was issued
6 holder of the private key for which the public key is being certified
7 public key, parameters and the algorithm used
8 optional, allows alternative identifier for issuer (rarely used)
9 optional, allows alternative identifier for subject (rarely used)
10 optional, allows organization to add extension fields specific to their needs

Figure 2. X.509 Certificate Format

Toolkit Memory Model

The Toolkit uses a very simple memory model. For every structure for which you allocate memory using a Toolkit function, you must then use another Toolkit function to release the memory. If you do not observe this simple model, your application will create memory leaks which can result in performance degradations and other problems.

NOTE: The Vtk_Init function and Vtk_Finish functions allocate resources to the Toolkit. Vtk_Init must be called first, before any other Toolkit function and Vtk_Finish must be called last, after all other functions have completed. Both must only be called once.

The Table 2 shows the Toolkit functions you can use to allocate and release memory.

Table 2. Memory Allocation and Release Functions

Allocation Function	Release Function
Vtk_CertGetExtensions	Vtk_ExtensionsDelete
Vtk_CertNew	Vtk_CertDelete
Vtk_CertStoreNew	Vtk_CertStoreDelete
Vtk_CtxtGetOption	Vtk_CtxtGetOptionDeleteContent
Vtk_CtxtNew	Vtk_CtxtDelete
Vtk_ExtensionGetByOid Vtk_ExtensionGetith	Vtk_ExtensionDelete
Vtk_StatusToStrings	Vtk_StatusStringsDelete
Vtk_ValHdlGetRevStatus Vtk_ValidationGetRevStatus Vtk_ValRespSingleCertDetails	Vtk_ValRespSingleCertDetailsDelete
Vtk_ValHdlGetRevStatus Vtk_ValidationGetRevStatus Vtk_ValRespDetails	Vtk_ValRespDetailsDelete
Vtk_ValidationAddCert Vtk_ValidationAddCerRaw Vtk_ValidationGetValHdl	Vtk_ValHdlDelete
Vtk_ValidationGetQueries	Vtk_ValQueriesDelete
Vtk_ValidationNew	Vtk_ValidationDelete

NOTE: In some functions, such as Vtk_ValidationAddCert and Vtk_ValidationAddCertRaw, a pointer to a pointer (**hdl) will allocate the memory without your application specifically calling one of the allocation functions. However, your application will continue to be responsible for releasing the memory allocated; otherwise, memory leaks and other problems will occur.

The Toolkit also provides two other resource allocation functions Vtk_Init and Vtk_Finish. Vtk_Init must be called first, before any other Toolkit function and Vtk_Finish must be called last, after all other functions have completed. Both must be called only once.

Integrating Crypto Libraries

The Toolkit currently supports the following crypto libraries for WIndows NT and UNIX:

- RSA BSAFE Crypto-C
- Entegrity SDP
- Baltimore CST
- Microsoft CAPI (Windows only)
- Generic API for 3rd party crypto vendors

Each crypto library combines with the Toolkit crypto layer to create a lower level component of the Toolkit. The Toolkit distribution media provides the files to support the crypto library. However, you must supply the crypto library and integrate it into the Toolkit architecture for the Toolkit to become operational. Further crypto library integration details are located on the Toolkit distribution media in the `crypto` directory.

NOTE: The Toolkit supports other crypto libraries. Contact support@valicert.com for information about integrating these crypto libraries.

Implementing Basic Validation in an Application

This section describes basic tasks that an application must perform to add validation functionality. These tasks can all be done using the Toolkit API and are described as steps within a procedure. The steps listed in this procedure are also indicated in the sample application to demonstrate how an application can implement these functions.

To integrate validation into your application

Step 1 Initialize the Toolkit

Before your application can use the Toolkit functions, your application must first initialize the Toolkit. Your application must call the Vtk_Init function. Your application should call this function only once. See * STEP 1 in sample application.

Step 2 Create a Context

A context is the global Toolkit environment that your application creates using the Vtk_CtxtNew function.

When your application creates the Toolkit context, your application establishes the default VA URL and the validation protocol employed (using the Vtk_CtxtSetDefaultVa function). See * STEP 2a in sample application.

Your application also establishes the list of trusted certificates for the VAs and CAs (using the Vtk_CtxtAddCerts or Vtk_CtxtAddCert function) and the default VA and protocol (using the Vtk_CtxtSetDefaultVa function). It can set the default protocol to OCSP or CRT. See * STEP 2b in sample application.

NOTE: If your application does not call the Vtk_CtxtSetDefaultVa function, the default VA is set to Global Validation Authority Service (GVAS) using the CRT protocol.

Your application passes this context in every Toolkit function it calls, thereby making the information contained in the context structure persistent across Toolkit calls.

Once the context is created, your application can perform a variety of operations using the Toolkit.

Step 3 Create a validation structure.

Your application must call the Vtk_ValidationNew function to allocate memory for a validation. The structure can encapsulate one or more validation queries. See * STEP 3 in sample application.

Step 4 Obtain user and issuer certificates to place in the validation structure.

Your application can call the Vtk_CertNew to create an empty certificate structure. See * STEP 4a in sample application. Once created, your application can call the Vtk_CertInit or the Vtk_CertLoadFromFile function to populate the certificate structure with data. See * STEP 4b in sample application.

Once the certificate structure is created and populated, your application can add the certificates to validation structure.

NOTE: Your application can skip this step if it uses the Vtk_ValidationAddCertRaw function in Step 5.

Step 5 Add certificates to the validation structure.

Your application must add certificates to the validation structure that it created in Step 3. It can call the Vtk_ValidationAddCert and Vtk_ValidationAddCertRaw functions. See * STEP 5 in sample application.

NOTE: Steps 4 and 5 can be repeated to add other certificates to the validation structure.

Step 6 Validate certificates.

When your application uses the Toolkit functions, most of the work is done locally between your application and the Toolkit using the API. However, when actual validation is to be performed, the application calls the Vtk_ValidationValidate function which requires interaction with the VA, usually over a TCP/IP based network. See * STEP 6 in sample application.

Step 7 Clean up the memory allocations.

Your application must delete all the structures that it has created. In this example, your application must call Vtk_CertDelete,

Vtk_ValidationDelete, and Vtk_CtxtDelete. See * STEP 7 in sample application.

Step 8 Release Toolkit resources

When all other functions complete, your application must call the Vtk_Finish function to release the resources allocated to the Toolkit. It must call this function only once. See * STEP 8 in sample application.

Sample Basic Application

The following is a sample application using the Toolkit API to perform an OCSP or CRT query to the Global VA Service.

```
/*
* Copyright 2000 ValiCert Inc. All Rights Reserved.
*
*
*
* ValiCert Validator Toolkit Sample Program.
*
* This file demonstrates performing common operations with the
* ValiCert Validator Toolkit.
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>
#include <string.h>
#include <memory.h>

/*
 * Toolkit includes.
 */
#include "vtk_defs.h"
#include "vtk_cert.h"
#include "vtk_error.h"
#include "vtk_ctxt.h"
#include "vtk_valid.h"


/*
 *
 * Local function prototypes
 *
 */
```

```
/*
 * showError
 *
 * Displays a textual representation of the Validator Toolkit error
 */
void showError(const char *funcName, Vtk_uint32 code);

/*
 * showStatus
 *
 * Displays textual representation of the passed in validation
 * status code.
 */
static void showStatus(Vtk_uint32 status);

/*
 * simpleValidation
 *
 * Performs a simple OCSP or CRT validation check.
 */
static void
simpleValidation(enum Vtk_ValidationMech mech, int argc,
  char **argv);


/*****************************************************************
*
* Main - Toolkit sample
* ***************************************************************
*/
int main(int argc, char *argv[])
{
      Vtk_uint32 retCode;
      enum Vtk_ValidationMech valMech;

      /*
      * Check command line arguments
      * usage:  vtsample OCSP | CRT userCertFile caCertFile
      */
      if (argc < 4)
{
      fprintf(stdout, "\nUsage:\t%s OCSP | CRT userCertFile
             caCertFile\n", argv[0]);
   exit(-1);
}

   if (strcmp(argv[1], "OCSP") == 0)
      valMech = VTK_VM_OCSP;
```

```
   else if (strcmp(argv[1], "CRT") == 0)
      valMech = VTK_VM_CRT;
   else
   {
      fprintf(stdout, "\nUnsupported validation mechanism.");
      fprintf(stdout,
         "\nUsage:\t%s OCSP | CRT userCertFile caCertFile\n",
              argv[0]);
      exit(-1);
   }

   /*
    * Skip over the application name and validation mechanism
    * parameters.
    */
   argc--;
   argc--;
   argv++;
   argv++;


   /*
    * STEP 1
    * Initialize the Toolkit library.
    */
   retCode = Vtk_Init();assert(retCode == VTK_OK);


   simpleValidation(valMech, argc, argv);


   /*
    * STEP 8
    * Release Toolkit resources.
    */
   Vtk_Finish();

   return 0;
} /* main */


/*
 * simpleValidation
 *
 * Performs a simple OCSP or CRT validation check.
 */
static void
simpleValidation(enum Vtk_ValidationMech mech, int argc, char
**argv)
```

```
{
   Vtk_Ctxt *ctxt;
   Vtk_Cert *userCert = NULL, *issuerCert = NULL;
   Vtk_Validation *q;
   Vtk_uint32 ret, status;


   fprintf(stdout, "\n\nPerforming certificate validation using
                %s...",
      (mech == VTK_VM_OCSP ? "OCSP" : "CRT"));


   /*
    * STEP 2a
    * Start by creating a Toolkit context.
    */
   ctxt = Vtk_CtxtNew();assert(ctxt != NULL);


/*
 * STEP 2b
 * Set appropriate validation mechanism for the context.
 * By default, the context is setup with Global Validation
 * Authority Service (GVAS) using the CRT protocol.
 * If asked to do OCSP, need to change the default protocol.
 */
if (mech == VTK_VM_OCSP)
{
      ret = Vtk_CtxtSetDefaultVa(ctxt, VTK_GVAS_URL, VTK_VM_OCSP);
      if (ret != VTK_OK)
      {
         showError("Vtk_CtxtSetDefaultVa", ret);
         goto done;
      }
}


/*
 * STEP 3
 * Create a Validation structure.
 * Validation structures encapsulate validation operation to a VA
 * using any of the Toolkit supported validation protocols.
 */
q = Vtk_ValidationNew(ctxt);assert(q);
```

```
/*
 * STEP 4a
 * Load passed in user and CA certificates.
 */
userCert = Vtk_CertNew(ctxt);assert(userCert);
issuerCert = Vtk_CertNew(ctxt);assert(issuerCert);

/*
 * STEP 4b
 * Populate the certificates with information from specified
 * file.
 */
  if ((ret = Vtk_CertLoadFromFile(ctxt, userCert, argv[0],
            VTK_DF_BASE64)) != VTK_OK)
  {
     showError("Vtk_CertLoadFromFile", ret);
     goto done;
  }

  if ((ret = Vtk_CertLoadFromFile(ctxt, issuerCert, argv[1],
        VTK_DF_BASE64)) != VTK_OK)
  {
     showError("Vtk_CertLoadFromFile", ret);
     goto done;
  }

/*
 * STEP 5
 * Add the passed in certificate to the validation structure.
 * For each certificate to be validated the Toolkit requires the
 * CA certificate along with the certificate to be validated.
 * This operation can be repeated to add other certificates to be
 * validated. Refer to Vtk_ValidationAddCertRaw for alternative
 * method of suppling the certificate data.
 */
  ret = Vtk_ValidationAddCert(ctxt, q, userCert, issuerCert,
        NULL);
  if (ret != VTK_OK)
  {
     showError("Vtk_ValidationAddCert", ret);
     goto done;
  }
```

```
/*
 * STEP 6
 * Perform the certificate validation.
 * The VA specified in the context will be queried
 * for the certificate(s) status.
 */
   ret = Vtk_ValidationValidate(ctxt, q, &status);


   /*
    * Display validation values.
   */
   if (ret == VTK_OK)
   {
      fprintf(stdout, "\n\nValidation succeeded.");

      /*
       * Display the certificate(s) status.
       */
      if (status & VTK_STATUS_OK)
         fprintf(stdout, "\nCertificate is valid.");
      else
         fprintf(stdout, "\nCertificate is not valid.");


      /*
       * Display detailed information about the performed
       * validation.
       */
      showStatus(status);

   }
   else
   {
      fprintf(stdout, "\n\nValidation failed.");
      showError("Vtk_ValidationValidate", ret);
   }


done:
```

```
/*
 * STEP 7
 * Cleanup memory
 */
   if (q)
      Vtk_ValidationDelete(q);

   if (userCert)
      Vtk_CertDelete(userCert);

   if (issuerCert)
      Vtk_CertDelete(issuerCert);

   if (ctxt)
      Vtk_CtxtDelete(ctxt);
} /* simpleValidation */


/*
 * showError
 *
 * Displays a textual representation of the Validator Toolkit error
 */
static void showError(const char *funcName, Vtk_uint32 code)
{

   fprintf(stdout, "\n\n**ERROR** in %s - 0x%X (%s)\n\n",
      funcName, code, Vtk_ErrorToString(code));
} /* showError */


/*
 * showStatus
 *
 * Displays textual representation of the passed in validation
 * status code.
 */
static void showStatus(Vtk_uint32 status)
{
   char ** statusStrings;

   statusStrings = Vtk_StatusToStrings(status);

/*
 * Ensure Vtk_StatusToStrings worked.
 */
   if (statusStrings)
   {
```

```
      char **temp = statusStrings;

      fprintf(stdout, "\nValidation Status:");

/*
 * Walk through the array of strings returned, printing each
* string, until the end of array is reached - NULL entry.
       */
      while (*temp)
         fprintf(stdout, "\n\t%s", *temp++);

      /*
       * Release status array.
       */
      Vtk_StatusStringsDelete(statusStrings);
   }
   else
      fprintf(stdout,
         "\n\n**ERROR - Vtk_StatusToStrings - no status strings
returned\n\n");

} /* showStatus */
```

Extending Validation in Your Application

This section describes tasks that an application can perform to extend its validation functionality. An application is not required to include these validation functions. However, these tasks are designed to help you develop an application with more robust validation functionality. These tasks can be done using the Toolkit API. Each task is described separately followed by sample code. The sample code demonstrates how to extend the validation functionality in your application. The tasks are as follows:

- Creating Context for Local VA
- Communicating with a VA
- Customizing VA information
- Getting Detailed Revocation Information
- Building and Validating Certificate Chains
- Getting Extension Information
- Adding Logging

Creating Context for Local VA

An application can create a context for the local VA. This is useful because the context uses the Global VA Service as the default VA. To create the context, your application must perform two tasks:

- Set the default VA using the Vtk_CtxtSetDefaultVa function
- Add certificate to the context to establish trust with the VA.

Code Sample for Creating Context for Local VA

```
/*
 * EvaOCSPCtxt
 *
 * Creates a validation context for use with local VA and OCSP.
 *
 * Parameters:
 *  evaUrl - url for local VA server for example,
 *  http://labrador.valicert.com:90
 *
 *  vaCertFile - VA's certificate file (this could be obtained from
 *  file or other means convenient to the application)
 */
static Vtk_Ctxt*
EvaOCSPCtxt(const char *evaUrl, const char *vaCertFile)
{
   Vtk_Ctxt *ret = NULL;
   Vtk_Cert *vaCert = NULL;
   Vtk_uint32 retCode;


   /*
    * Create a default Toolkit context.
    */
   ret = Vtk_CtxtNew(); assert(ret != NULL);


   /*
    *  Customize the context to use OCSP with local EVA.
    */
   if ((retCode = Vtk_CtxtSetDefaultVa(ret, evaUrl, VTK_VM_OCSP))
        != VTK_OK)
   {
      showError("EvaOCSPCtxt - Vtk_CtxtSetDefaultVa", retCode);
      exit(-1);
   }

   /*
```

```
     * Create a Vtk_Cert structure, and read the certificate from
     * a file. The certificate can be obtained by any means
     * convenient to the application.
     */
    vaCert = Vtk_CertNew(ret);assert(vaCert != NULL);

    if ((retCode = Vtk_CertLoadFromFile(ret, vaCert, vaCertFile,
                   VTK_DF_BASE64))
          != VTK_OK)
    {
       showError("Vtk_CertLoadFromFile", retCode);
       Vtk_CertDelete(vaCert);
       Vtk_CtxtDelete(ret);
       return NULL;
    }

    /*
     * Add the certificate to the context to establish trust with the
     * VA
     */
    if ((retCode = Vtk_CtxtAddCert(ret, VTK_VA_CERT, vaCert))
           != VTK_OK)
    {
       Vtk_CtxtDelete(ret);
       Vtk_CertDelete(vaCert);
       showError("EvaOCSPCtxt - Vtk_CtxtAddCert", retCode);
       return NULL;
    }

    /*
     * VA certificate is no longer needed; delete it.
     */
    Vtk_CertDelete(vaCert);


    return ret;
} /* EvaOCSPCtxt */
```

Communicating with a VA

Typically, communication with the VA is handled by the Toolkit using the Vtk_ValidationValidate function. With this function, the Toolkit can send validation queries to one or more VAs to perform validation. The application can then retrieve validation status using Vtk_ValidationGetRevStatus or Vtk_ValHdlGetRevStatus (if the Vtk_ValHdl auxiliary structure is obtained for the certificate).

Under some circumstances, you may want your application to handle the communication with the VA instead of the Toolkit. Some of the reasons your application should handle the communication with the VA are as follows:

- to use SSL for the communication with the VA
- to use asynchronous I/O with the VA
- to use your own source of validation responses or CRLs

The Toolkit imposes no restrictions on how the application handles the communication with the VA. Instead, the Toolkit provides the Vtk_ValidationGetQueries function with the Vtk_ValQuery structure for the application to obtain the validation queries to be sent to the VA and provides the Vtk_ValidationFromQueries to check the information in the responses.

NOTE: The application is responsible for releasing the memory occupied by the response buffers.

Code Sample for Communicating with VA

This code sample demonstrates certificate validation when the application handles the communication with VA.

```
/*
 * QueryValidation
 *
 *
 * Parameters:
 *     pCtxt       - pointer to Toolkit context
 *     pUserCert   - user certificate to be validated
 *     pIssuerCert - issuer certificate of the user certificate
 *
 */

void QueryValidation(Vtk_Ctxt *pCtxt, const Vtk_Cert *pUserCert,
                         const Vtk_Cert *pIssuerCert)
{
   Vtk_Validation *pVal = NULL;
   Vtk_uint32 ret, status;
   Vtk_ValQuery **ppValQueries = NULL;
   int valQueryCount;
   int i;
```

```
/*
 * Create a Validation structure.
 * Validation structures encapsulate the validation query sent to
 * the VA. The application can send the query using any of the
 * supported validation protocols.
 */
pVal = Vtk_ValidationNew(pCtxt);assert(pVal);


/*
 * Add the passed in certificate to the validation structure.
 * For each certificate to be validated, the Toolkit requires the
 * CA certificate as well as the certificate to be validated.
 *
 * An application can repeat this operation to add other
 * certificates to the validation structure.
 */
ret = Vtk_ValidationAddCert(pCtxt, pVal, pUserCert, pIssuerCert,
                            NULL);
if (ret != VTK_OK)
{
   showError("Vtk_ValidationAddCert", ret);
   goto done;
}

/*
 * Vtk_ValidationGetQueries
 *
 * Obtains an array of validation query messages to allow the
 * application instead of the Toolkit to perform the I/O with the
 * VA. The application should set the response field for each
 * query. The returned array should be deleted using
 * Vtk_ValQueriesDelete.
 */
ret = Vtk_ValidationGetQueries(pCtxt, pVal, &valQueryCount,
                               &ppValQueries);
if (ret != VTK_OK)
{
   showError("Vtk_ValidationGetQueries", ret);
   goto done;
}

/*
 * Set Response field for each valQuery
 */
for (i=0; i<valQueryCount; i++)
{
   /*
```

```
         * Get Response from VA - using application's communication
         * function. This function will use data such as host, port,
         * and request obtained using the Vtk_ValidationGetQueries
         * function to get response from the VA.
         *
         * Application is responsible for deleting memory occupied by
         * response member of validation query which is set by the
         * GetVAResponse function.
         * Note: Function GetVAResponse is not part of the Toolkit.
         *       An application must provide such a function that
         *       would handle the communication with the VA.
         */
        ret = GetVARepsonse(ppValQueries[i]->host,
                 ppValQueries[i]->port,ppValQueries[i]->request,
                 &(ppValQueries[i]->response));
        if (ret != VTK_OK)
        {
           showError("Vtk_ValidationGetQueries", ret);
           goto done;
        }
     }


     /*
      * Vtk_ValidationValidateFromQueries
      *
      * Validates response obtained from VA
      */
     ret = Vtk_ValidationValidateFromQueries(pCtxt, pVal,
                                        ppValQueries, &status);
     if (ret != VTK_OK)
     {
        showError("Vtk_ValidationValidateFromQueries", ret);
        goto done;
     }


     /*
      *                IMPORTANT
      *
      * At this point, the application would continue processing
      * validation results such as display, store, or would call
      * another function
      *
      */

done:
```

```
        /*
         * Cleanup memory
         */

        /*
         * Application is responsible for deleting memory occupied by
         * validation queries and the response part of each
         * query obtained by the GetVAResponse function, which handles
         * the communication with the VA.
         */
        if (ppValQueries)
        {
           for (i=0; i<valQueryCount; i++)
           {
              if (ppValQueries[i]->response.dPtr)
              {
              free(ppValQueries[i]->response.dPtr);
              ppValQueries[i]->response.dPtr = NULL;
              }
           }

           Vtk_ValQueriesDelete(ppValQueries);
        }

        if (pVal)
           Vtk_ValidationDelete(pVal);

    } /* QueryValidation */
```

Customizing VA information

You can customize validation information about the VA that is validating certificates issued by specific CAs. Customizing this information allows an application to contact different VAs for the CAs.

You can use the Vtk_CtxtSetVaInfo function to set the validation information. The VA validation information includes the VA URL, validation mechanism and optionally, the list of trusted VA certificates to be used with the VA.

NOTE: If you do not set the list of trusted VA certificates using the Vtk_CtxtSetVaInfo function, you must set the list through the Vtk_CtxtAddCert or Vtk_CtxtAddCerts functions.

The VA URL and validation mechanism specified in the Vtk_CtxtSetVaInfo function will be used as the default for all validations performed for certificates issued by this CA. If you specify CRL as the validation mechanism, you must also set details about the protocol in the Vtk_ProtocolDetails structure.

Code Sample for Customizing VA Information

The following code sample demonstrates how to customize VA information for a specific CA. It includes information for the OCSP, CRT and CRL validation mechanisms.

```
/*
 * SetVAForSpecificCA
 *
 * Parameters:
 * pCtxt    - pointer to Toolkit context
 * pCaCert1 - CA certificates for which specific VA will be set
 * pCaCert2
 * pCaCert3
 * vaUrl1   - URL of the OCSP VA, for example:
              http://ocsp.valicert.net:80/
 * vaUrl2   - URL of the CRT VA, for example:
              http://ci.valicert.net:80/
 * vaUrl3   - URL of the CRL VA, for example:
              http://testlab/crl_test/testlab.crl
 * pVaCerts - certificate store with trusted VA certificates
 *
 */

int SetVAForSpecificCA(Vtk_Ctxt *pCtxt, const Vtk_Cert *pCaCert1,
              const Vtk_Cert *pCaCert2, const Vtk_Cert *pCaCert3,
              const char *vaUrl1, const char *vaUrl2,
              const char *vaUrl3, const Vtk_CertStore *pVaCerts)
{
   Vtk_uint32 ret;
   enum Vtk_ValidationMech mech;
   Vtk_ProtocolDetails protocol;


/*
 *
 * This call sets the validation information specific to an
 * individual CA. The VA is using OCSP protocol - Protocol Details
 * need not to be provided. The VA certificate must in the with
 * trusted VA certificates store, pVaCerts
    *
    *
    */
```

```
   mech = VTK_VM_OCSP;
   ret = Vtk_CtxtSetVaInfo(pCtxt, pCaCert1, vaUrl1, mech, NULL,
                           pVaCerts);
   {
      showError("Vtk_CtxtSetVaInfo", ret);
      return -1;
   }

 /*
  *
  * This call sets the individual VA information (in this case -
  * CRT validation mechanism for CA represented by certificate
  * pCaCert2. VA certificate is in this case provided in following
  * function Vtk_CtxtAddCerts.
  *
  */
   mech = VTK_VM_CRT;
   ret = Vtk_CtxtSetVaInfo(pCtxt, pCaCert2, vaUrl2, mech, NULL,
                            NULL);
   {
      showError("Vtk_CtxtSetVaInfo", ret);
      return -1;
   }

   ret = Vtk_CtxtAddCerts(pCtxt, VTK_VA_CERT, pVaCerts);
   {
      showError("Vtk_CtxtAddCerts", ret);
      return -1;
   }


 /*
  *
  * This call sets the individual VA information for CA represented
  * by certificate pCaCert2. Since VA is using CRL validation
  * mechanism, protocol details must be provided.
  *
  *
  * VA certificate is already set in the context through
  * Vtk_CtxtAddCerts function.so it need not be set in this call.
  *
  */
   protocol.type = VTK_VM_CRL;
   protocol.d.crl.encoding = VTK_DF_DER;
   protocol.d.crl.type = VTK_DT_CRL;

   ret = Vtk_CtxtSetVaInfo(pCtxt, pCaCert3, vaUrl3, VTK_VM_CRL,
                           &protocol, NULL);
```

```
   {
      showError("Vtk_CtxtSetDefaultVa", ret);
      return -1;
   }


   return 0;
}
```

Getting Detailed Revocation Information

The Toolkit allows an application to retrieve detailed revocation information for a single certificate using either the Vtk_ValidationGetRevStatus or Vtk_ValHdlGetRevStatus validation function. The application can request detailed revocation information for the entire validation response or a single certificate. The detailed revocation information is returned in the Vtk_ValRespDetails or Vtk_ValRespSingleCertDetails structure.

To use the Vtk_ValHdlGetRevStatus function, an application can add a certificate to the validation query with a validation handle using Vtk_ValidationAddCert or Vtk_ValidationAddCertRaw functions. Alternatively, the application can use the Vtk_ValidationGetValHdl function.

If the application requests detailed revocation information (either respDetails or certDetails) but this information is not available in the response, the function returns VTK_OK, but the return values in the structure are not set.

Code Sample for Obtaining Revocation Information

The following code sample shows how to obtain revocation information after certificate validation.

```
/*
 *
 * GetRevocationInfo
 * Parameters:
 *     pCtxt       - pointer to Toolkit context
 *     pUserCert   - certificate which will be validated
 *     pIssuerCert - issuer certificate of the user certificate
 */

void GetRevocationInfo(Vtk_Ctxt *pCtxt, const Vtk_Cert *pUserCert,
const Vtk_Cert *pIssuerCert)
{
   Vtk_Validation *pVal = NULL;
   Vtk_uint32 ret, status;
```

```
Vtk_ValHdl *pHdl = NULL;

/*
 * Create a Validation structure.
 * Validation structures encapsulate the validation query sent to
 * the VA. The application can send the query using any of the
 * supported validation protocols.
 */
pVal = Vtk_ValidationNew(pCtxt);assert(pVal);


/*
 * Add the passed in certificate to the validation structure.
 * For each certificate to be validated, the Toolkit requires the
 * CA certificate as well as the certificate to be validated.
 *
 * The function will also set the validation handle (Vtk_ValHdl)
 * which can be used to obtain validation details for user
 * certificates.
 *
 */
ret = Vtk_ValidationAddCert(pCtxt, pVal, pUserCert, pIssuerCert,
                                &pHdl);
if (ret != VTK_OK)
{
   showError("Vtk_ValidationAddCert", ret);
   goto done;
}

/*
 * Vtk_ValidationValidate
 *
 * Performs certificate validation.
 */
ret = Vtk_ValidationValidate(pCtxt, pVal, &status);
if (ret != VTK_OK)
{
   showError("Vtk_ValidationValidate", ret);
   goto done;
}



/*
 * Displays validation response details using the validation
 * handle, that is, the Vtk_ValHdl structure.
 */
displayValidationDetailsHdl(pCtxt, pHdl, pVal);
```

```
   /*
    *            IMPORTANT
    *
    * Alternatively, if the validation handle is not available, an
    * application can get validation details using the user and
    * issuer certificate pair.
    */
   displayValidationDetails(pCtxt, pVal, pUserCert, pIssuerCert);


done:

   /* Cleanup memory */

   if (pVal)
      Vtk_ValidationDelete(pVal);

   if (pHdl)
      Vtk_ValHdlDelete(pHdl);
}


/*
 * displayValidationDetailsHdl
 *
 * Displays validation response details using the validation
 * handle, that is, the Vtk_ValHdl structure.
 *
 * The Vtk_ValHdl is used to obtain detailed validation information
 * for a particular certificate in the validation query.
 * The Vtk_ValRespDetails structure details revocation information
 * for the entire validation response. It includes header
 * information common to all responses and header information
 * specific to the OCSP, CRT, or CRL protocol.
 *
 * The Vtk_ValRespSingleCertDetails structure represents validation
 * information for a specific certificate.
 *
 * Parameters:
 *     pCtxt       - pointer to Toolkit context
 *     pHdl        - validation handle which identifies the
 *                   certificate
 *     pVal        - pointer to validation query structure that
 *                   contains the validated user certificate
 *
 */
```

```
void displayValidationDetailsHdl(const Vtk_Ctxt *pCtxt,
             const Vtk_ValHdl *pHdl, const Vtk_Validation *pVal)
{
   Vtk_ValRespDetails *hdr = NULL;
   Vtk_ValRespSingleCertDetails *certDetails = NULL;
   Vtk_uint32 ret, status;

   assert(pHdl);

   /*
    * Obtain validation details using a validation handle. The
    * validation handle can be created using the
    * Vtk_ValidationAddCert, Vtk_ValidationAddCertRaw, or the
    * Vtk_ValidationGetValHdlfunction.
    *
    */
   if ((ret = Vtk_ValHdlGetRevStatus(pCtxt, pHdl, &status, &hdr,
                            &certDetails)) != VTK_OK)
   {
      showError("Vtk_ValHdlGetRevStatus", ret);
      return;
   }

   if (hdr)
   {
      /* Print Validation Header information. */
      printRespHdr(hdr);
      Vtk_ValRespDetailsDelete(hdr);
   }

   if (certDetails)
   {
      /* Print single certificate validation information. */
      printRespCertDetails(certDetails);
      Vtk_ValRespSingleCertDetailsDelete(certDetails);
   }

   return;
}



/*
 * displayValidationDetails
 *
 * Display validation response details using the user and issuer
 * certificate pair.
 *
```

```
 * Parameters:
 *     pCtxt       - pointer to Toolkit context
 *     pVal        - pointer to validation query structure that
 *                   contains validated user certificate
 *     pUserCert   - user certificate
 *     pIssuerCert - issuer certificate
 */
void displayValidationDetails(const Vtk_Ctxt *pCtxt,
          const Vtk_Validation *pVal, const Vtk_Cert *pUserCert,
          const Vtk_Cert *pIssuerCert)
{
   Vtk_ValRespDetails *hdr = NULL;
   Vtk_ValRespSingleCertDetails *certDetails = NULL;
   Vtk_uint32 ret, status;

   assert(pUserCert);
   assert(pIssuerCert);

   /*
    * Obtain validation details using the certificate pair.
    */
   if ((ret = Vtk_ValidationGetRevStatus(pCtxt, pVal, pUserCert,
            pIssuerCert, &status, &hdr, &certDetails)) != VTK_OK)
   {
      showError("Vtk_ValidationGetRevStatus", ret);
      return;
   }


   if (hdr)
   {
      /* Print Validation Header information. */
      printRespHdr(hdr);
      Vtk_ValRespDetailsDelete(hdr);
   }

   if (certDetails)
   {
      /* Print single certificate validation information. */
      printRespCertDetails(certDetails);
      Vtk_ValRespSingleCertDetailsDelete(certDetails);
   }
   return;
}
```

```
/*
 * Display contents of a Vtk_ValRespDetails structure
 */
void printRespHdr(const Vtk_ValRespDetails *hdr)
{
   const char* protocols[] = { "CRT", "OCSP", "CRL" };
   char *issuer;


   /*
    * Print Validation Header information.
    */
   printf("\nValidation response Header...");
   printf("\n\tProtocol %s,", protocols[hdr->type - 1]);
   printf("\n\tVersion: %d,\t Issue time: %s", hdr->version,
           ctime(&hdr->issueTime));

   /*
    * Either the issuerIdByName or issueridByKey will be set.
    */
   if (hdr->issuerIdByName)
      issuer = (char*)hdr->issuerIdByName->dPtr;
   else
      issuer = (char*)hdr->issuerIdByKey->dPtr;

   printf("\tIssuer: %s", issuer);

   /*
    * Display additional, protocol specific details.
    */
   if (hdr->type == VTK_VM_CRT)
   {
      printf("\n\tCRT next update: %s",
               ctime(&hdr->d.crt->nextUpdate));
      printf("\tCRT valid until: %s",
               ctime(&hdr->d.crt->validUntil));
   }
}

/*
 * Print contents of a Vtk_ValRespSingleCertDetails structure.
 */
void printRespCertDetails(const Vtk_ValRespSingleCertDetails
                          *certDetails)
{
   printf("\nSingle certificate details...");

   if (certDetails)
```

```
    {
       printf("\n\tThis update: %s",
          (certDetails->thisUpdate == -1 ? "not specified" :
             ctime(&certDetails->thisUpdate)));
       printf("\tNext update: %s",
          (certDetails->nextUpdate == -1 ? "not specified" :
             ctime(&certDetails->nextUpdate)));

       /*
        * If this certificate is revoked, there may be additional
        * information.
        */
       if (certDetails->certStatus & VTK_STATUS_REVOKED)
       {
          /*
           * revocation time is optional parameter -1 means not set
           */
          if (certDetails->revocationTime != -1)
          printf("\n\tRevocation time: %s",
                   ctime(&certDetails->revocationTime));

          /* revocation reason is optional field -1 means not set */
          if (certDetails->revocationReason
                != VTK_REV_STATUS_UNKNOWN)
          printf("\tRevocation reason: %d",
                  certDetails->revocationReason);
          else
          printf("\tRevocation reason not specified.");
       }
    }

    printf("\n\n");
}
```

Building and Validating Certificate Chains

A certificate chain is a hierarchy of certificates that lead to a trusted certificate, usually a CA's certificate. The purpose of the certificate chain is to allow a sender to establish trust with the recipient. Within the Toolkit, an application can build a chain using the Vtk_ValidationAddCertChain function. This validation function builds a certificate chain for the specified certificate and adds all the certificates to the validation query.

The CA certificates stored in the Vtk_Ctxt are used while constructing the chain using the Vtk_ValidationAddCertChain function. When an application calls this function, the application can use Vtk_ChainBuildCallback function to provide a function pointer that will be called every time the Toolkit discovers a new link in the certificate chain.

Certificates in a certificate chain are either from the intermediate certificate store or the trusted certificate store. Certificates in the intermediate certificate store are not the highest certificate in the certificate hierarchy. However, an application can trust the intermediate certificate and discontinue building a certificate chain or continue building the certificate chain until a certificate that the application is satisfied to trust, usually one from the trusted certificate store, has been encountered.

Code Sample for Building and Validating Certificate Chain

This sample demonstrates how to validate the entire certificate chain for the specified certificate.

```
/*
 * ChainValidation
 *
 * Parameters:
 *     pCtxt         - pointer to Toolkit context
 *     pUserCert     - user certificate to be validated
 *     pCAsFile      - name of the file with trusted CA
 *                     certificates in Base64 format.
 *
 */

int ChainValidation(Vtk_Ctxt *pCtxt, const Vtk_Cert *pUserCert,
                          const char *pCAsFile)
{
   Vtk_CertStore *pCaStore = NULL;
   Vtk_Validation *pVal = NULL;
   Vtk_uint32 ret, status;

   /*
    * Create a validation structure.
    * Validation structures encapsulate validation operation to a VA
    * using any of the toolkit supported validation protocols.
    */
   pVal = Vtk_ValidationNew(pCtxt);assert(pVal);
```

```
/*
 * Initialize structures for certificate store
 * with trusted CA certificates.
 */
pCaStore = Vtk_CertStoreNew(pCtxt);assert(pCaStore);

/*
 * Load trusted CA certificates from specified files.
 */
if ((ret = Vtk_CertStoreLoadFromFile(pCtxt, pCaStore, pCAsFile,
              VTK_DF_BASE64)) != VTK_OK)
{
   Vtk_CertStoreDelete(pCaStore);
   Vtk_ValidationDelete(pVal);
   showError("Vtk_CertStoreLoadFromFile", ret);
   return -1;
}

/*
 * This call adds certificates to the list of trusted
 * certificates. These can be VA or CA certificates, or both.
 */
ret = Vtk_CtxtAddCerts(pCtxt, VTK_TRUSTED_CA_CERT, pCaStore);
Vtk_CertStoreDelete(pCaStore);pCaStore = NULL;
if (ret != VTK_OK)
{
   showError("Vtk_CtxtAddCerts", ret);
   Vtk_ValidationDelete(pVal);
   return -1;
}


/*
 * Vtk_ValidationAddCertChain
 *
 * This call builds a certificate chain for the specified
 * certificate and adds all the certificates to the validation
 * structure. The CA certificates stored in the Vtk_Ctxt are used
 * while constructing the chain.
 */
ret = Vtk_ValidationAddCertChain(pCtxt, pVal, pUserCert, NULL,
                        NULL);
if (ret != VTK_OK)
{
   showError("Vtk_ValidationAddCert", ret);
   Vtk_ValidationDelete(pVal);
   return -1;
}
```

```
   /*
    * Vtk_ValidationValidate
    *
    * Performs the certificate validation
    */
   ret = Vtk_ValidationValidate(pCtxt, pVal, &status);
   if (ret != VTK_OK)
   {
      showError("Vtk_ValidationValidate", ret);
      Vtk_ValidationDelete(pVal);
      return -1;
   }

   /*
    *               IMPORTANT
    *
    * At this point the application would continue processing
    * validation results such as display, store, or call another
    * function.
    */


   Vtk_ValidationDelete(pVal);
   return 0;
} /* ChainValidation */
```

Getting Extension Information

Your application can get information about certificate extensions for a specific certificate. Your application can do this by first getting a list of extensions using the Vtk_CertGetExtensions function which returns a Vtk_Extensions structure for the certificate. This structure contains the list of extensions that can be parsed using several of the extension functions. It represents X.509 extensions used in certificates, CRLs, and the OCSP and CRT protocols.

An application can obtain this structure using the Vtk_CertGetExtensions function or from the Vtk_ValRespDetails or Vtk_ValRespSingleCertDetails structure returned by the Vtk_CRLValidateCert, Vtk_ValidationGetRevStatus and Vtk_ValHdlGetRevStatus functions.

The Toolkit provides several other functions that allow your application to parse the Vtk_Extensions structure and return the following:

- number of extensions in the list
- extension in the list based on its OID
- extension based on its position in the extension list

These search functions can be used to search a list of any type of extensions, that is certificate, OCSP, CRT, or CRL extensions.

The Vtk_ExtensionGetByOID can be used to search for a specific Object Identifier (OID) in the list of extensions currently in Vtk_Extensions structure. The OID can be specified in dot notation. The application must call Vtk_ExtensionDelete when finished with the returned structure, otherwise memory leaks and other problems can occur.

The Vtk_ExtensionsGetCount function can be used to determine the number of extensions currently in the Vtk_Extensions structure for the specified context. This function can be used to search a list of any type of extensions, that is certificate, OCSP, CRT, or CRL extensions.

The Vtk_ExtensionsGetith function can be used to search for a specific occurrence of an extension within the list of extensions currently in Vtk_Extensions structure. The application must call Vtk_ExtensionDelete when finished with the returned structure, otherwise memory leaks and other problems can occur.

NOTE: The Vtk_ExtensionsGetith function adds a comment, enclosed by parentheses, to the OID. For example:

```
2.5.29.15(X509v3 Key Usage)
```

If you use the Vtk_ExtensionGetByOID function to search the extensions list, be sure that the OID does not contain a comment.

Code Sample for Getting Extension Information

This code sample demonstrates how to process extensions, get the number of extensions in the extension container, and access individual extensions in the extension container.

```
/*
 * Extensions
 *
 *
 * Parameters:
 * pCtxt        - pointer to Toolkit context
 * pExtensions  - container for extensions; this can certificate,
 *                OCSP,CRT or CRL extensions; You can obtain
 *                certificate extensions by calling
 *                Vtk_CertGetExtensions. The OCSP/CRT/CRL
 *                extensions are part of the Vtk_ValRespDetails
 *                and Vtk_ValRespSingleCertDetails data
 *                structures.
 */

void Extensions(const Vtk_Ctxt *pCtxt, const Vtk_Extensions
*pExtensions)
{
   Vtk_uint32 ret;
   Vtk_Buffer oidBuf;
   char oid[12] = "2.5.29.31";
   Vtk_Extension *pFoundExt = NULL;
   int i;
   int nNumExt;
   Vtk_Extension *pExt;

/*
 * Print OID of all certificate extensions
 */

/*
 * Get Number of extensions
 */
   nNumExt = Vtk_ExtensionsGetCount(pCtxt, pExtensions);

/*
 * Print OID part of each extension
 */
   for (i=0; i<nNumExt; i++)
   {
```

```
   /*
    * Get next extension from extension container
    *
   * Application is responsible for deleting extension obtained by
   * function Vtk_ExtensionsGetith
   */
      pExt = NULL;
      ret = Vtk_ExtensionsGetith(pCtxt, pExtensions, i, &pExt);
      if (ret != VTK_OK)
      {
         showError("Vtk_ExtensionsGetith", ret);
         return;
      }
   /*
    * Print out the OID
    */
      if (pExt)
      {
         printf("/nOid: %s", pExt->oid.dPtr);
      /*
       * Release current extension
       */
         Vtk_ExtensionDelete(pExt);
      }
   }
/*
 *  Search for specific extension - 2.5.29.31 (CRL Distribution
 *  Points)
 */
   oidBuf.type = VTK_DF_STRING;
   oidBuf.dPtr = (unsigned char *) oid;
   oidBuf.len = strlen(oid);

   ret = Vtk_ExtensionGetByOid(pCtxt, pExtensions, &oidBuf,
&pFoundExt);
   if (ret != VTK_OK)
      myPrintf("\nExtension not found .\n");

   /*
    *                IMPORTANT
    *
    * Application continues using the extension it has found.
    */
/*
 * Application is responsible for deleting found extension
 */
   if (pFoundExt)
      Vtk_ExtensionDelete(pFoundExt);
```

Adding Logging

Your application can include logging of important Toolkit activity. The ability to provide activity logs in an external file is helpful for debugging, troubleshooting, performance-tuning and as a history-maintenance tool for Toolkit users. Logging functions record specific Toolkit actions and any errors that are encountered.

By default, logging information is stored in an ASCII text file that can be viewed with any text editor.

You can also customize logging to take advantage of any existing logging capabilities in your application, for example you may want to write log messages to a database or to the system log facilities.

Toolkit stores several types of log records which are also use indicate logging levels. Toolkit functions store these types of log records:

- Error logs—display any encountered error or warning.
- Information logs—provide description and status of key Toolkit activities.
- Debug logs—provides information which usually accompanies an error message for example the value of some variables when an error occurred or information about each successfully called function.
- Validation logs—provide data that is either request data sent to a VA or response data received from a VA.

Specifying a level of logging turns on the levels below it, for example if you specify Debug logs, Debug, Information, and Error logs will be recorded.

Code Samples For Adding Logging

The following code sample demonstrates how to use the default Toolkit logging functionality in your application

```
/*
 * InitLog
 *
 * Initializing and starting logging activity in application
 */
Vtk_uint32 InitLog(Vtk_Ctxt *pCtxt)
{
   Vtk_LogOptions logOpts;
   Vtk_uint32 ret;

   /*
    * Default Toolkit Logging
    * Initialize logging options structure:
```

```
    * Log Level - log all error, info, debug and validation messages
    * Log Mode  - overwrite old log file, no flushing after each
message
    * Log File Name  - use default log file name (vc_toolkit.log)
    */
   logOpts.logLevel = LOG_VaData;
   logOpts.logMode = VTK_LOG_MODE_DEFAULT | VTK_LOG_MODE_OVERWRITE;
   logOpts.logFileName.type = VTK_DF_STRING;
   logOpts.logFileName.dPtr = NULL;
   logOpts.logFileName.len =  0;
   logOpts.openLogCB   = NULL;
   logOpts.closeLogCB  = NULL;
   logOpts.writeLogCB  = NULL;

   /*
    * Open Log
    */
   if ((ret = Vtk_OpenLog(pCtxt, &logOpts)) != VTK_OK)
      showError("InitLog", ret);

   /*
    *             IMPORTANT
    *
    * From now on until the function Vtk_CloseLog is called, all
    * Toolkit messages (error, info, debug, validation) will be
logged
    * into text file vc_toolkit.log
    *
    * Application should call Vtk_CloseLog function to stop logging
    *
    */

   return ret;
}
```

The following code sample demonstrates how an application can provide an alternate way to process logging messages. In this example error, info and debug messages are written in the file `messages.log`, while validation data (VA requests and responses) are written in the file `va_data.log`.

```
/*
 * Logging structure
 */
typedef struct
{
```

```
   FILE *pMsgFile;      /* file stream for error, info and debug
messages */
   FILE *pVaDataFile;   /* file stream for validation data */
} MyLogInfo;


/*
 * InitLog
 *
 * Initializing and starting logging activity in application
 */
Vtk_uint32 InitLog(Vtk_Ctxt *pCtxt)
{
   Vtk_LogOptions logOpts;
   Vtk_uint32 ret;

   /*
    * Alternative Toolkit Logging
    * Initialize logging options structure:
    * Log Level - log all error, info, debug and validation messages
    * Set callback functions
    */
   logOpts.logLevel = LOG_VaData;
   logOpts.logMode = VTK_LOG_MODE_DEFAULT;
   logOpts.logFileName.type = VTK_DF_STRING;
   logOpts.logFileName.dPtr = NULL;
   logOpts.logFileName.len =  0;
   logOpts.openLogCB   = MyOpenLog;
   logOpts.closeLogCB  = MyCloseLog;
   logOpts.writeLogCB  = MyWriteLog;

   /* Open Log */
   if ((ret = Vtk_OpenLog(pCtxt, &logOpts)) != VTK_OK)
      showError("InitLog", ret);

   /*
    *             IMPORTANT
    *
    * From now on until the function Vtk_CloseLog is called, all
    * Toolkit messages (error, info, debug, validation) will be
logged
    * in log files ( as defined in MyWriteLog callback function )
    *
    * Application must call Vtk_CloseLog function to stop logging
and
    * release alocated resources
    *
    */
```

```
   return ret;
}


/*
 * MyOpenLog
 *
 * Callback function for initializing the alternative logging
 */
Vtk_uint32 VTK_CALLBACK MyOpenLog(const Vtk_Ctxt *pCtxt, void
**userHdl)
{
   MyLogInfo *pLogInfo;
   *userHdl = NULL;

   /*
    * Allocate memory for alternative logging structure
    */
   if ((pLogInfo = (MyLogInfo *) malloc(sizeof(MyLogInfo))) ==
NULL)
      return VTK_ERR_OUT_OF_MEMORY;

   /*
    * Init MyLogInfo structure with pointers to file streams
    */
   if ((pLogInfo->pMsgFile = (FILE *)fopen("messages.log", "w")) ==
NULL)
   {
      free(pLogInfo);
      return VTK_ERR_OPEN_LOG_FILE;
   }
   if ((pLogInfo->pVaDataFile = (FILE *)fopen("va_data.log", "w"))
== NULL)
   {
      fclose(pLogInfo->pMsgFile);
      free(pLogInfo);
      return VTK_ERR_OPEN_LOG_FILE;
   }

   /*
    * Pass pointer to application's logging structure in userHdl
variable
    */
   *userHdl = pLogInfo;
   return VTK_OK;
}
```

```
/*
 * MyCloseLog
 *
 * Callback function for closing the alternative logging
 */
void VTK_CALLBACK MyCloseLog(const Vtk_Ctxt *pCtxt, void* userHdl)
{
   MyLogInfo *pLogInfo;

   if (!userHdl)
      return;

   /*
    * Write final logging message
    */
   Vtk_WriteLog(pCtxt, LOG_Info, "Closing log files ...");

   /*
    * Close both log file streams
    */
   pLogInfo = (MyLogInfo *) userHdl;
   fclose(pLogInfo->pMsgFile);
   fclose(pLogInfo->pVaDataFile);

   /*
    * Release memory allocated for MyLogInfo structure
    */
   free(pLogInfo);
   return;
}



/*
 * MyWriteLog
 *
 * Callback function for alternative processing (storing) of
logging messages
 */
void VTK_CALLBACK MyWriteLog(const Vtk_Ctxt *pCtxt,
            enum Vtk_CtxtLogType type, const char *pMsg, void*
userHdl)
{
    struct tm *tmptr;
   struct timeb timebuffer;
   MyLogInfo *pLogInfo;
   char typeStr[5] = "   ";
```

```
   if (!userHdl)
      return;

   pLogInfo = (MyLogInfo *) userHdl;

   /*
    * Set mesage type prefix
    */
   if (type == LOG_Error)
      strcpy(typeStr, "ERR");
   else if (type == LOG_Info)
      strcpy(typeStr, "INF");
   else if (type == LOG_Debug)
      strcpy(typeStr, "DBG");
   else if (type == LOG_VaData)
      strcpy(typeStr, "VAD");

   /*
    * Get current time
    */
   ftime( &timebuffer );
    tmptr = localtime(&(timebuffer.time));


   if (type == LOG_Error || type == LOG_Info || type == LOG_Debug)
      fprintf(pLogInfo->pMsgFile,
              "%04d/%02d/%02d %02d:%02d:%02d.%03hu %s: %s\n",
                tmptr->tm_year+1900, tmptr->tm_mon+1, tmptr-
>tm_mday,
                tmptr->tm_hour, tmptr->tm_min, tmptr->tm_sec,
        timebuffer.millitm, typeStr, pMsg);
   else
      fprintf(pLogInfo->pVaDataFile,
              "%04d/%02d/%02d %02d:%02d:%02d.%03hu %s: %s\n",
                tmptr->tm_year+1900, tmptr->tm_mon+1, tmptr-
>tm_mday,
                tmptr->tm_hour, tmptr->tm_min, tmptr->tm_sec,
        timebuffer.millitm, typeStr, pMsg);

   return;
}
```

Implementing Specialized Validation Processing

This section describes very specialized tasks that an application can perform to further enhance its validation functionality. An application is not required to include these validation functions. Most applications will not implement this functionality. These tasks can be done using the Toolkit API. Each task is described separately followed by sample code. The sample code demonstrates how an application can implement the specialized functionality.

The tasks are as follows:

- Producing Signed Requests
- Checking Delegated VA Certificates
- Setting Proxy Information
- Adding OCSP Extensions
- Getting Validation Handle for Specific Certificate

Producing Signed Requests

OCSP optionally allows an application to produce signed requests. For an environment that requires signed requests, your application needs to:

- Customize the context
- Implement a signing callback

A context is the global Toolkit environment that your application creates using the Vtk_CtxtNew function. Typically, a single context is created in the start-up code of the application, it sets configuration values, and is used in all subsequent function calls for the Toolkit. Your application must customize the context with requestor signing information. It can add this functionality for handling signed requests by calling the Vtk_CtxtSetOption function and setting the Vtk_OCSPSignInfo structure in the context. This structure supplies a signing function callback and signer information to be included in the request. This structure is used only if the option CO_OCSPSignInfo is specified as the value for the Vtk_CtxtOptionType enumeration which defines configuration option types for a Vtk_Ctxt structure. Configuration options can be configured using the Vtk_CtxtSetOption function and can be retrieved using the Vtk_CtxtGetOption function.

Your callback function performs the actual signing. The Vtk_OCSPSignCallBack function enables the application to add signatures to the outgoing requests. The parameters include the following:

- userHdl—Parameter that the Toolkit passes. The application provides this parameter.
- digest—DER encoding of hashed data bytes, that is, the data to be signed.
- sigout—Buffer that the Toolkit allocates for the application to store the resulting signature. The application uses the length parameter to set the size of the signature written.
- maxSigOutBuf—Size of the sigOut buffer allocated by the Toolkit.
- padding—Type of padding the Toolkit requires. Type 1 padding indicates that the Toolkit requires PKCS#1 padding.

When the Toolkit gets to the point of its processing where OCSP signing occurs, the Toolkit calls the call back the application has specified. The application will then create the signature on the token.

Once the application completes its processing, it needs to return VTK_OK to indicate that the OCSP signature has been added and the Toolkit can continue with its processing. If the application encounters an error when it tries to create the signature, it needs to return any nonzero value to terminate the operation and stop the Toolkit from creating the request.

Code Sample for Customizing a Context

The following sample function shows how to create a Vtk_Ctxt structure and customize it for an environment that requires signed requests.

```
/*
 * myOCSPRequestSignCallbackFunction
 *
 * Callback function for producing signed OCSP requests.
 */
int VTK_CALLBACK myOCSPRequestSignCallbackFunction(void* userHdl,
                  Vtk_Buffer *digest, Vtk_Buffer *sigOut,
                  int maxSigOutBuf, int padding);
```

```
/*
 * SignCtxtCreate
 *
 * Creates a Toolkit Context configured for signing of OCSP
 * requests.
 *
 * Parameters:
 * signCert - the signer's certificate
 * appData - application specific data to passed to the callback
 */
static Vtk_Ctxt* SignCtxtCreate(Vtk_Cert *signCert,
                                struct my_sign_st *appData)
{
   Vtk_Ctxt *ctxt = NULL;
   Vtk_uint32 retCode;
   Vtk_CtxtOption ctxtOption;
   Vtk_OCSPSignInfo signInfo;

   /*
    * Create default Toolkit context
    */
   if ((ctxt = Vtk_CtxtNew()) == NULL)
      return NULL;

   /*
    * Setup the request signing callback information for the
    * context.
    */
   signInfo.OCSPSignCB = myOCSPRequestSignCallbackFunction;
   signInfo.signerCert = signCert;
   signInfo.userHdl = (void*)appData; /* any application data */
   ctxtOption.option = CO_OCSPSignInfo;
   ctxtOption.d.aSignInfo = &signInfo;

   if ((retCode = Vtk_CtxtSetOption(ctxt, &ctxtOption)) != VTK_OK)
   {
      showError("Vtk_CtxtSetOption - CO_OCSPSignInfo", retCode);
      Vtk_CtxtDelete(ctxt);
      return NULL;
   }

   return NULL;
} /* SignCtxtCreate */
```

NOTE: The scope of the userHdl parameters needs to be at least as long as that of the Vtk_Ctxt in which it is set. The Toolkit does not interpret these parameters, but instead passes these parameters to the callbacks until the parameters have been reset or the Vtk_Ctxt is no longer valid.

Code Sample for Signing OCSP Requests

The following sample provides skeleton code that shows a signing callback function an application can set through the Vtk_OCSPSignInfo structure. This function can then be called every time an OCSP request is created and needs signing. The callback function can be triggered by either calling Vtk_ValidationValidate or Vtk_ValidationGetQueries.

```
/*
 * myOCSPRequestSignCallbackFunction
 *
 * Callback function for signed OCSP requests.
 * This function will get invoked every time the Toolkit
 * requires a signature on the request.
 */
int VTK_CALLBACK myOCSPRequestSignCallbackFunction(void* userHdl,
         Vtk_Buffer *digest, Vtk_Buffer *sigOut, int maxSigOutBuf,
         int padding)
{
   struct mySign_st *signHdl;

   /*
    * This sample callback requires and sets the userHdl parameter.
    */
   if (userHdl == NULL)
      return VTK_ERR_USER_CALLBACK;

   /*
    * Convert the application handle to proper type - this
    * parameter is set when configuring the context;
    * The application determines whether this parameter is set.
    */
   signHdl = (struct mySign_st*) userHdl;

   if (padding == 1) /* PKCS #1 padding required */
   {
```

```
            /*
             * The digest parameter contains the DER encoding of the
             * data that needs to be encrypted with the private key.
             * The resulting data needs to be written to the sigOut
             * buffer and the length set in the sigOut->len field.
             * The Toolkit allocated the data in sigOut
             * structure and it is of size maxSigOutBuf.
             */

            /*
             * The application performs the following:
             * -generate the signature on digest buffer
             * -write resulting signature to sigOut->dPtr
             *  buffer setting the sigOut->length; the toolkit
             *  has allocated space in sigOut buffer of size
             *  maxSigOutBuf
             */


         } else
            return VTK_ERR_USER_CALLBACK;


         return VTK_OK;
      } /* myOCSPRequestSignCallbackFunction */
```

Checking Delegated VA Certificates

An application can check delegated VA certificates and determine whether to accept or reject the certificate used to validate the validation response. An application can do this using the Vtk_DelegatedIssuerCallback function, but only in VA or CA delegated trust paradigms.

The Vtk_Callback structure sets the callback function and parameter for the context. This structure is used only if the option CO_DelegatedIssuerCB is specified as the value for the Vtk_CtxtOptionType enumeration which defines configuration option types for a Vtk_Ctxt structure.

Configuration options can be configured using the Vtk_CtxtSetOption function and can be retrieved using the Vtk_CtxtGetOption function.

The Toolkit calls your application's callback function when verifying OCSP or CRT responses in delegated mode. In delegated mode, the certificate in the OCSP or CRT response is from a non-trusted VA derived by chaining to the VA's or CA's trusted certificate. This means the signer of the response is not directly part of the trusted VA or CA certificate store.

The Toolkit first checks the context for the chain from the delegated certificate to the trusted root. It then verifies the non-trusted VA's certificate using the trusted certificate. Finally, the Toolkit calls the application's callback function during its checking of OCSP and CRT responses in delegated trust models to see if the certificate should be accepted.

Once the application completes its processing, it returns VTK_OK to indicate that the certificate is good and can be used for validating the response. If the application has detected a problem and does not want the Toolkit to use the certificate to validate the response, it returns a nonzero value.

Code Sample for Checking Delegated Certificates

```
/*
 * myDelegatedCallbackFunction
 *
 * Delegated issuer check callback function.
 */
int VTK_CALLBACK myDelegatedCallbackFunction(void *userHdl,
         const Vtk_Cert *delegatedCert,
         const Vtk_Cert *trustedCert);


/*
 * DelegatedCtxtCreate
 *
 * Creates a ValiCert Validator Toolkit Context configured with
 * a callback for delegated OCSP/CRT VA trust models.
 *
 * Parameters:
 * appData - application specific data to be passed to the callback
 */
static Vtk_Ctxt* DelegatedCtxtCreate(struct my_sign_st *appData)
{
 Vtk_Ctxt *ctxt = NULL;
 Vtk_uint32 retCode;
 Vtk_CtxtOption ctxtOption;
 Vtk_Callback cb;

 /*
 * Create default toolkit context
 */
 if ((ctxt = Vtk_CtxtNew()) == NULL)
 return NULL;
```

```
 /*
 * Set the delegation certificate callback function.
 * This callback will be used when checking the validation response
 * from a VA which operates in a delegated model.
 */
 /*
 * Set my callback structure fields.
 */
 cb.f.DelCertCB = myDelegatedCallbackFunction;
                                    /* application function */
 cb.userHdl = appData; /* application parameter */

 /*
 * Set ctxt option data
 */
 ctxtOption.option = CO_DelegatedCertCB; /* callback option */
 ctxtOption.d.aCB = &cb; /* callback option data */

 if ((retCode = Vtk_CtxtSetOption(ctxt, &ctxtOption)) != VTK_OK)
 {
 showError("Vtk_CtxtSetOption - CO_DelegatedIssuerCB", retCode);
 Vtk_CtxtDelete(ctxt);
 return NULL;
}

 return ctxt;
} /* DelegatedCtxtCreate */


/*
 * myDelegatedCallbackFunction
 *
 * The delegation certificate callback function.
 * This callback will be used when checking the validation response
 * from a VA which operates in a delegated model.
 *
 * This sample function will just display the delegated issuer
 * certificate.
 *
 * Parameters:
 * userHdl - application specified pointer set when installing the
 *           callback function in the toolkit context
 * delegatedCert - responder certificate
 * trustedCert - trusted issuer of the delegatedCert (this is one of
 *               the certificates set in the Vtk_Ctxt)
 */
```

```
int VTK_CALLBACK myDelegatedCallbackFunction(void *userHdl,
       const Vtk_Cert *delegatedCert, const Vtk_Cert *trustedCert)
{
   myDelegCallback_st *cbStruct;
   Vtk_uint32 retCode;
   Vtk_CertInfo *certInfo = NULL;

   /*
    * This example has set the userHdl when setting the callback in
    * the Toolkit context, so it requires the handle in this call.
    */
   if (userHdl == NULL)
      return VTK_ERR_USER_CALLBACK;

   cbStruct = (myDelegCallback_st*)userHdl;

   /*
    * This function will obtain the details of the VA and its
    * issuer's certificates.
    */

   /*
    * The callback function userHdl has stored the Toolkit context
    * so that it can use it in this call.
    */
   if ((retCode = Vtk_CertGetInfo(cbStruct->ctxt,
         delegatedCert,&certInfo)) != VTK_OK)
   {
      showError("Vtk_CertGetInfo", retCode);

      /*
       * Returns VTK_OK, but if needed the application
       * could return VTK_ERR_USER_CALLBACK to indicate
       * that it should not trust the VA's certificate.
       */
      return VTK_OK;

   }

   /*
    * Now that we have the certificate details, display them.
    */
   displayCert("Delegated VA certificate", certInfo);

   /*
    * No longer need the certificate information; delete it.
    */
   Vtk_CertInfoDelete(certInfo);
```

```
   /*
    * Get the details of the issuer of the VA's certificate.
    */
   if ((retCode = Vtk_CertGetInfo(cbStruct->ctxt, trustedCert,
          &certInfo)) != VTK_OK)
   {
      showError("Vtk_CertGetInfo", retCode);
      return VTK_OK;
   }

   /*
    * Display the VA's issuer certificate.
    */
   displayCert("VA's issuer certificate", certInfo);


   /*
    * No longer need the certificate information; delete it.
    */
   Vtk_CertInfoDelete(certInfo);

   return VTK_OK;
} /* myDelegatedCallbackFunction */
```

Setting Proxy Information

An application can set information about the HTTP proxy using the Vtk_ProxyInfo structure. This structure is used to specify the proxy host and port number used in the OCSP, CRT, and HTTP-CRL protocols. It is not used for LDAP-CRL.

This structure is used only if the option CO_HTTPProxy is specified as the value for the Vtk_CtxtOptionType enumeration which defines configuration option types for a Vtk_Ctxt structure. Configuration options can be configured using the Vtk_CtxtSetOption function and can be retrieved using the Vtk_CtxtGetOption function.

Code Sample for Setting Proxy Information

```
/*
 * setProxyInfo
 *
 * Function to set the HTTP Proxy information to use by the ValiCert
 * Validator Toolkit.
 *
 * Parameters:
 * ctxt - a valid ValiCert Toolkit context previously created
 * with Vtk_CtxtNew function call
 * proxyHost - host machine for the proxy (e.g. "merced")
 * proxyPort - port for the proxy
 */
int setProxyInfo(Vtk_Ctxt *ctxt, char *proxyHost, int proxyPort)
{
 Vtk_ProxyInfo proxyInfo;
 Vtk_CtxtOption ctxtOption;
 Vtk_uint32 ret;

 /*
 * set the proxy info structure with supplied data
 */
 proxyInfo.host = proxyHost;
 proxyInfo.port = proxyPort;

 /*
 * set the ctxt option structure for use with proxy info
 */
 ctxtOption.option = CO_HTTPProxy;
 ctxtOption.d.aProxyInfo = &proxyInfo;

 /*
 * set the ctxt option
 */
 ret = Vtk_CtxtSetOption(ctxt, &ctxtOption); assert(ret == VTK_OK);
 return 0;
} /* setProxyInfo */
```

Adding OCSP Extensions

An application can add OCSP extension to an entire OCSP validation request or to a single certificate within the request. A validation request can be made up of one or more certificates. An extension can be any data that the application wants to add to a validation request.

To add an extension, an application must allocate memory for the Vtk_Extension structure and creates an empty extension structure using the

The Vtk_ExtensionNew extension function. The application can then call the Vtk_ExtensionInit function to initialize the Vtk_Extension data structure, The data structure is based on the data the application passes to it.

Once the extension is created and initialized, the application can add the extension to an entire validation request or to a specific certificate in the validation query. To add to an entire validation request, the application must call the Vtk_ValidationAddReqExt function. An extension added using this function applies to all the certificates in the request. To add an extension to a specific certificate in the query, the application has two options. One option is for the application to use the Vtk_ValidationAddReqExtForSingleCert, which uses the certificate and issuer certificate information passed in this function to identify the certificate to which the application wants an extension added. The other option is to use the Vtk_ValidationAddReqExtForSingleCertHdl function, which uses the Vtk_ValHdl passed in this function to identify the certificate to which the application wants an extension added.

NOTE: If the application wants to identify the certificate by means of its validation handle using this function, the application must first obtain the Vtk_ValHdl structure by calling the Vtk_AddCert, Vtk_AddCertRaw, or Vtk_ValidationGetValHdl function.

Code Sample for Adding OCSP Extensions

```
/*
 * AddRequestExtension
 *
 * Adds an extension to the OCSP query associated with a particular
 * certificate (identified through Vtk_ValHdl) and to the entire
 * OCSP request.
 *
 * Parameters:
 * ctxt - ValiCert Toolkit context
 * q - validation query structure
 * hdl - validation handle identifying the particular certificate
 *       information in the OCSP query for which the extension
 *       is to be added
 */
int AddRequestExtension(Vtk_Ctxt *ctxt, Vtk_Validation *q,
              Vtk_ValHdl *hdl)
{
 const char extOid[] = "1.2.3";
```

```
const char extData[] = "testMsg";
Vtk_uint32 retCode;
Vtk_Extension *ext = NULL;
Vtk_Buffer oid;
Vtk_Buffer data;


/*
* Initialize OID buffer
*/
oid.type = VTK_DF_STRING;
oid.dPtr = (Vtk_Byte*) extOid;
oid.len = strlen(extOid);

/*
* Initialize data buffer
*/
data.type = VTK_DF_STRING;
data.dPtr = (Vtk_Byte*) extData;
data.len = strlen(extData) + 1;


/*
* Create an extension object
*/
ext = Vtk_ExtensionNew(ctxt); assert(ext);


/*
* Initialize with extension data
*/
if ((retCode = Vtk_ExtensionInit(ctxt, ext, &oid, 0, &data))
                          != VTK_OK)
{
Vtk_ExtensionDelete(ext);
showError("Vtk_ExtensionInit", retCode);
return -1;
}

/*
* Add extension to the validation query for a specific
* certificate.
*/
if ((retCode = Vtk_ValidationAddReqExtForSingleCertHdl(ctxt, hdl,
          ext)) != VTK_OK)
```

```
 {
 Vtk_ExtensionDelete(ext);
 showError("Vtk_ValidationAddReqExtForSingleCertHdl", retCode);
 return -1;
}


 /*
 * Add extension to the entire OCSP request.
 */
 if ((retCode = Vtk_ValidationAddReqExt(ctxt, q, ext)) != VTK_OK)
 {
 Vtk_ExtensionDelete(ext);
 showError("Vtk_ValidationAddReqExt", retCode);
 return -1;
}


 /*
 * Delete the extension data
 */
 Vtk_ExtensionDelete(ext);
 return VTK_OK;
} /* AddRequestExtension */

/*
 * GetResponseExtension
 *
 * Obtains an OCSP extension from an OCSP response associated with
 * a particular certificate (identified through Vtk_ValHdl) and
 * from the set of extensions present for the entire response.
 *
 * Parameters:
 * ctxt - ValiCert Toolkit context
 * hdl - Validation handle identifying the particular certificate
 *       information in the OCSP query for which the extension
 *       is to be obtained
 */
int
GetResponseExtension(const Vtk_Ctxt *ctxt, Vtk_ValHdl *hdl)
{
   Vtk_Extension *ext = NULL;
   const char extOid[] = "1.2.3";
   Vtk_Buffer oid;
   Vtk_uint32 retCode;
   Vtk_uint32 status;
   Vtk_ValRespDetails *respDetails = NULL;
   Vtk_ValRespSingleCertDetails *certDetails = NULL;
```

```
/*
 * Initialize Oid buffer
 */
oid.type = VTK_DF_STRING;
oid.dPtr = (Vtk_Byte*) extOid;
oid.len = strlen(extOid);

/*
 * Obtain the extension from the response.
 */
if ((retCode = Vtk_ValHdlGetRevStatus(ctxt, hdl, &status,
       &respDetails, &certDetails)) != VTK_OK)
{
   showError("Vtk_ValHdlGetRevStatus", retCode);
   return -1;
}

/*
 * First get the extension from the overall response extensions.
 */
if (respDetails)
{
   retCode = Vtk_ExtensionGetByOid(ctxt,
         respDetails->extensions, &oid, &ext);
   if (retCode == VTK_ERR_NOT_FUND)
   {
      printf("\nSpecified extension (%s) not found on OCSP
     response.", extOid);
   }
   else if (retCode == VTK_OK)
   {
      printf("\nFound extensions %s in OCSP response.");

      /*
       * Delete the extension data.
       */
      Vtk_ExtensionDelete(ext);
   }
   else
      showError("Vtk_ExtensionGetByOid", retCode);
}


/*
 * Get the extension from the specific certificate OCSP reply.
 */
if (certDetails)
{
```

```
      retCode = Vtk_ExtensionGetByOid(ctxt,
                certDetails->extensions, &oid, &ext);
      if (retCode == VTK_ERR_NOT_FUND)
      {
         printf("\nSpecified extension (%s) not found on OCSP
          response.", extOid);
      }
      else if (retCode == VTK_OK)
      {
         printf("\nFound extensions %s in OCSP response.");

         /*
          * Delete the extension data.
          */
         Vtk_ExtensionDelete(ext);
      }
      else
         showError("Vtk_ExtensionGetByOid", retCode);
   }


   /*
    * Delete the OCSP response details.
    */
   Vtk_ValRespDetailsDelete(respDetails);
   Vtk_ValRespSingleCertDetailsDelete(certDetails);

   return VTK_OK;
} /* GetResponseExtension */
```

Getting Validation Handle for Specific Certificate

The Vtk_ValidationGetValHdl validation function creates a validation handle for a specific certificate within a validation query. This function is useful if the application added certificates to the query using the Vtk_ValidationAddCertChain function or did not specify the Vtk_ValHdl when it added the certificate using the Vtk_ValidationAddCert or Vtk_ValidationAddCertRaw function.

Code Sample for Getting Validation Handle

```
/*
 * getValHdl
 *
 * Function to obtain a Vtk_ValHdl from a Vtk_Validation structure.
 * A Vtk_ValHdl can be obtained either at time of adding the
 * certificate to the Vtk_Validation structure or by using the
 * Vtk_ValidationGetValHdl function.
 *
 * Parameters:
 * ctxt - ValiCert Validator Toolkit context
 * val - Vtk_Validation structure
 * cert - certificate for which to obtain the validation handle
 * issuerCert - issuer certificate of the "cert"; this can be
 *    NULL if the issuer certificate has been previously
 *    added to the Vtk_Ctxt by calling Vtk_CtxtAddCert(s).
 */
int getValHdl(const Vtk_Ctxt *ctxt, Vtk_Validation *val,
     const Vtk_Cert *cert, const Vtk_Cert *issuer)
{
   Vtk_ValHdl *hdl = NULL;
   Vtk_uint32 ret;


   /*
    * First add the certificate to the Vtk_Validation structure
    */
   if ((ret = Vtk_ValidationAddCert(ctxt, val, cert, issuer, NULL))
         != VTK_OK)
   {
      showError("Vtk_ValidationAddCert", ret);
      return -1;
   }

   /*
    *                  IMPORTANT
    *
    * The application would need to continue with its normal
    * processing such as perform the validation or add more
    * certificates to be validated. The code for this processing
    * could be inserted here. Once the application completes
    * its processing, the application can obtain the validation
    * handle for the certificate using the sample code below.
    *
    */
```

```
   /*
    * Obtain the validation handle for the certificate
    */
   if ((ret = Vtk_ValidationGetValHdl(ctxt, val, cert, issuer,
          &hdl)) != VTK_OK)
   {
      showError("Vtk_ValidationGetValHdl", ret);
      return -1;
   }

   /*
    * The validation handle can now be used to obtain certificate
    * revocation details through using the
    * Vtk_ValidationAddReqExtForSingleCertHdl function.
    */


   /*
    * Free any resources associated with the validation handle.
    */
   if (hdl)
      Vtk_ValHdlDelete(hdl);

   return 0;
} /* getValHdl */
```

CHAPTER
3

Toolkit Reference

This section provides reference information about data structures and functions available to developers who are integrating validation into their applications.

- Constants
- Enumerations
- Data Structures
- Callback Functions
- Functions

For task-oriented information on how to use these functions in your application, refer to Chapter 2, "Using the Toolkit."

Constants

The Toolkit provides one constant that is used by a few of the Toolkit functions.

VTK_GVAS_URL

```
const char* VTK_GVAS_URL;
```

Description

A constant that defines the URL for the ValiCert Global VA Service. It can be used to specify this URL in several Toolkit function calls.

Parameters

VTK_GVAS_URL	Symbol that can be used to define the Global VA Service URL.

Notes

None

See Also

“Vtk_CtxtSetDefaultVa” on page 164

“Vtk_CtxtSetVaInfo” on page 168

“Vtk_ValidationSetVaInfo” on page 221

Enumerations

The Toolkit provides several enumerated types that are used by the Toolkit data structures and functions. They are listed in alphabetical order.

Vtk_CtxtLogType

```
enum Vtk_CtxtLogType {
    LOG_Error = 0,
    LOG_Info,
    LOG_Debug,
    LOG_VaData
};
```

Description

This enumerated type specifies the different types of logging messages.

Parameters

LOG_Error	Messages that explain an error that caused the program to stop.
LOG_Info	Messages that describe toolkit activities, such as loading a certificate from a file, setting default VA information, or sending a request to a VA.
LOG_Debug	Detailed information which usually accompanies an error message for example the value of some variables when an error occurred or information about each successfully called function.
LOG_VaData	Validation data that is either request data sent to a VA or response data received from a VA.

Notes

Log types are also used to specify the level of logging in the structure Vtk_LogOptions. Specifying a level will also include all logging below. For example specify the log level of LOG_Debug to include LOG_Debug, LOG_Info and LOG_Error messages. Specify log level of LOG_VaData to include all log messages.

See Also

“Vtk_LogOptions” on page 99

Vtk_CtxtOptionType

```
enum Vtk_CtxtOptionType{
    CO_EnableRelocationProtocol,
    CO_EnableServiceLocatorExt,
    CO_ClientInfoExt,
    CO_OCSPNonceExt,
    CO_HTTPProxy,
    CO_MaxTimeSkew,
    CO_UseAIAData,
    CO_LoadLDAPLib,
    CO_CRLCacheDir,
    CO_MaxCrlCacheTime,
    CO_CrlNoNextUpdateCacheTime,
    CO_LDAPSearchTimeout,
    CO_DelegatedIssuerCB,
    CO_OCSPSignInfo,
};
typdef struct Vtk_CtxtOption_st Vtk_CtxtOption;

struct Vtk_CtxtOption_st {
    enum Vtk_CtxtOptionType option;
    union {
           char* aChar;
           int anInt;
           Vtk_ProxyInfo * aProxyInfo;
           Vtk_Callback *aCB;
           Vtk_OCSPSignInfo *SignInfo;
    } d;
};
```

Description

An enumeration that defines configuration option types for a Vtk_Ctxt structure. These options can be configured using the Vtk_CtxtSetOption function and can be retrieved using the Vtk_CtxtGetOption function.

The Vtk_CtxtOption structure is used to pass Vtk_Ctxt option data in these functions. The option can be specified as any of the following data types using the corresponding variable shown in parentheses:

- integer (d.anInt)
- character (d.char)

- structure (d.aProxyInfo, d.aCB, d.aSignInfo)

Parameters

CO_EnableRelocationProtocol	Type: int Enables or disables the ValiCert Relocation protocol used in validation responses to inform the Toolkit which validation responder to use. Typically, the protocol selects the geographically closest responder. To set the relocation information in the validation responses, the protocol uses the time zone information which is passed in the validation request as an extension. The possible values are 0 and 1. The default is 1, enable the relocation protocol. This option is not supported at this time.
CO_EnableServiceLocatorExt	Type: int Includes or excludes the service locator request extension. This extension allows an OCSP server to reroute a request to the OCSP server authorized to sign the certificate. It applies to OCSP requests only. The possible values are 0 and 1. The default is 1, include the service locator extension.
CO_ClientInfoExt	Type: int Includes or excludes the client information extension. This extension identifies the UserAgent, OCSP Client, or CRT Client used to make the validation request. It applies to OCSP/ CRT requests only. The possible values are 0 and 1. The default is 1, include the client information extension.
CO_OCSPNonceExt	Type: int Includes or excludes the nonce extension. This extension cryptographically binds a request and response to prevent replay attacks. This extension can be configured for OCSP requests and responses. The possible values are 0 and 1. The default is 1, include the OCSP Nonce extension.

CO_HTTPProxy	Type: Vtk_ProxyInfo Information about the HTTP Proxy to use for CRL and OCSP/CRT over HTTP. The information includes the port and host for the proxy. Note that this is not used for LDAP-CRL.
CO_MaxTimeSkew	Type: int Maximum time difference, in seconds, allowed between the client and the server. It can be any integer. The default is 300 seconds. Tip: If the difference between the times is greater than this configured value, you may see many responses that indicate the response is expired or not yet valid.
CO_UseAIAData	Type: int Uses the Authority Information Access (AIA) certificate extension to determine which VA to use to validate a certificate. The AIA makes checks for a VA in the following: ❖ certificate ❖ CA ❖ context The default is 0, do not use AIA.
CO_LoadLDAPLib	Type: int Enables or disables the force loading of the Netscape LDAP SDK that ships with the Toolkit. If this option is enabled, it forces loading of the LDAP library when the LDAP function is called. This option applies only if checking CRLs over LDAP. The possible values are 0 and 1. The default is 0, do not load the library until needed.
CO_CrlCacheDir	Type: char Directory location for caching the CRL. This can be any valid directory. The default is the VtkCrlCacheDir in the current directory of the application.

CO_MaxCrlCacheTime	Type: int Maximum cache duration of CRLs, in seconds. The default cache duration is until the CRL expires.
CO_CrlNoNextUpdateCacheTime	Type: int Determines whether the no nextUpdate CRLs (that is CRLs without a nextUpdate field) are cached along with the other CRLs in the directory defined by the CO_CrlCacheDir option. The default is -1, do not cache the no next Update CRLs. A positive value results in CRLs cached for the specified number of seconds.
CO_LDAPSearchTimeOut	Type: int Maximum number of seconds that the client waits for a response from the LDAP server before returning an error. The default is 120 seconds.
CO_DelegatedIssuerCB	Type: Vtk_Callback Callback for the delegated OCSP/CRT response issuer. See “Vtk_Callback” on page 85
CO_OCSPSignInfo	Type:Vtk_OCSPSignInfo OCSP signing information. See “Vtk_LogOptions” on page 99.

Notes

None

See Also

“Vtk_CtxtOptionType” on page 95

“Vtk_CtxtGetOption” on page 158

“Vtk_CtxtOptionDeleteContent” on page 162

“Vtk_CtxtSetOption” on page 166

Vtk_DataFormat

```
enum Vtk_DataFormat{
   VTK_DF_STRING,
   VTK_DF_DER,
   VTK_DF_BASE64,
   VTK_DF_HEX,
};
```

Description

An enumeration that defines the format of data passed between the Toolkit and the application.

Parameters

VTK_DF_STRING	Null-terminated printable string
VTK_DF_DER	Distinguished Encoding Rules
VTK_DF_BASE64	Base 64 encoding
VTK_DF_HEX	Hexadecimal encoding

Notes

None

See Also

None

Vtk_DataType

```
enum Vtk_DataType{
   VTK_DT_CRL = 0,
   VTK_DT_PKCS7,
};
```

Description

An enumeration that defines the type of data passed between the Toolkit and the application. This enumeration is used with only the CRL protocol.

Parameters

VTK_DT_CRL	CRL data is being passed.
VTK_DT_PKCS7	PKCS7 data is being passed. This is a wrapper for CRL data.

Notes

None

See Also

“Vtk_CRLProtocolDetails” on page 91

“Vtk_ProtocolDetails” on page 102

Vtk_RevocationReason

```
enum Vtk_RevocationReason{
   VTK_REV_STATUS_UNKNOWN = -1,
   VTK_REV_UNSPECIFIED = 0,
   VTK_REV_KEY_COMPROMISE,
   VTK_REV_CA_COMPROMISE,
   VTK_REV_AFFILIATION_CHANGED,
   VTK_REV_SUPERSEDED,
   VTK_REV_CESSATION_OF_OPERATION,
   VTK_REV_CERTIFICATE_HOLD,
   VTK_REV_REMOVE_FROM_CRL
};
```

Description

An enumeration that defines the reason a certificate has been revoked. The revoking entity optionally provides this information at time of revocation. This enumeration is returned in the Vtk_ValRespSingleCertDetails structure.

VTK_REV_STATUS_UNKNOWN	Certificate status is unknown because the revocation reason field was not present in the response.
VTK_REV_UNSPECIFIED	Certificate has been revoked for an unspecified reason. This is a catchall category.
VTK_REV_KEY_COMPROMISE	the private key has been compromised.
VTK_REV_CA_COMPROMISE	CA signing the certificate has been compromised.
VTK_REV_AFFILIATION_CHANGED	The owner of the certificate is no longer affiliated with the organization.
VTK_REV_SUPERSEDED	Certificate has been superseded by another certificate.
VTK_REV_CESSATION_OF_ OPERATION	Operations for the entity has terminated. For example, if a company is no longer conducting business.
VTK_REV_CERTIFICATE_HOLD	Certificate has been placed on hold.
VTK_REV_REMOVE_FROM_CRL	Certificate serial number has been removed from the CRL.

Notes

None

See Also

“Vtk_ValRespSingleCertDetails” on page 109

“Vtk_ValHdlGetRevStatus” on page 195

“Vtk_ValidationGetRevStatus” on page 212

Vtk_ValidationMech

```
enum Vtk_ValidationMech {
    VTK_VM_CRT = 1,
    VTK_VM_OCSP,
    VTK_VM_CRL
};
```

Description

This enumeration specifies the validation mechanism employed by the application to validate certificates. It is used in the Vtk_CtxtSetDefaultVa, Vtk_CtxtSetVaInfo, and Vtk_ValidationSetVaInfo functions.

Parameters

VTK_VM_CRT	Certificate Revocation Trees (CRTs). High-performance validation method proprietary to ValiCert. This is the default.
VTK_VM_OCSP	Online Certificate Status Protocol. This validation method requires a certificate recipient to check certificate status by sending a request to an OCSP server. This method provides up-to-date certificate status information, but requires network communication to the OCSP server.
VTK_VM_CRL	Certificate Revocation Lists (CRLs). This validation method requires that the verifier download CRLs published by CA and confirm that certificate is not on the list. This method is provided to accommodate legacy CRL-based systems.

Notes

None

See Also

“Vtk_ProxyInfo” on page 104

“Vtk_ValRespDetails” on page 107

“Vtk_ValRespSingleCertDetails” on page 109

Data Structures

The Toolkit provides several data structures that are used by the Toolkit functions. The data structure can have one of the following type definitions:

- integer (signed and unsigned)
- structure
- char (signed and unsigned)
- constant

Vtk_Buffer

```
typedef struct {
   enum Vtk_DataFormat type;
   Vtk_Byte* dPtr;
   Vtk_uint32 len;
} Vtk_Buffer;
```

Description

This is a general purpose structure used to pass data between the Toolkit or VA and your application. It defines information such as the encoding method employed and the length of the buffer. The most common use of this buffer is to pass certificate data, which is typically 1K in length.

It is used in the Vtk_ValidationAddCertRaw, Vtk_CertInit, Vtk_ExtensionsGetByOid, Vtk_CtxtSetOption, and Vtk_CtxtGetOption functions. It is also used in the Vtk_ValQuery structure for request and response information.

Parameters

type	Type of encoding applied to the data contained in the buffer. The possible values are: ❖ VTK_DF_STRING—null-terminated printable string ❖ VTK_DF_DER—DER encoding ❖ VTK_DF_BASE64—Base64 encoding ❖ VTK_DF_HEX—Hexadecimal encoding For more information about these encoding types, see “Vtk_DataFormat” on page 76.
dPtr	Pointer to the location of the buffer.
len	Length of the buffer.

Notes

The Toolkit provides the VTK_BUF_INIT macro for initializing the Vtk_Buffer and VTK_EMPTY_BUF for checking whether it is empty.

When the buffer is initialized, Vtk_Buffer contains the following values:

```
Vtk_DataFormat = VTK_DF_STRING
dPtr = NULL
len = 0
```

When checking if the buffer is empty, the following is checked:

```
dPtr == NULL
len < 0
```

See Also

“Vtk_DataFormat” on page 76

“Vtk_ValQuery” on page 111

Vtk_Byte

```
typedef unsigned char Vtk_Byte;
```

Description

This unsigned character is used within the Vtk_Buffer data structure to point to the location of the buffer.

Parameters

None

Notes

None

See Also

“Vtk_Buffer” on page 82

Vtk_Callback

```
typedef struct {
   union {
          Vtk_DelegatedIssuerCallback DelIssuerCB;
   }f;
   void* userHdl;
}Vtk_Callback;

struct Vtk_CtxtOption_st
{
   enum Vtk_CtxtOptionType option;
   union {
          char* aChar;
          int anInt;
          Vtk_ProxyInfo *aProxyInfo;
          Vtk_Callback *aCB;
          Vtk_OCSPSignInfo *aSignInfo;
   } d;
};
```

Description

This structure sets the callback function and parameter for the context. This structure is used only if the option CO_DelegatedIssuerCB is specified as the value for the Vtk_CtxtOptionType enumeration. The CO_DelegatedIssuerCB option only applies if the CA is issuing delegated certificates.

Parameters

DelCertCB	An application callback function executed when verifying OCSP and CRT responses. This function can be invoked only if issuer of the OCSP/CRT response is delegated by the CA or VA.
userHdl	Application specific data pointer. The Toolkit calls this parameter. The Toolkit treats this pointer as opaque data.

Notes

None

See Also

“Vtk_CtxtLogType” on page 69

Vtk_Cert

```
typedef void* Vtk_Cert;
```

Description

This structure contains an X.509 certificate. Although the certificate structure is opaque to your application, the Toolkit provides several functions that allow your application to extract specific fields from within the X.509 certificate. In addition, it provides parsing functions that allow your application to convert, encode and decode certificate structures.

This structure is used throughout the Toolkit, wherever an X.509 certificate is required.

Parameters

None

Notes

None

See Also

None

Vtk_CertInfo

```
typedef struct {
   Vtk_uint32 version;
   Vtk_Buffer serialNum;
   Vtk_Buffer signatureAlg;
   Vtk_Buffer issuer;
   time_t notBefore;
   time_t notAfter;
   Vtk_Buffer subject;
   Vtk_Buffer publicKeyAlg;
   Vtk_Buffer* issuerUniqueId;
   Vtk_Buffer* subjectUniqueId;
} Vtk_CertInfo;
```

Description

This structure contains detailed X.509 certificate information. It is returned in the Vtk_CertGetInfo function.

Parameters

version	Version number of the encoded certificate.
serialNum	Unique serial number of the certificate assigned by the CA.
signatureAlg	Algorithm used by the CA to sign the certificate, for example, RSA.
issuer	Name of the CA that has signed and issued the certificate. The application supplies the list of trusted CAs in the Vtk_Ctxt structure.
notBefore	Start time of the certificate's validity period.
notAfter	End time of the certificate's validity period.
subject	Holder of the private key for which the public key is being certified.
publicKeyAlg	Encryption algorithm and hashing algorithm used in the public key. For example, RSA and SHA1 or RSA with MD5.
issuerUniqueId	Alternative identifier for issuer. This is an optional parameter that is rarely used.

subjectUniqueId	Alternative identifier for subject. This is an optional parameter that is rarely used.

Notes

None

See Also

“Vtk_CertGetInfo” on page 129

“Vtk_CertInfoDelete” on page 133

Vtk_CertStore

```
typedef void* Vtk_CertStore;
```

Description

This structure serves as a container for one or more X.509 certificates. It is opaque to your application.Your application can use this structure to pass certificate lists to and from Toolkit functions.

This structure is used in many functions that allow you to add, delete, load, and create certificate stores.

Parameters

none

Notes

None

See Also

“Vtk_CertStoreAddCert” on page 141

“Vtk_CertStoreAddCertRaw” on page 143

“Vtk_CertStoreDelete” on page 145

“Vtk_CertStoreLoadFromFile” on page 146

“Vtk_CertStoreNew” on page 148

Vtk_CRLProtocolDetails

```
typedef struct {
   enum Vtk_DataFormat encoding;
   enum Vtk_DataType type;
}Vtk_CRLProtocolDetails;
```

Description

This structure is used to specify detailed information specific to the CRL protocol. This structure is used in the Vtk_ProtocolDetails structure.

Parameters

encoding	An enumeration that defines the format of the CRL. See “Vtk_DataFormat” on page 76 for possible values.
type	An enumeration that defines the structure of the CRL. This can be PKCS7 and CRL. See “Vtk_DataType” on page 77 for possible values.

Notes

None

See Also

“Vtk_ProtocolDetails” on page 102

“Vtk_CtxtSetVaInfo” on page 168

Vtk_CRLRespDetails

```
typedef struct Vtk_CRLRespDetails_st
Vtk_CRLRespDetails;

struct Vtk_CRLRespDetails_st {
   time_t nextUpdate,
};
```

Description

This structure is used to return detailed header information specific to the CRL protocol. It is only used within the Vtk_ValRespDetails structure which returns revocation information for the entire validation response.

Parameters

nextUpdate	Time at which the CRL is expected to be updated again.

Notes

None

See Also

“Vtk_ValRespDetails” on page 107

Vtk_CRTRespDetails

```
typedef struct Vtk_CRTRespDetails_st
Vtk_CRTRespDetails;

struct Vtk_CRTRespDetails_st {
   Vtk_uint32 minVersionToRead,
   time_t nextUpdate,
   time_t validUntil,
   Vtk_Buffer hashAlgorithm,
   Vtk_uint32 fullTreeLeafCount,
};
```

Description

This structure is used to return detailed header information specific to the CRT protocol. It is only used within the Vtk_ValRespDetails structure which returns revocation information for the entire validation response.

Parameters

minVersionToRead	Minimum version of the CRT protocol required to process the response.
nextUpdate	Time at which the CRT is expected to be updated again.
validUntil	Time that the CRT expires.
hashAlgorithm	Hash algorithm used for creating the CRT/hash tree.
fullTreeLeafCount	Number of leaf nodes in the CRT tree.

Notes

None

See Also

“Vtk_ValRespDetails” on page 107

Vtk_Ctxt

```
typedef void* Vtk_Ctxt;
```

Description

The validation context structure. It will store information, such as VA URL for a particular CA, trusted VA certificates, and trusted CA certificates, which is persistent over validation checks. In addition, default validation options such as relocation information usage, usage of service locator extension will be stored in this structure. The context is initialized with the Global VA Service as the default VA and CRT as the default protocol.

For usage information, see Step 2 “Create a Context” on page 11.

Notes

None

See Also

“Vtk_CtxtSetDefaultVa” on page 164

Vtk_CtxtOptionType

```
typedef struct Vtk_CtxtOption_st Vtk_CtxtOption;

struct Vtk_CtxtOption_st {
   enum Vtk_CtxtOptionType option;
   union {
          char* aChar;
          int anInt;
          Vtk_ProxyInfo * aProxyInfo;
          Vtk_Callback *aCB;
          Vtk_OCSPSignInfo *SignInfo;
   } d;
};
```

Description

A structure that defines the configuration option for a Vtk_Ctxt structure. These options can be configured using the Vtk_CtxtSetOption function and can be retrieved using the Vtk_CtxtGetOption function.

The Vtk_CtxtOption structure is used to pass Vtk_Ctxt option data in these functions. The option can be specified as either an integer, char, Vtk_ProxyInfo structure, Vtk_CallBack structure, or Vtk_OCSPSignInfo structure. The data type for the option is shown in parentheses within the description of each option.

Parameters

option	One of the Vtk_CtxtOptionType values.
aChar	Character string that applies, for example, when the CO_CrlCacheDir context type option is specified.
anInt	Integer type that applies when many of the context type options are specified.
aProxyInfo	Proxy structure that applies when the CO_HTTP context option type is specified.

aCB	Callback that applies, for example, when the CO_DelegatedIssuerCB is specified.
aSignInfo	OCSP Sign Information structure that applies, for example, when the CO_OCSPSignInfo context option is specified.

Notes

When this structure is populated in the Vtk_CtxtGetOption call, the application should release the structure's contents using Vtk_CtxtOptionDeleteContent.

See Also

"Vtk_CtxtLogType" on page 69

"Vtk_Callback" on page 85

"Vtk_LogOptions" on page 99

"Vtk_ProxyInfo" on page 104

"Vtk_CtxtGetOption" on page 158

"Vtk_CtxtOptionDeleteContent" on page 162

"Vtk_CtxtSetOption" on page 166

Vtk_Extension

```
typedef struct {
   Vtk_Buffer oid;
   int critical;
   Vtk_Buffer value;
   void *vtkPrivateData;
}Vtk_Extension;
```

Description

This structure is used to represent an X.509 extension. Extensions can be present in certificates, CRLs, and the OCSP and CRT protocols. The data added depends on the type of extension.

This structure is returned by several of the extension functions and describes a specific extension within the list of extensions.

Parameters

oid	Object Identifier for the extension.
critical	Value that determines whether the extension is critical. The possible values are 0 and 1.
value	Buffer representing the value of the extension.
vtkPrivateData	Toolkit private data which is opaque to the application.

Notes

None

See Also

“Vtk_ErrorToString_r” on page 171

“Vtk_ExtensionGetByOid” on page 175

“Vtk_ExtensionsGetith” on page 185

Vtk_Extensions

```
typedef void* Vtk_Extensions;
```

Description

This structure contains the list of extensions that can be parsed using several of the extension functions. It represents X.509 extensions used in certificates, CRLs, and the OCSP and CRT protocols.

Parameters

None

Notes

If the application wants to return information about a specific extension with the list, it must obtain it from the Vtk_Extension structure.

See Also

“Vtk_Extension” on page 97

“Vtk_CertGetExtensions” on page 127

“Vtk_ExtensionGetByOid” on page 175

“Vtk_ExtensionsDelete” on page 181

“Vtk_ExtensionsGetCount” on page 183

“Vtk_ExtensionsGetith” on page 185

Vtk_LogOptions

```
typedef struct {
   enum Vtk_CtxtLogType logLevel;
   Vtk_uint32 int logMode;
   Vtk_Buffer logFileName;
   Vtk_OpenLogCallback openLogCB;
   Vtl_CloseLogCallback closeLogCB;
   Vtk_WriteLogCallback writeLogCB;
}Vtk_LogOptions;
```

Description

This structure defines the logging options. The parameters are set by the arguments of the function Vtk_OpenLog() in the Toolkit context.

Parameters

logLevel	An enumerated type, used here to specify the logging message level. Specifying a log level will also include all logging levels listed below it in the enumerated type Vtk_CtxtLogType definition. For example specify the log level of LOG_Debug to include debugging, info and error log messages. Specify log level of LOG_VaData to include all log messages.
logMode	Specifies the mode of logging. Modes can be combined using the bitwise OR (|) operator. The modes are: VTK_LOG_MODE_DEFAULT—New logs are appended to old log file, no flushing. VTK_LOG_MODE_OVERWRITE—New log messages overwrite any old data. VTK_LOG_MODE_FLUSH—Each message is flushed to the log file.
logFileName	The name and path of the Log File. If an empty buffer is passed, the default (vc_toolkit.log) is used.
openLogCB	A pointer to a user defined callback function that opens and initializes the log stream. If NULL is passed, the default toolkit function is used.
closeLogCB	A pointer to a user defined callback function that closes logging. If NULL is passed, the default toolkit function is used.

writeLogCB	A pointer to a user defined callback function that writes logging messages. If NULL is passed, the default toolkit function is used.

Notes

The callback pointers are used to indicate user-defined callback functions. Use these to customize Toolkit logging, for example to log to an application specific log facility.

Either provide all three logging callback functions or none.

See Also

“Data Structures” on page 81

“Vtk_OpenLogCallback” on page 121

“Vtk_WriteLogCallback” on page 123

“Vtk_CloseLogCallback” on page 115

“Vtk_OpenLog” on page 189

Vtk_OCSPSignInfo

```
typdef struct {
   Vtk_OCSPSignCallback OCSPSignCB;
   void *userHdl
   Vtk_Cert* signerCert;
}Vtk_OCSPSignInfo;
```

Description

This structure defines OCSP signing information for the context. This structure applies only if the Vtk_CtxtOptionType enumeration value is C0_OCSPSignInfo.

Specify this structure only if the application wants OCSP signed requests.

Parameters

OCSPSignCB	Function pointer to the OCSPSigncallback.
userHdl	Application specific handle that the Toolkit passes back to the application in the callback.
signerCert	Certificate that corresponds to the private key the application uses in the callback.

Notes

None

See Also

“Vtk_CtxtLogType” on page 69

“Vtk_OCSPSignCallBack” on page 119

Vtk_ProtocolDetails

```
typedef struct Vtk_ProtocolDetails_st
Vtk_ProtocolDetails;

typedef struct {
    enum Vtk_DataFormat encoding;
    enum Vtk_DataType type;
}Vtk_CRLProtocolDetails;

struct Vtk_ProtocolDetails_st
{
    enum Vtk_ValidationMech type
    union
    {
            Vtk_CRLProtocolDetails crl;
    } d;
```

Description

A structure that defines additional VA protocol information. Currently this structure is applicable to only CRL protocol information.

Parameters

type	Validation mechanism type employed by the user. The possible values are: ❖ VTK_VM_CRL For more information about this value, see “Vtk_ValidationMech” on page 80.
crl	CRL retrieval mechanism employed by the application. The information includes the data type and format applied to the data passed between the Toolkit and the application.

Notes

Currently, this structure is used only for CRLs.

See Also

“Vtk_ValQuery” on page 111

“Vtk_CtxtSetValnfo” on page 168

Vtk_ProxyInfo

```
typdef struct {
   char *host;
   int port;
} Vtk_ProxyInfo;
```

Description

This structure is used to specify information about the HTTP Proxy. This structure is used to specify the proxy host and port number used in the OCSP, CRT, and HTTP-CRL protocols.

Parameters

host	Name of the HTTP Proxy host.
port	Port number of the HTTP Proxy host.

When this structure is returned from the Vtk_CtxtGetOption function, you must use the Vtk_CtxtOptionDeleteContent function to release memory allocated to it.

See Also

“Vtk_CtxtLogType” on page 69

“Vtk_CtxtGetOption” on page 158

“Vtk_CtxtOptionDeleteContent” on page 162

Vtk_ValHdl

```
typedef void* Vtk_ValHdl;
```

Description

This data structure links validation information to specific certificate data and can be used to obtain validation status for a specific certificate. It is opaque to your application. The following validation functions allocate memory and resources for this structure:

- Vtk_ValidationAddCert
- Vtk_ValidationAddCertRaw
- Vtk_ValidationAddReqExtForSingleCertHdl

The Vtk_ValHdlGetRevStatus function uses this structure and the Vtk_ValHdlDelete function releases its memory and resources.

Parameters

None

Notes

None

See Also

“Vtk_ValidationAddCert” on page 198

“Vtk_ValidationAddCertRaw” on page 200

“Vtk_ValidationDelete” on page 210

“Vtk_ValidationGetRevStatus” on page 212

“Vtk_ValidationAddReqExtForSingleCertHdl” on page 208

Vtk_Validation

```
typedef void* Vtk_Validation;
```

Description

This data structure encapsulates a set of validation queries that can be sent to one or more VAs. The validation query structure is opaque to your application. It supports the OCSP, CRT, and CRL validation mechanisms. This data structure is used by the several validation and validation query functions.

Parameters

None

Notes

None

See Also

“Vtk_ValidationAddCert” on page 198

“Vtk_ValidationAddCertRaw” on page 200

“Vtk_ValidationDelete” on page 210

“Vtk_ValidationGetRevStatus” on page 212

“Vtk_ValidationGetQueries” on page 215

“Vtk_ValidationGetValHdl” on page 217

“Vtk_ValidationValidate” on page 223

“Vtk_ValidationValidateFromQueries” on page 225

Vtk_ValRespDetails

```
typdef struct {
   enum Vtk_ValidationMech type;
   Vtk_uint32 version;
   Vtk_Buffer *issuerIdByName;
   Vtk_Buffer *issueridByKey;
   time_t issueTime;
   Vtk_Extensions *extensions;
   union {
          Vtk_CRTRespDetails *crt;
          Vtk_CRLRespDetails *crl;
   } d;
} Vtk_ValRespDetails;
```

Description

This structure contains header information which details revocation information for the entire validation response. It includes header information common to all responses and header information specific to the OCSP, CRT, or CRL protocol employed. Additional protocol-specific information is available for the CRT and CRL protocols.

Parameters

type	Validation mechanism used to validate certificates.
version	Version number of the protocol
issuerIdByName	Issuer of validation information identified by name.This can be a VA or CA. It is NULL if a value is provided in the issuerIdByKey field.
issuerIdByKey	Issuer of validation information identified by public key hash. This can be a VA only. It is NULL if a value is provided in the issuerByName field.
issueTime	Time the response was issued.
extensions	Extensions included in the response.
crt	Additional CRT-specific information. See “Vtk_CRTRespDetails” on page 93
crl	Additional CRL-specific information. See “Vtk_CRLRespDetails” on page 92

Notes

This structure is used to return validation response information from the Vtk_CRLValidateCert, Vtk_ValidationGetRevStatus and Vtk_ValHdlGetRevStatus functions.

See Also

“Vtk_CRLRespDetails” on page 92

“Vtk_CRTRespDetails” on page 93

“Vtk_CRLValidateCert” on page 151

“Vtk_ValHdlGetRevStatus” on page 195

“Vtk_ValidationGetRevStatus” on page 212

Vtk_ValRespSingleCertDetails

```
typedef struct {
   enum Vtk_ValidationMech type;
   Vtk_uint32 certStatus;
   time_t thisUpdate;
   time_t nextUpdate;
   time_t revocationTime;
   enum Vtk_RevocationReason revocationReason;
   Vtk_Extensions *extensions;
}Vtk_ValRespSingleCertDetails;
```

Description

This structure contains detailed revocation information for a single certificate. A revocation response or CRL will contain this information for all the certificates they describe.

Parameters

type	Validation mechanism used to validate certificates.
certStatus	Status of the certificate.
thisUpdate	Time at which the status information was issued.
nextUpdate	Time at which the revocation data is expected to be updated again.
revocationTime	Time at which the certificate was revoked.
revocationReason	Reason the certificate was revoked. This field is optional. It is Vtk_Rev_Status_Unknown when not specified. See “Vtk_RevocationReason” on page 78 for a list of possible reasons.
extensions	Extensions included in the response.
crl	Additional CRL-specific information. See “Vtk_CRLRespDetails” on page 92

Notes

None

See Also

“Vtk_Extensions” on page 98

“Vtk_ValRespDetails” on page 107

“Vtk_CRLValidateCert” on page 151

“Vtk_ValHdlGetRevStatus” on page 195

“Vtk_ValidationGetRevStatus” on page 212

Vtk_ValQuery

```
typedef struct
{
    enum Vtk_ValidationMech type;
    char *host;
    int port;
    char *url;
    Vtk_ProtocolDetails *protocolDetails;
    Vtk_Buffer request;
    Vtk_Buffer response;
    void *vtkPrivateData;
} Vtk_ValQuery;
```

Description

This structure encapsulates a single validation query interaction with the VA. The query can be for more than one certificate, but it is specific to a single VA. An application uses this structure when it wants to communicate with the VA directly instead of using the Toolkit to communicate with the VA. An application may want to communicate directly when it wants perform asynchronous I/O or use SSL for the communication with the VA.

It supports, OCSP, CRT, and CRLs.

Parameters

type	Validation mechanism used to validate certificates.
host	Name of the VA host the application wants to communicate.
port	Port number of the VA host.
url	URL for VA, for example http://ci.valicert.net:80.
protocolDetails	Any additional protocol-specific information. Currently, addition protocol information is available only for CRLs.
request	Buffer structure that contains the request to be sent to the VA.

response	Buffer structure that contains the response sent from the VA.Once the operation is complete, the application needs to provides this structure to release it.
vtkPrivateData	Private data the Toolkit wants to include.

Notes

The actual communication between the application and the VA is done through the Vtk_ValidationValidateFromQueries. The application must create this structure from the Vtk_Validation structure using the Vtk_ValidationGetQueries.

See Also

“Vtk_ProtocolDetails” on page 102

“Vtk_Validation” on page 106

“Vtk_ValidationGetQueries” on page 215

“Vtk_ValidationValidateFromQueries” on page 225

“Vtk_ValQueriesDelete” on page 227

Callback Functions

The Toolkit provides callback functions to allow the application to return a value that directs the Toolkit about how to proceed. The callbacks are described separately in alphabetical order.

Vtk_ChainBuildCallBack

```
typedef int (VTK_CALLBACK*Vtk_ChainBuildCallback)
(
   void *userHdl,
   const Vtk_Cert *nextCert
);
```

Description

This callback function is used for certificate chain building. When an application calls the Vtk_ValidationAddCertChain function, the application can use this callback to provide a function pointer that will be called every time the Toolkit discovers a new link in the certificate chain.

Once the application completes its processing, it returns a 0 or 1 to the Toolkit. The values are as follows:

- VTK_OK—indicates that the Toolkit can add the certificate to the chain.
- 1—indicates that the Toolkit should not add the certificate and should search for other certificates.

Parameters

userHdl	Parameter that the Toolkit calls. The application provides this parameter.
nextCert	Certificate that is to be added to the chain

Notes

None

See Also

“Vtk_ValidationAddCertChain” on page 202

Vtk_CloseLogCallback

```
typedef void (VTK_CALLBACK *Vtk_CloseLogCallback)
(
    const Vtk_Ctxt *pCtxt,
    void* userHdl
);
```

Description

This callback function is used to close the log output stream.

When the Toolkit closes logging or the application calls Vtk_CloseLog, the Toolkit calls this callback to close all resources opened in Vtk_OpenLogCallback.

Parameters

pCtxt	A pointer to the Toolkit context.
userHdl	The value set in the Vtk_OpenLogCallback function, which specifies the application specific output structure.

Return Value

None

Notes

To override the default logging mechanism, use this function to close all open resources initialized by the Vtk_OpenLogCallback function.

To override the default logging mechanism, you must provide the Vtk_WriteLogCallback, Vtk_OpenLogCallback and Vtk_CloseLogCallback functions.

See Also

“Vtk_CloseLog” on page 150

“Vtk_OpenLogCallback” on page 121

“Vtk_WriteLogCallback” on page 123

Vtk_DelegatedIssuerCallBack

```
typedef int
(VTK_CALLBACK*Vtk_DelegatedIssuerCallback)
(
   void *userHdl,
   const Vtk_Cert *delegatedCert,
   const Vtk_Cert *trustedCert
);
```

Description

This callback function allows applications to examine and reject the certificate used to validate the validation response. It is used when the signer of the response is not directly part of the trusted VA or CA certificate store. The Toolkit calls back the application during its checking of OCSP and CRT responses in delegated trust models. It is only used in VA or CA delegated trust paradigms.

Once the application completes its processing, it returns a 0 or 1 to the Toolkit. The values are as follows:

- VTK_OK—indicates that the certificate is good and can be used for validating the response.
- 1—indicates the application has detected a problem and does not want the Toolkit to use the certificate to validate the response.

Parameters

userHdl	Parameter that the Toolkit calls. The application provides this parameter.
delegatedCert	Certificate that has been delegated and needs to be checked
trustedCert	CA or VA that issued the delegated certificate.

Notes

This callback is set through the Vtk_CtxtSetOption.

See Also

“Vtk_CtxtLogType” on page 69

“Vtk_Callback” on page 85

Vtk_OCSPSignCallBack

```
typedef int (VTK_CALLBACK*Vtk_OCSPSignCallback)
(
    void *userHdl,
    Vtk_Buffer *digest,
    Vtk_Buffer *sigOut,
    int maxSigOutBuf,
    int padding
);
```

Description

This callback is used for signing OCSP requests. It enables the application to add signatures to the outgoing requests. When the Toolkit gets to the point of its processing where OCSP signing occurs, this callback directs the Toolkit to call back the application. The application will then create the signature on the token.

Once the application completes its processing, it returns a 0 or 1 to the Toolkit. The values are as follows:

- VTK_OK—indicates that the OCSP signature has been added, the Toolkit can continue with its processing.
- 1—indicates an error was encountered when the application tried to add the signature, the Toolkit should not add the certificate.

Parameters

userHdl	Parameter that the Toolkit calls. The application provides this parameter.
digest	DER encoding of hashed data bytes.
sigout	Buffer that the Toolkit allocates for the application to store the resulting signature. Note: The application uses the length parameter to set the size of the signature written.
maxSigOutBuf	Size of the sigOut buffer allocated by the Toolkit.
padding	Type of padding requests. Use 1 to request PKCS#1 padding.

Notes

OCSP requests are not required to be signed.

This callback is set through the Vtk_CtxtSetOption.

See Also

“Vtk_CtxtLogType” on page 69

“Vtk_LogOptions” on page 99

Vtk_OpenLogCallback

```
typedef Vtk_uint32
(VTK_CALLBACK *Vtk_OpenLogCallback)
(
   const Vtk_Ctxt *pCtxt,
   void **userHdl
);
```

Description

This callback function is used to open the log output stream.

When the application calls Vtk_OpenLog to open logging the Toolkit uses this callback function to initialize the logging resource.

Parameters

pCtxt	A pointer to the Toolkit context.
userHdl	An application specific value, which is passed to the other logging callback functions. The Toolkit treats this value as opaque.

Return Value

VTK_OK	This indicates the logging output stream was opened correctly.
!VTK_OK	This indicates the application has detected a problem and does not want the Toolkit to use logging.

Notes

You can provide your own function to open an alternative store location like a database. In this case set a pointer to the structure defining an alternative logging resource.

The userHdl value is passed to the Vtk_CloseLogCallback and Vtk_WriteLogCallback functions.

You must provide all three callback functions to override the default Toolkit logging mechanism.

See Also

“Vtk_CloseLogCallback” on page 115

“Vtk_WriteLogCallback” on page 123

“Vtk_LogOptions” on page 99

“Vtk_OpenLog” on page 189

Vtk_WriteLogCallback

```
typedef void (VTK_CALLBACK *Vtk_WriteLogCallback)
(
    const Vtk_Ctxt *pCtxt,
    enum Vtk_CtxtLogType type,
    const char *pMsg,
    void* userHdl
);
```

Description

This callback function specifies how the application processes logging messages. When the Toolkit logs a message or the application calls Vtk_WriteLog, the Toolkit uses this callback function to process log messages.

The default Toolkit implementation is for the application to write log messages to the log file (according to the options specified in the structure definition Vtk_LogOptions).

Parameters

pCtxt	A pointer to the Toolkit context.
type	The type of log message (see the Vtk_CtxtLogType enumerated type).
pMsg	The log message.
userHdl	The value which specifies the output logging structure, set in the Vtk_OpenLogCallback function.

Return Value

None

Notes

To override the default logging mechanism, you must provide the Vtk_WriteLogCallback, Vtk_OpenLogCallback and Vtk_CloseLogCallback functions.

See Also

“Vtk_WriteLog” on page 233

“Vtk_LogOptions” on page 99

“Vtk_OpenLogCallback” on page 121

“Vtk_CloseLogCallback” on page 115

Functions

The functions are described separately in alphabetical order. Each description indicates which function category it belongs to. Table 3 lists and briefly describes the categories of functions and routines available in the Toolkit API.

Table 3. Function Categories

Category	Description
Certificate	Encapsulate X.509 certificates when interacting with the Toolkit.
Certificate Store	Group certificates in a single list.
Extension	Manipulate extension structure which encapsulates any X.509 extension. An X.509 extension includes certificate extensions as well as CRL, OCSP, and CRT protocol extensions.
General	Initialization and release functions called for allocating and releasing resources for the Toolkit.
Validation	Perform OCSP, CRT, or CRL validation checking for certificates. Encapsulate one or more validation queries for processing which can result in multiple queries to different VAs using different protocols, or limit the query to a single protocol or VA.
Context	Define the global Toolkit context and the certificate validation context
Validation Query	Allow application to perform communication with the VA to validate certificates instead of using the ToolKit to communicate with the VA.

Vtk_CertDelete

```
#include <vtk_cert.h>
#include <vtk_defs.h>

void Vtk_CertDelete(
   Vtk_Cert *cert     /* input */
);
```

Description

This certificate encapsulation function releases memory allocated for certificate data and its structure. Once this function completes successfully, the Vtk_Cert structure becomes invalid.

Parameters

cert	Certificate for which the memory is to be released.

Return Value

none	The function has completed successfully and the Vtk_Cert structure has been deleted.

Notes

The application must call this function for each certificate created using Vtk_CertNew and certificate returned using the Vtk_CertGetIssuer function. If the application does not call Vtk_CertDelete, memory leaks and other problems can occur. For more information about the memory model employed by the Toolkit, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_Cert” on page 87

“Vtk_CertGetIssuer” on page 131

“Vtk_CertNew” on page 139

Vtk_CertGetExtensions

```
#include <vtk_cert.h>
#include <vtk_defs.h>
#include <vtk_errs.h>

Vtk_uint32 Vtk_CertGetExtensions(
   const Vtk_Ctxt* ctxt, /* input */
   const Vtk_Cert *from, /* input */
   Vtk_Extensions **into /* output */
);
```

Description

This certificate encapsulation function returns a list of certificate extensions for a specified certificate in the specified context. This function allows you to directly get the list of extensions for the certificate without getting all of the certificate information contained in the Vtk_CertInfo structure.

The Toolkit provides several other functions that allow your application to parse the Vtk_Extensions structure and return the following:

- number of extensions in the list
- extension in the list based on its OID
- extension based on its position in the extension list

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
from	Pointer to the certificate for which the extensions are to be returned.
into	Pointer to structure into which the list of certificate extensions is to be placed.

Return Value

VTK_OK	The function has completed successfully and the list of certificate extensions has been placed into the structure.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The application must call the Vtk_ExtensionsDelete function for the structure returned by Vtk_CertGetExtensions. If the application does not call Vtk_ExtensionsDelete, memory leaks and other problems can occur. For more information about the memory model employed by the Toolkit, see “Toolkit Memory Model” on page 9.

In the case that the certificate has no extension, the function returns VTK_OK and sets the specified output structure to NULL.

See Also

“Vtk_CertInfo” on page 88

“Vtk_ExtensionGetByOid” on page 175

“Vtk_ExtensionsDelete” on page 181

“Vtk_ExtensionsGetCount” on page 183

“Vtk_ExtensionsGetith” on page 185

Vtk_CertGetInfo

```
#include <vtk_cert.h>
#include <vtk_err.h>

Vtk_uint32Vtk_CertGetInfo(
   const Vtk_Ctxt* ctxt, /* input */
   const Vtk_Cert *from, /* input */
   Vtk_CertInfo **dest   /* output */
);
```

Description

This certificate encapsulation function returns detailed certificate information for the specified certificate.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
from	Pointer to the certificate for which the information is to be returned.
dest	Pointer to structure into which the information is to be copied.

Return Value

VTK_OK	The function has completed successfully and the buffer has been deleted.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The application must call the Vtk_CertInfoDelete function to release the memory allocated to the structure returned by the Vtk_CertGetInfo function. If the application does not call Vtk_CertInfoDelete, memory leaks and other problems can occur. For more information about the memory model employed by the Toolkit, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_CertInfo” on page 88

“Vtk_CertInfoDelete” on page 133

Vtk_CertGetIssuer

```
#include <vtk_cert.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CertGetIssuer(
   const Vtk_Ctxt* ctxt,   /* input */
   const Vtk_Cert *cert,   /* input */
   Vtk_Cert **issuer   /* output */
);
```

Description

This certificate encapsulation function returns the certificate of the CA that has signed and issued the specified certificate. The function searches the certificates in the specified context.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
cert	Pointer to the certificate for which the issuer is to be returned.
issuer	Pointer to the issuer information extracted from the Vtk_Cert structure.

Return Value

VTK_OK	The function has completed successfully and the issuer information has been extracted and placed in Vtk_CertInfo.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The application must call the Vtk_CertDelete function for the structure returned by Vtk_CertGetIssuer. If the application does not call Vtk_CertDelete, memory leaks and other problems can occur. For more information about the memory model employed by the Toolkit, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_CertInfo” on page 88

“Vtk_CertDelete” on page 126

Vtk_CertInfoDelete

```
#include <vtk_cert.h>
#include <vtk_defs.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CertInfoDelete(
   Vtk_CertInfo *ci /* input */
);
```

Description

This certificate encapsulation function releases memory allocated to the Vtk_CertInfo structure and the detailed certificate information. Once this function completes successfully, the Vtk_CertInfo structure becomes invalid.

Parameters

ci	Pointer to certificate information structure for which the memory is to be released.

Return Value

VTK_OK	The function has completed successfully and the Vtk_CertInfo structure has been deleted.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The application must call this function for certificate information returned using the Vtk_CertGetInfo function. If the application does not call Vtk_CertInfoDelete, memory leaks and other problems can occur. For more information about the memory model employed by the Toolkit, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_CertInfo” on page 88

“Vtk_CertGetInfo” on page 129

Vtk_CertInit

```
#include <vtk_cert.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CertInit(
   const Vtk_Ctxt* ctxt,      /* input */
   Vtk_Cert *dest,            /* output */
   const Vtk_Buffer *source   /* input */
);
```

Description

This certificate encapsulation function initializes the Vtk_Cert data structure based on the data the application passes to it in the Vtk_Buffer structure.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
dest	Pointer to the structure after it has been initialized with the passed in data.
source	Data the application passes to the Toolkit. The data includes the encoding and decoding method employed. The possible values are: ❖ VTK_DF_DER ❖ VTK_DF_BASE64

Return Value

VTK_OK	The function has completed successfully and the certificate has been initialized with the data passed in.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

None

See Also

“Vtk_Buffer” on page 82

“Vtk_CertDelete” on page 126

“Vtk_CertLoadFromFile” on page 137

“Vtk_CertNew” on page 139

Vtk_CertLoadFromFile

```
#include <vtk_cert.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CertLoadFromFile(
   const Vtk_Ctxt* ctxt,        /* input */
   Vtk_Cert *into,              /* output */
   const char *fileName,        /* input */
   enum Vtk_DataFormat format /* input */
);
```

Description

This certificate encapsulation function initializes a Vtk_Cert data structure using the data in the file that the application passes to it.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
dest	Pointer to the newly initialized Vtk_Cert structure.
fileName	Name of file that contains the certificate data.
format	Encoding and decoding method employed. The possible values are: ❖ VTK_DF_DER ❖ VTK_DF_BASE64

Return Value

VTK_OK	The function has completed successfully and the certificate structure has been initialized from the named file.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

Encoding information is passed in the Vtk_DataFormat enumeration. Compare this to the Vtk_CertInit function which uses the Vtk_Buffer structure to pass encoding and certificate information.

See Also

“Vtk_CertInit” on page 135

“Vtk_CertDelete” on page 126

Vtk_CertNew

```
#include <vtk_cert.h>
#include <vtk_err.h>

Vtk_Cert* Vtk_CertNew(
   const Vtk_Ctxt *ctxt    /* input */
);
```

Description

This certificate encapsulation function allocates memory for the Vtk_Cert structure and creates an empty X.509 certificate structure.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.

Return Value

Vtk_Cert	The function has completed successfully. The function returns the Vtk_Cert structure that has been newly created and initialized.
NULL	The function has failed.

Notes

This function must be called before any other certificate encapsulation functions. Once the empty certificate structure is created, the application can populate the certificate with data using the Vtk_CertInit and Vtk_CertLoadFromFile. The application must call Vtk_CertDelete to release the memory allocated by this function when it no longer needs the structure.

See Also

“Vtk_Cert” on page 87

“Vtk_CertDelete” on page 126

“Vtk_CertGetIssuer” on page 131

“Vtk_CertInit” on page 135

“Vtk_CertLoadFromFile” on page 137

Vtk_CertStoreAddCert

```
#include <vtk_cert.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CertStoreAddCert(
   const Vtk_Ctxt *ctxt,     /* input */
   Vtk_CertStore *dest,       /* output */
   const Vtk_Cert *aCert      /* input */
);
```

Description

This certificate store function adds a single certificate to the Vtk_CertStore structure which was created using the Vtk_CertStoreNew function.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
dest	Pointer to certificate store to which this certificate is to be added. A certificate store is a container for one or more certificates.
aCert	Pointer to the certificate that is to be added to the store. This certificate is contained in an initialized structure.

Return Value

VTK_OK	The function has completed successfully and the certificate has been added to the certificate store.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

Vtk_Cert structure is used to encapsulate the X.509 certificates. Compare this to the VtkCertStoreAddRaw function which uses the Vtk_Buffer structure to pass encoding and certificate information.

To add several certificates, the application can call this function several times or more conveniently, the Vtk_CertStoreLoadFromFile to add several certificates at one time.

A certificate store can contain certificates that use different encoding formats. There is no practical limit on the number of certificates that a certificate store can contain.

The application is responsible for releasing the memory allocated to the certificate it has passed in to the function.

See Also

“Vtk_CertStoreAddCertRaw” on page 143

“Vtk_CertStoreLoadFromFile” on page 146

Vtk_CertStoreAddCertRaw

```
#include <vtk_cert.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CertStoreAddCertRaw(
   const Vtk_Ctxt *ctxt,         /* input */
   Vtk_CertStore *dest,          /* output */
   const Vtk_Buffer *certData    /* input */
);
```

Description

This certificate store function adds a single certificate to the Vtk_CertStore which was created using the Vtk_CertStoreNew function. The certificate that is to be added is not within an initialized structure; however, it is created from the user supplied data contained in the buffer.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
dest	Certificate store to which this certificate is to be added. A certificate store is a container for one or more certificates.
certdata	Certificate data that is to be added to the certificate store. It is not within an initialized certificate structure. Data can represent the certificate encoded using DER or Base64.

Return Value

VTK_OK	The function has completed successfully and the certificate has been added.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

Encoding and certificate information is passed in the Vtk_Buffer structure. Compare this to the VtkCertStoreAddCert function which uses the Vtk_Cert structure to encapsulate the X.509 certificate.

This function is provided as a convenience to allow an application to add a certificate that is not contained in an initialized structure.

See Also

“Vtk_Buffer” on page 82

“Vtk_Cert” on page 87

“Vtk_CertStoreAddCert” on page 141

“Vtk_CertStoreNew” on page 148

Vtk_CertStoreDelete

```
#include <vtk_cert.h>
#include <vtk_err.h>

void Vtk_CertStoreDelete(
   Vtk_CertStore *toFree     /* input */
);
```

Description

This certificate store function releases memory and resources previously allocated by the Vtk_CertStoreNew function. Once this function completes successfully, the Vtk_CertStore structure becomes invalid.

Parameters

toFree	Pointer to the Vtk_CertStore structure for which memory and resources are to be released

Return Value

none	The function has completed successfully and the certificate store has been deleted.

Notes

The application must call this function for each certificate store created using Vtk_CertStoreNew. If the application does not call Vtk_CertStoreDelete, memory leaks and other problems can occur. For more information about the Toolkit memory model, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_CertStoreNew” on page 148

Vtk_CertStoreLoadFromFile

```
#include <vtk_cert.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CertStoreLoadFromFile(
   const Vtk_Ctxt *ctxt,       /* input */
   Vtk_CertStore *dest,         /* output */
   const char *fileName,        /* input */
   enum Vtk_DataFormat format   /* input */
);
```

Description

This certificate store function initializes a Vtk_CertStore data structure using the data in the file that the application passes to it.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
dest	Pointer to the newly initialized Vtk_CertStore structure.
filename	File that contains the certificate data. The file can contain one or more certificates.
format	Encoding and decoding method employed. The possible values for this enumeration are: ❖ VTK_DF_DER ❖ VTK_DF_BASE64

Return Value

VTK_OK	The function has completed successfully and the certificate store structure has been initialized from the named file.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

Encoding information is passed in the Vtk_DataFormat enumeration. Compare this to the VtkCertStoreAddCert function which uses the Vtk_Cert structure to pass certificate information and the Vtk_CertStoreAddRaw function which uses the Vtk_Buffer structure to pass encoding and certificate information.

See Also

“Vtk_CertStoreAddCert” on page 141

“Vtk_CertStoreAddCertRaw” on page 143

Vtk_CertStoreNew

```
#include <vtk_cert.h>
#include <vtk_err.h>

Vtk_CertStore* Vtk_CertStoreNew(
   const Vtk_Ctxt *ctxt    /* input */
);
```

Description

This certificate store function creates and initializes an empty Vtk_CertStore structure which can be used to group and store VA or CA certificates. During the creation phase, this function allocates memory and resources for the structure. The certificate store is used during query validation and chain building operations.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.

Return Value

Vtk_CertStore	The function has completed successfully. The function returns the Vtk_CertStore structure that has been newly created and initialized.
NULL	The function has failed.

Notes

Once the certificate store structure is created, the application can add certificates individually (Vtk_CertStoreAddCert), as raw encoded data (Vtk_CertStoreAddRaw), or several at a time from a file (Vtk_CertStoreLoadFromFile).

The application must call the Vtk_CertStoreDelete function for each certificate store created using Vtk_CertStoreNew. If the application does not call Vtk_CertStoreDelete, memory leaks and other problems can occur. For more information about the Toolkit memory model, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_CertStoreDelete” on page 145

Vtk_CloseLog

```
include <vtk_error.h>
include <vtk_defs.h>

void Vtk_CloseLog(Vtk_Ctxt *pCtxt);
```

Description

Call this function to stop Toolkit logging.

This function calls the Vtk_CloseLogCallback function and removes the logging options from the context. If the user's callback function is not defined, all log messages are flushed to the log file and the file is closed.

Parameters

pCtxt	A pointer to the Toolkit context.

Return Value

None

Notes

None

See Also

"Vtk_CloseLogCallback" on page 115

"Vtk_OpenLog" on page 189

"Vtk_WriteLog" on page 233

Vtk_CRLValidateCert

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CRLValidateCert(
 const Vtk_Ctxt *ctxt,              /* input */
 const Vtk_Cert *cert,              /* input */
 Vtk_uint32 *status,                /* output */
 Vtk_ValRespDetails **respDetails,
                                    /* input/output */
 Vtk_ValRespSingleCertDetails *certDetails
                                    /* input/output */
);
```

Description

This validation function checks whether the specified certificate is on the CRL and returns a status information about the certificate. The application can optionally specify to return detailed revocation information for the entire validation response or the single certificate.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
cert	Pointer to the certificate to check for in the CRL.
status	Pointer to the status information for the certificate. The status values are defined as a bit field. Therefore, a single status value can represent multiple status codes. For a list of the possible certificate status codes, see Appendix A, "Error and Status Codes."
respDetails	Address to the pointer to the detailed revocation information for the entire response. (It points to a structure allocated by the Toolkit in this call.) This function returns the requested information in the structure. In cases where the result is not needed, the application can pass in NULL. The application must release this structure using the Vtk_ValRespDetailsDelete function.

certDetails	Address to the pointer to the detailed revocation information for a single certificate response. (It points to a structure allocated by the Toolkit in this call.) This function returns the requested information in the structure. In cases where the result is not needed, the application can pass in NULL. The application must release this structure using the Vtk_ValRespSingleCertDetailsDelete function.

Return Value

VTK_OK	The function has completed successfully and if specified, returns the Vtk_ValRespDetails or Vtk_ValRespSingleCertDetail structure.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The function uses the configuration data in the context to obtain the CRL.

Unlike most validation functions, this function does not require the Vtk_Validation structure.

If the application specifies Vtk_RespDetails or Vtk_ValRespSingleCertDetails, but the VA does not have any information for the specified certificate, the function returns successfully (VTK_OK) and sets the specified structure to NULL.

The application must release these structures using the Vtk_ValRespDetailsDelete or Vtk_ValRespSingleCertDetailsDelete function.

See Also

“Vtk_ValRespDetails” on page 107

“Vtk_ValRespSingleCertDetails” on page 109

“Vtk_ValRespDetailsDelete” on page 229

“Vtk_ValRespSingleCertDetailsDelete” on page 231

Vtk_CtxtAddCert

```
#include <vtk_ctxt.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CtxtAddCert(
   Vtk_Ctxt *ctxt,          /* input */
   int type,                /* input */
   const Vtk_Cert *cert     /* input */
);
```

Description

This Toolkit context function adds an individual certificate to the list of trusted certificates maintained by the Toolkit in the context structure. The application can add a VA or CA certificate to the list, depending on the specified type.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated. The certificate is to be added to this structure.
type	Type of certificate to add to the trusted certificate list. The possible values are: ❖ VTK_VA_CERT for a trusted VA certificate ❖ VTK_TRUSTED_CA_CERT for trusted CA certificate ❖ VTK_INTERMEDIATE_CERT for intermediate CA certificate These bit fields can be combined.
cert	Certificate to add to the trusted list.

Return Value

VTK_OK	The function has completed successfully and the certificate has been added.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The application must delete the certificate using the Vtk_CertDelete function.

See Also

“Vtk_CtxtAddCerts” on page 155

“Vtk_CtxtNew” on page 160

Vtk_CtxtAddCerts

```
#include <vtk_ctxt.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CtxtAddCerts(
   Vtk_Ctxt* ctxt,              /* input */
   int type,                    /* input */
   const Vtk_CertStore *certs   /* input */
);
```

Description

This Toolkit context function adds an one or more certificates to the list of trusted certificates maintained by the Toolkit in the context structure. The application can add a VA, CA or both types of certificates to the list.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated. The certificate is to be added to this structure.
type	Type of certificate to add to the trusted certificate list. The possible values are: ❖ VTK_VA_CERT for a trusted VA certificate ❖ VTK_TRUSTED_CA_CERT for trusted CA certificate ❖ VTK_INTERMEDIATE_CERT for intermediate CA certificate These bit fields can be combined.
certs	Certificate store (list of certificates) to add to the trusted list.

Return Value

VTK_OK	The function has completed successfully and the buffer has been deleted.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

None

See Also

“Vtk_CtxtAddCert” on page 153

“Vtk_CtxtNew” on page 160

Vtk_CtxtDelete

```
#include <vtk_ctxt.h>
#include <vtk_err.h>

void Vtk_CtxtDelete(
   Vtk_Ctxt *toFree     /* input */
);
```

Description

This Toolkit context function releases memory and resources previously allocated by the Vtk_CtxtNew function. Once this function completes successfully, the Vtk_Ctxt structure becomes invalid.

Parameters

toFree	Pointer to the Vtk_Ctxt structure for which memory and resources are to be released.

Return Value

none	The function has completed successfully and the validation context has been deleted.

Notes

The application must call this function for each context created using Vtk_CtxtNew. If the application does not call Vtk_CtxtDelete, memory leaks and other problems can occur. For more information about the Toolkit memory model, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_CtxtNew” on page 160

Vtk_CtxtGetOption

```
#include <vtk_ctxt.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CtxtGetOption(
   const Vtk_Ctxt* ctxt,     /* input */
   Vtk_CtxtOption *outData   /* output */
);
```

Description

This Toolkit context function returns current context option information for the specified context. The size of the structure varies with the type of information currently set for the context.

For a comprehensive list of the supported context option types, see “Vtk_CtxtLogType” on page 69.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated. The structure for which this function gets the option information.
outData	Option information returned. The information depends on the option type currently set.

Return Value

VTK_OK	The function has completed successfully and the Vtk_CtxtOption structure with the current option information has been returned to the application.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The application must release the returned Vtk_CtxtOption structure using the Vtk_CtxtOptionDeleteContent function.

See Also

“Vtk_CtxtLogType” on page 69

“Vtk_CtxtSetOption” on page 166

Vtk_CtxtNew

```
#include <vtk_ctxt.h>
#include <vtk_err.h>

Vtk_Ctxt* Vtk_CtxtNew(void);
```

Description

This Toolkit context function creates, initializes, and allocates memory for a validation context structure. This structure contains global information that is persistent over validation checks such as the VA URL for a specific CA and trusted certificates for VAs and CAs.

By default the context uses the Global VA Service as its default VA and CRT as the default protocol. The context contains pre-loaded ValiCert Global VA Service certificates and its URL defined by the Vtk_GVAS_URL constant.

Parameters

None

Return Value

Vtk_Ctxt	The function has completed successfully. The function returns the Vtk_Ctxt structure that has been newly created and initialized.
NULL	The function has failed.

Notes

All Toolkit calls require a Vtk_Ctxt parameter. Before calling this function, or any Toolkit function, the application must first call the Vtk_Init function.

See Also

“Vtk_CtxtAddCert” on page 153

“Vtk_CtxtAddCerts” on page 155

“Vtk_CtxtDelete” on page 157

“Vtk_Init” on page 188

Vtk_CtxtOptionDeleteContent

```
#include <vtk_ctxt.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CtxtOptionDeleteContent(
   Vtk_CtxtOption *data
);
```

Description

This Toolkit context function releases memory previously allocated to the Vtk_CtxtOption structure with the Vtk_CtxtGetOption function. Once this function completes successfully, the Vtk_CtxtOption structure becomes invalid. Other resources allocated to the structure are not released.

Parameters

data	Pointer to the Vtk_CtxtOption structure for which memory and resources are to be released.

Return Value

VTK_OK	The function has completed successfully and the context option structure has been deleted.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

Only the context option data contained within the structure is released. The application must call this function for each context option structure returned from the Vtk_CtxtGetOption function. If the application does not call Vtk_CtxtOptionDeleteContent, memory leaks and other problems can occur. For more information about the Toolkit memory model, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_CtxtGetOption” on page 158

Vtk_CtxtSetDefaultVa

```
#include <vtk_ctxt.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CtxtSetDefaultVa(
   Vtk_Ctxt* ctxt,                  /* input */
   const char *vaUrl,               /* input */
   enum Vtk_ValidationMech mech     /* input */
);
```

Description

This Toolkit context function sets the default VA for the context passed in. The VA URL and validation mechanism specified in this function will be used as the default for all validations performed without an explicit VA.

When a context is newly created using the Vtk_CtxtNew function, it sets the default VA to the Global VA Service and CRT as the default protocol. The default URL for the Global VA Service is specified in the Vtk_GVAS_URL constant.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated. It is the Toolkit context for which this function sets the default VA.
vaUrl	URL for default VA, for example http://ocsp.valicert.net:80.
mech	Validation mechanism employed to validate certificates. The possible values are: ❖ VTK_VM_CRT—Certificate Revocation Trees ❖ VTK_VM_OCSP—Online Certificate Status Protocol

Return Value

VTK_OK	The function has completed successfully and the default VA has been set.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The same default VA can have different protocols for different contexts.

See Also

“Vtk_CtxtNew” on page 160

Vtk_CtxtSetOption

```
#include <vtk_ctxt.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CtxtSetOption(
   Vtk_Ctxt* ctxt,        /* input */
   Vtk_CtxtOption *data   /* input */
);
```

Description

This Toolkit context function allows the application to set a specific option for the specified context. The application sets the option through the Vtk_CtxtOption structure which identifies the option type. For a comprehensive list of the supported context option types, see “Vtk_CtxtLogType” on page 69.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated. The Toolkit context for which this function sets the option.
data	Option information to be set. The information depends on the option type.

Return Value

VTK_OK	The function has completed successfully and the option has been set.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

None

See Also

“Vtk_CtxtLogType” on page 69

“Vtk_CtxtGetOption” on page 158

Vtk_CtxtSetVaInfo

```
#include <vtk_ctxt.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_CtxtSetVaInfo(
  Vtk_Ctxt* ctxt,                   /* input */
  const Vtk_Cert *caCert,           /* input */
  const char *vaUrl,                /* input */
  enum Vtk_ValidationMech mech,     /* input */
  const Vtk_ProtocolDetails *protDetails,
                                    /* input */
  const Vtk_CertStore *vaCerts      /* input */
);
```

Description

This Toolkit context function sets validation information about the VA that is validating certificates issued by a specific CA. The VA validation information includes the VA URL, validation mechanism and optionally, the list of trusted VA certificates to be used with this VA or CA.

The VA URL and validation mechanism specified in this function will be used as the default for all validations performed for certificates issued by this CA. If you specify CRL as the validation mechanism, you must also set details about the protocol in the Vtk_ProtocolDetails structure.

You are not required to set the list of trusted certificates using this function. However, if you do not, the VA certificates set through the Vtk_CtxtAddCert and Vtk_CtxtAddCerts calls are used.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated. The Toolkit context for which this function sets VA information.
caCert	CA for which this validation information should be set.
vaUrl	URL for default VA, for example http://ocsp.valicert.net:80.

mech	Validation mechanism employed to validate certificates. The possible values are: ❖ VTK_VM_CRT—Certificate Revocation Tree ❖ VTK_VM_OCSP—Online Certificate Status Protocol ❖ VTK_VM_CRL—Certificate Revocation List
protDetails	Any additional CRL protocol-specific information.
vaCerts	List of trusted VA certificates. This is optional. If not specified, the list of trusted VA certificates set through the Vtk_CtxtAddCert and Vtk_CtxtAddCerts is used.

Return Value

VTK_OK	The function has completed successfully and the VA information is set.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

None

See Also

“Vtk_CRLProtocolDetails” on page 91

“Vtk_ProtocolDetails” on page 102

“Vtk_CtxtAddCert” on page 153

“Vtk_CtxtAddCerts” on page 155

Vtk_ErrorToString

```
#include <vtk_errs.h>

const char *Vtk_ErrorToString(
   Vtk_uint32 errorCode     /* input */
);
```

Description

This return code translation routine returns a static string representation of the specified return code.

Parameters

errorCode	Error code to be translated. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Return Value

char *	The function has completed successfully and the error code has been translated. The corresponding symbol for this value is VTK_OK.
NULL	The function has failed.

Notes

The return value must not be deleted.

This function is not MT-Safe.

See Also

“Vtk_ErrorToString_r” on page 171

“Vtk_StatusToStrings” on page 191

Vtk_ErrorToString_r

```
#include <vtk_errs.h>

const char *Vtk_ErrorToString_r(
   Vtk_uint32 errorCode,  /* input /*
   char *buf,             /* input/output /*
   int buflen             /* input /*
);
```

Description

This function converts the specified Toolkit error code to a string. The caller must supply a buffer `buf` of length `buflen` to store the result. Toolkit copies the error string to the buffer. If the buffer is too small the string is truncated. A buffer of 128 bytes should be sufficient.

Parameters

errorCode	The error code to be translated. For information about possible error values, refer to Appendix A, "Error and Status Codes."
buf	The buffer into which the function copies the error.
buflen	The length of the buffer.

Return Value

char *	The supplied buf pointer.
NULL	The function has failed due to an invalid error code.

Notes

The return value must not be deleted.

This is an MT-Safe version of `Vtk_ErrorToString`.

See Also

“Vtk_ErrorToString” on page 170

“Vtk_StatusToStrings” on page 191

Vtk_ExtensionDelete

```
#include <vtk_cert.h>
#include <vtk_defs.h>
#include <vtk_errs.h>

void Vtk_ExtensionDelete(
   Vtk_Extension *ext     /* input */
);
```

Description

This extension function deletes an extension structure that contains the search results from the Vtk_ExtensionGetByOid or Vtk_ExtensionGetith functions.

Once this function completes successfully, the Vtk_Extension structure becomes invalid.

Parameters

ext	Extension structure for which the memory is to be released.

Return Value

none	The function has completed successfully and the Vtk_Extension structure has been deleted.

Notes

The structure deleted by this function is different from the Vtk_Extensions structure returned by the Vtk_CertGetExtensions function

The application must call this function for the extension structure returned using the Vtk_ExtensionGetByOid or Vtk_ExtensionGetith function. If the application does not call Vtk_ExtensionDelete, memory leaks and other problems can occur. For more information about the memory model employed by the Toolkit, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_Extension” on page 97

“Vtk_ExtensionGetByOid” on page 175

“Vtk_ExtensionInit” on page 177

“Vtk_ExtensionNew” on page 179

“Vtk_ExtensionsGetith” on page 185

Vtk_ExtensionGetByOid

```
#include <vtk_cert.h>
#include <vtk_defs.h>
#include <vtk_errs.h>

Vtk_uint32 Vtk_ExtensionGetByOid(
   const Vtk_Ctxt *ctxt,       /* input */
   const Vtk_Extensions *exts, /* input */
   const Vtk_Buffer *oid,      /* input */
   Vtk_Extension **dest        /* output */
);
```

Description

This extension function can be used to search for a specific Object Identifier (OID) in the list of extensions currently in Vtk_Extensions structure. This search function can be used to search a list of any type of extensions, that is certificate, OCSP, CRT, or CRL extensions.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
exts	Pointer to extensions structure that is to be parsed and for which memory has been allocated. It can be created using the Vtk_CertGetExtensions function or through the Vtk_ValRespDetails or Vtk_ValRespSingleCertDetails structure.
oid	Pointer to the OID to search for within the Vtk_Extensions structure. The OID can be specified in dot notation.
dest	Pointer to structure into which the extensions meeting the search criteria are to be placed.

Return Value

VTK_OK	The function has completed successfully and the Vtk_Extension structure with the specified OID has been returned.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The application must call Vtk_ExtensionDelete when finished with the returned structure, otherwise memory leaks and other problems can occur. For more information about the memory model employed by the Toolkit, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_ValRespDetails” on page 107

“Vtk_ValRespSingleCertDetails” on page 109

“Vtk_CertGetExtensions” on page 127

“Vtk_ErrorToString_r” on page 171

Vtk_ExtensionInit

```
#include <vtk_cert.h>
#include <vtk_defs.h>
#include <vtk_errs.h>

Vtk_uint32 Vtk_ExtensionInit(
   const Vtk_Ctxt *ctxt,    /* input */
   Vtk_Extension *ext,      /* output */
   const Vtk_Buffer *oid,   /* input */
   int critical,           /* input */
   const Vtk_Buffer *value /* input */
);
```

Description

This extension function initializes a Vtk_Extension data structure based on the data the application passes to it.

Parameters

ctxt	Pointer to the Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
ext	Pointer to the structure created using the Vtk_ExtensionNew function.
oid	Pointer to the object identifier for this extension. The Vtk_DataFormat type is VTK_DF_STRING, a null-terminated printable string in dot format, that is 1.2.3.4
critical	Value that determines whether the extension is critical. The possible values are 0 and nonzero. If the value is 0, the extension is not critical. If the value is nonzero, the extension is critical.
value	Pointer to the buffer that contains the data to be placed in the extension structure. The content of this buffer will be encoded as an ASN1 octet string for the extension.

Return Value

VTK_OK	The function has completed successfully and the Vtk_Extension structure has been initialized with the data passed in.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

None

See Also

“Vtk_Extension” on page 97

“Vtk_ErrorToString_r” on page 171

“Vtk_ExtensionNew” on page 179

Vtk_ExtensionNew

```
#include <vtk_cert.h>
#include <vtk_defs.h>
#include <vtk_errs.h>

Vtk_Extension* Vtk_ExtensionNew(
   const Vtk_Ctxt *ctxt     /* input */
);
```

Description

This extension function allocates memory for the Vtk_Extension structure and creates an empty extension structure.

Parameters

ctxt	Pointer to the Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.

Return Value

Vtk_Extension	The function has completed successfully. The function returns the Vtk_Extension structure that has been newly created and initialized.
NULL	The function has failed.

Notes

This function must be called before initializing the extension data with application information. Once the extension structure is created, the application can populate the extension with data using the Vtk_ExtensionInit. The application must call Vtk_ExtensionDelete to release the memory allocated by this function when it no longer needs the structure.

See Also

“Vtk_Extension” on page 97

“Vtk_ErrorToString_r” on page 171

“Vtk_ExtensionInit” on page 177

Vtk_ExtensionsDelete

```
#include <vtk_cert.h>
#include <vtk_defs.h>
#include <vtk_errs.h>

void Vtk_ExtensionsDelete(
   Vtk_Extensions *ext     /* input */
);
```

Description

This extensions function releases memory allocated for an extensions structure created by the Vtk_CertGetExtensions function.

Once this function completes successfully, the Vtk_Extensions structure becomes invalid.

Parameters

ext	Extensions structure for which the memory is to be released.

Return Value

none	The function has completed successfully and the Vtk_Extensions structure has been deleted.

Notes

The application must call this function for the extensions structure returned using the Vtk_CertGetExtensions function. If the application does not call Vtk_ExtensionsDelete to release memory allocated to this structure, memory leaks and other problems can occur. For more information about the memory model employed by the Toolkit, see “Toolkit Memory Model” on page 9.

The Toolkit also provides the Vtk_ExtensionDelete function for deleting an individual Vtk_Extension structure returned as part of the various extension parsing functions.

See Also

“Vtk_Extensions” on page 98

“Vtk_ValRespDetails” on page 107

“Vtk_ValRespSingleCertDetails” on page 109

“Vtk_CertGetExtensions” on page 127

“Vtk_ErrorToString_r” on page 171

Vtk_ExtensionsGetCount

```
#include <vtk_cert.h>
#include <vtk_defs.h>
#include <vtk_errs.h>

int Vtk_ExtensionsGetCount(
   const Vtk_Ctxt *ctxt,        /* input */
   const Vtk_Extensions *exts   /* input */
);
```

Description

This extension function can be used to determine the number of extensions currently in the Vtk_Extensions structure for the specified context. This function can be used to search a list of any type of extensions, that is certificate, OCSP, CRT, or CRL extensions.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
exts	Pointer to extensions structure that is to be parsed and for which memory has been allocated. It can be created using the Vtk_CertGetExtensions function or through the Vtk_ValRespDetails or Vtk_ValRespSingleCertDetails structure.

Return Value

val	The function has completed successfully and this is the number of extensions within the Vtk_Extensions structure.
0	No extensions are present.

Notes

None

See Also

“Vtk_ValRespDetails” on page 107

“Vtk_ValRespSingleCertDetails” on page 109

“Vtk_CertGetExtensions” on page 127

Vtk_ExtensionsGetith

```
#include <vtk_cert.h>
#include <vtk_defs.h>
#include <vtk_errs.h>

Vtk_uint32 Vtk_ExtensionsGetith(
   const Vtk_Ctxt *ctxt,         /* input */
   const Vtk_Extensions *exts,   /* input */
   int i,                        /* input */
   Vtk_Extension **dest          /* output */
);
```

Description

This extension function can be used to search for a specific occurrence of an extension within the list of extensions currently in Vtk_Extensions structure. This search function can be used to search a list of any type of extensions, that is certificate, OCSP, CRT, or CRL extensions.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
exts	Pointer to extensions structure that is to be parsed and for which memory has been allocated. It can be created using the Vtk_CertGetExtensions function or through the Vtk_ValRespDetails or Vtk_ValRespSingleCertDetails structure
i	Integer index into the extensions list. The index is 0 based. The application can call the Vtk_ExtensionsGetCount function to determine the total number of extensions.
dest	Pointer to structure into which the extensions meeting the search criteria is to be placed.

Return Value

VTK_OK	The function has completed successfully and the structure has been returned.

error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The application must call Vtk_ExtensionDelete when finished with the returned structure, otherwise memory leaks and other problems can occur. For more information about the memory model employed by the Toolkit, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_ValRespDetails” on page 107

“Vtk_ValRespSingleCertDetails” on page 109

“Vtk_CertGetExtensions” on page 127

“Vtk_ErrorToString_r” on page 171

“Vtk_ExtensionsGetCount” on page 183

Vtk_Finish

```
#include <vtk_ctxt.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_Finish(void);
```

Description

This general purpose function releases resources that have been allocated to the Toolkit. The application must call this function last after completing its work with the Toolkit.

Parameters

None

Return Value

VTK_OK	The function has completed successfully and the Toolkit resources have been released.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

This function must be called only once.

See Also

“Vtk_Init” on page 188

Vtk_Init

```
#include <vtk_ctxt.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_Init(void);
```

Description

This general purpose function initializes Toolkit and its communication library. The application must call this function before using the Toolkit.

Parameters

None

Return Value

VTK_OK	The function has completed successfully and the Toolkit and its communication library have been initialized.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

This function must be called only once.

See Also

“Vtk_Finish” on page 187

Vtk_OpenLog

```
include <vtk_error.h>
include <vtk_defs.h>

Vtk_uint32 Vtk_OpenLog(
   Vtk_Ctxt *pCtxt,     /* input */
   const Vtk_LogOptions *pLogOptions
                        /* input */
);
```

Description

This function initializes and commences logging for the specified Toolkit context.

If no user defined callback was specified, a text log file is opened with the name and mode set in log options.

Parameters

pCtxt	A pointer to the Toolkit context.
pLogOptions	Log options (see the data structure Vtk_LogOptions for details).

Return Value

VTK_OK	The function has completed successfully.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

None

See Also

“Vtk_LogOptions” on page 99

“Vtk_OpenLogCallback” on page 121

“Vtk_CloseLog” on page 150

“Vtk_WriteLog” on page 233

Vtk_StatusToStrings

```
#include <vtk_errs.h>

const char **Vtk_StatusToString(
   Vtk_uint32 status     /* input/output */
);
```

Description

This translation routine returns a null-terminated array of strings containing text associated with every value contained in the status value. The end of the array is indicated by a NULL entry.

Parameters

status	Status value to be translated. The status value is defined as a bit field which can represent multiple status codes.

Return Value

char **	The function has completed successfully and the status values have been translated into the requisite number of strings, one for each status code. The corresponding symbol for this value is VTK_OK.
NULL	The function has failed.

Notes

The application must call the Vtk_StatusStringsDelete function for each array created.

See Also

“Vtk_ErrorToString” on page 170

“Vtk_StatusStringsDelete” on page 192

Vtk_StatusStringsDelete

```
#include <vtk_err.h>

void Vtk_StatusStringsDelete(
   char** statusStrings     /* input */
);
```

Description

This translation function releases memory and resources previously allocated by the Vtk_StatusToStrings function. Once this function completes successfully, the resources are released.

Parameters

char	Pointer to the status strings that are to be deleted.

Return Value

none	The function has completed successfully and the statusStrings has been deleted.

Notes

The application must call this function for each status string created using Vtk_StatusToStrings. If the application does not call Vtk_StatusToStringsDelete, memory leaks and other problems can occur. For more information about the Toolkit memory model, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_StatusToStrings” on page 191

Vtk_ValHdlDelete

```
#include <vtk_valid.h>
#include <vtk_err.h>

void Vtk_ValHdlDelete(
   Vtk_ValHdl *hdl    /* input */
);
```

Description

This validation function releases memory previously allocated to this data structure with the Vtk_ValidationAddCert or Vtk_ValAddCertRaw function. Once this function completes successfully, the Vtk_ValHdl structure becomes invalid.

Parameters

hdl	Pointer to the auxiliary data structure for which memory and resources are to be released

Return Value

none	The function has completed successfully and the validation query structure has been deleted.

Notes

The auxiliary data structure is used to link individual certificate validation requests to their detail. The entire structure is released. The application must call this function for each validation handle structure created. If the application does not call Vtk_ValHdlDelete, memory leaks and other problems can occur. For more information about the Toolkit memory model, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_ValHdlGetRevStatus” on page 195

“Vtk_ValidationAddCert” on page 198

“Vtk_ValidationAddCertRaw” on page 200

Vtk_ValHdlGetRevStatus

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_ValHdlGetRevStatus(
   const Vtk_Ctxt *ctxt,             /* input */
   const Vtk_ValHdl *hdl,            /* input */
   Vtk_uint32 *status,               /* output */
   Vtk_ValRespDetails **respDetails,
                                  /* input/output */
   Vtk_ValRespSingleCertDetails **certDetails
                                  /* input/output */
   );
```

Description

This validation function retrieves validation status information for a single certificate. The application can specify to return detailed revocation information for the entire validation response or a single certificate. For information about the detailed information see Vtk_ValRespDetails or Vtk_ValRespSingleCertDetails.

This function is similar to the Vtk_ValidationGetRevStatus function. However, it is more optimized. Instead of checking the entire validation structure, this function uses the validation handle that has been specified for the certificate when the certificate was added to the validation query.

This function only applies to those certificates for which the application has specified a validation handle using Vtk_ValidationAddCert or Vtk_ValidationAddCertRaw functions.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
hdl	Pointer to a auxiliary data structure used to link individual certificate validation requests to their detail.

status	Pointer to the status information for the certificate. The status values are defined as a bit field. Therefore, a single status value can represent multiple status codes. For a list of the possible certificate status codes, see Appendix A, "Error and Status Codes."
respDetails	Address to the pointer to the detailed revocation information for the entire response. (It points to a structure allocated by the Toolkit in this call.) This function returns the requested information in the structure. In cases where the result is not needed, the application can pass in NULL. The application must release this structure using the Vtk_ValRespDetailsDelete function.
certDetails	Address to the pointer to the detailed revocation information for a single certificate response. (It points to a structure allocated by the Toolkit in this call.) This function returns the requested information in the structure. In cases where the result is not needed, the application can pass in NULL. The application must release this structure using the Vtk_ValRespSingleCertDetailsDelete function.

Return Value

VTK_OK	The function has completed successfully and if specified, returns the Vtk_ValRespDetails or Vtk_ValRespSingleCertDetail structure.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

If the application specifies Vtk_RespDetails or Vtk_ValRespSingleCertDetails, but the VA does not have any information for the specified certificate, the function returns successfully and sets the specified structure to NULL.

The application must release these structures using the Vtk_ValRespDetailsDelete or Vtk_ValRespSingleCertDetailsDelete function.

See Also

“Vtk_ValRespDetails” on page 107

“Vtk_ValRespSingleCertDetails” on page 109

“Vtk_ValRespDetailsDelete” on page 229

“Vtk_ValRespSingleCertDetailsDelete” on page 231

Vtk_ValidationAddCert

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_ValidationAddCert(
   const Vtk_Ctxt *ctxt,       /* input */
   Vtk_Validation *val,        /* input/output */
   const Vtk_Cert *cert,       /* input */
   const Vtk_Cert *issuerCert, /* input */
   Vtk_ValHdl **hdl            /* output */
);
```

Description

This validation function adds a single certificate to the validation query (Vtk_Validation structure created using the Vtk_ValidationNew function) that will be sent to the VA or Global VA Service for validation. To add the certificate to the validation query, the application must specify the end-user certificate and its issuer certificate. If the application wants to later set protocol specific extensions for this certificate validation or obtain validation status for this certificate, the application can use the optional Vtk_ValHdl structure.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
val	Pointer to the validation query structure to which the certificate is to be added.
cert	Pointer to the certificate to add to the validation query structure for validation.
issuerCert	Pointer to issuer certificate for the certificate to be validated.
hdl	Pointer to a auxiliary data structure used to link individual certificate validation requests to their detail.This is an optional return value.

Return Value

VTK_OK	The function has completed successfully and the certificate has been added to the validation query.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

If the application allocates memory to the Vtk_ValHdl structure, it can set extensions, obtain validation status, and obtain details specific to a certificate instead of as an aggregate value of all the certificates. However, since memory is allocated to this auxiliary structure, the application must call the Vtk_ValHdlDelete function to release it when the structure is no longer needed.

Alternatively, the application can use the Vtk_ValidationGetValHdl function to create the validation handle after the certificate is added.

See Also

“Vtk_ValHdlGetRevStatus” on page 195

“Vtk_ValidationAddCertRaw” on page 200

“Vtk_ValidationGetRevStatus” on page 212

“Vtk_ValidationGetValHdl” on page 217

“Vtk_ValidationGetValHdl” on page 217

“Vtk_ValidationValidate” on page 223

Vtk_ValidationAddCertRaw

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_ValidationAddCertRaw(
   const Vtk_Ctxt *ctxt,        /* input */
   Vtk_Validation *val,        /* input/output */
   const Vtk_Buffer *cert,      /* input */
   const Vtk_Buffer *issuerCert, /* input */
   Vtk_ValHdl **hdl             /* output */
);
```

Description

This validation function adds a single certificate to the validation query (Vtk_Validation structure created using the Vtk_ValidationNew function) that will be sent to the VA or Global VA Service for validation. The certificate that is to be added is created from the user supplied data contained in the buffer. To add the certificate to the validation query, the application must specify the end-user certificate and its issuer certificate. The certificates can be specified in raw DER/BASE64 format. If the application wants to later set protocol specific extensions for the this certificate validation query or obtain validation status for this certificate, the application must also use the optional Vtk_ValHdl structure.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
val	Pointer to the validation query structure to which the certificate is to be added.
cert	Pointer to the certificate to add to the validation query structure for validation.
issuerCert	Pointer to issuer certificate for the certificate to be validated.
hdl	Pointer to a auxiliary data structure used to link individual certificate validation requests to their detail.This is an optional return value.

Return Value

VTK_OK	The function has completed successfully and the certificate has been added to the validation query.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

If the application specifies the Vtk_ValHdl structure, it can set extensions, obtain validation status, and obtain details specific to a certificate instead of as an aggregate value of all the certificates. However, since memory is allocated to this auxiliary structure, the application must call the Vtk_ValHdlDelete function to release it when the structure is no longer needed.

If the application does not use the optional Vtk_ValHdl structure, that is, it passes NULL as the parameter when adding the certificate, the application can use the Vtk_ValidationGetValHdl function to create the validation handle after the certificate is added.

See Also

“Vtk_ValHdlGetRevStatus” on page 195

“Vtk_ValidationAddCert” on page 198

“Vtk_ValidationGetRevStatus” on page 212

“Vtk_ValidationGetValHdl” on page 217

“Vtk_ValidationValidate” on page 223

Vtk_ValidationAddCertChain

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_ValidationAddCertChain(
   const Vtk_Ctxt *ctxt,              /* input */
   Vtk_Validation *val,         /* input/output */
   const Vtk_Cert *cert,              /* input */
   VTK_CHAINBUILD_CALLBACK callback, /* input */
   void *userHdl                     /* input */
);
```

Description

This validation function builds a certificate chain for the specified certificate and adds all the certificates to the validation structure. The CA certificates stored in the Vtk_Ctxt are used while constructing the chain. When an application calls the Vtk_ValidationAddCertChain function, the application can use the Vtk_ChainBuildCallback function to provide a function pointer that will be called every time the Toolkit discovers a new link in the certificate chain.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
val	Validation query structure that contains certificates to be validated.
cert	Certificate for which the application wants to build a chain.
callback	Function pointer to the chain building callback function. Use NULL when the callback function is not needed.
userHdl	Application specific data pointer. The Toolkit calls this parameter. The Toolkit treats this pointer as opaque data. It will be passed to user specified callback function.

Return Value

VTK_OK	The function has completed successfully and the certificate has been added to the validation query.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The application can use the Vtk_ValidationGetValHdl function to create a validation handle after the certificate is added. The application can then use the validation handle to identify a specific certificate in the certificate chain.

See Also

“Vtk_ChainBuildCallBack” on page 113

“Vtk_ValidationGetValHdl” on page 217

Vtk_ValidationAddReqExt

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_ValidationAddReqExt(
   const Vtk_Ctxt *ctxt,       /* input */
   Vtk_Validation *val,        /* input/output */
   const Vtk_Extension *ext    /* input */
);
```

Description

This validation function adds an extension to an entire OCSP or CRT validation request. A validation request can be for one or more certificates. An extension added using this function applies to all the certificates in the request. An application can call Vtk_ExtensionNew to create an extension and can call Vtk_ExtensionInit to initialize the data. The data by can be any data that the application wants to add to a validation request.

An extension added with this function can have a VA server extension counterpart that provides Stateful Validation using the VA API described in the ValiCert Enterprise VA Installation and Administration Guide.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
val	Pointer to the validation query structure that contains certificates to be validated.
ext	Pointer to the extension to be added to the request.

Return Value

VTK_OK	The function has completed successfully and the extension has been added to the validation query.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

An application can add extensions to the entire request as well as individual certificates in the validation request.

See Also

“Vtk_ValidationAddReqExtForSingleCert” on page 206

“Vtk_ValidationAddReqExtForSingleCertHdl” on page 208

Vtk_ValidationAddReqExtForSingleCert

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_ValidationAddReqExtForSingleCert(
   const Vtk_Ctxt *ctxt,       /* input */
   Vtk_Validation *val,        /* input/output */
   const Vtk_Cert *cert,       /* input */
   const Vtk_Cert *issuerCert, /* input */
   const Vtk_Extension *ext,   /* input */
);
```

Description

This validation function adds an extension to a specific certificate in a validation request. The certificate and issuer certificate information passed in this function are used to identify the certificate to which the application wants an extension added.

Compare this function with the Vtk_ValidationAddReqExtForSingleCert function which identifies the certificate using its Vtk_ValHdl.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
val	Pointer to the validation query structure that contains certificates to be validated.
cert	Pointer to the certificate in the validation query structure to add the extension.
issuerCert	Pointer to issuer certificate of the certificate to which an extension is to be added. This value can be NULL if the issuer certificate was added into the Vtk_Ctxt.
ext	Pointer to the extension to add to the specific certificate.

Return Value

VTK_OK	The function has completed successfully and the extension has been added to the certificate in the validation query.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

An application can add extensions to the entire request as well as individual certificates in the validation request.

See Also

“Vtk_ValidationAddReqExt” on page 204

“Vtk_ValidationAddReqExtForSingleCertHdl” on page 208

Vtk_ValidationAddReqExtForSingleCertHdl

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_uint32
Vtk_ValidationAddReqExtForSingleCertHdl(
   const Vtk_Ctxt *ctxt,       /* input */
   const Vtk_ValHdl *hdl,      /* input */
   const Vtk_Extension *ext,   /* input */
);
```

Description

This validation function adds an extension to a specific certificate in a validation request. The Vtk_ValHdl passed in this function is used to identify the certificate to which the application wants an extension added. If the application wants to identify the certificate by means of its validation handle using this function, the application must first obtain the Vtk_ValHdl structure by calling the Vtk_AddCert, Vtk_AddCertRaw, or Vtk_ValidationGetValHdl function.

Compare this function with the Vtk_ValidationAddReqExtForSingleCert function which allows an application to identify a specific certificate using the certificate/issuer pair.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
hdl	Pointer to the certificate in the validation query structure to add the extension.
ext	Pointer to the extension to add to the specific certificate.

Return Value

VTK_OK	The function has completed successfully and the extension has been added to the certificate in the validation query.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

An application can add extensions to the entire request as well as individual certificates in the validation request.

See Also

“Vtk_ValidationAddCert” on page 198

“Vtk_ValidationAddCertRaw” on page 200

“Vtk_ValidationAddReqExt” on page 204

“Vtk_ValidationAddReqExtForSingleCertHdl” on page 208

“Vtk_ValidationGetValHdl” on page 217

Vtk_ValidationDelete

```
#include <vtk_valid.h>
#include <vtk_err.h>

void Vtk_ValidationDelete(
   Vtk_Validation *val    /* input */
);
```

Description

This validation function releases memory and resources previously allocated to the validation query structure with the Vtk_ValidationNew function. Once this function completes successfully, the Vtk_Validation structure becomes invalid.

Parameters

val	Pointer to the Vtk_Validation query structure for which memory and resources are to be released.

Return Value

none	The function has completed successfully and the validation query structure has been deleted.

Notes

The entire structure and all validation queries contained within are released. The application must call this function for each validation query structure created using Vtk_ValidationNew. If the application does not call Vtk_ValidationDelete, memory leaks and other problems can occur. For more information about the Toolkit memory model, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_ValidationGetValHdl” on page 217

Vtk_ValidationGetRevStatus

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_ValidationGetRevStatus(
   const Vtk_Ctxt *ctxt,            /* input */
   const Vtk_Validation *val,       /* input */
   const Vtk_Cert *cert,            /* input */
   const Vtk_Cert *issuerCert,      /* input */
   Vtk_uint32 *status,              /* output */
   Vtk_ValRespDetails **respDetails,
                                 /* input/output */
   Vtk_ValRespSingleCertDetails **certDetails
                                  /* input/output */
);
```

Description

This validation function retrieves validation status information for a single certificate. The application can request detailed revocation information for the entire validation response or a single certificate. For information about the detailed information, see Vtk_ValRespDetails or Vtk_ValRespSingleCertDetails.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
val	Pointer to the validation data structure that encapsulates a set of validation queries that can be sent to one or more VAs. The validation query structure is opaque to your application.
cert	Pointer to the certificate for which status information is being requested.
issuerCert	Pointer to the issuer certificate.

status	Pointer to the status information for the certificate. The status values are defined as a bit field. Therefore, a single status value can represent multiple status codes. For a list of the possible certificate status codes, see Appendix A, "Error and Status Codes."
respDetails	Address to the pointer to the detailed revocation information for the entire response. (It points to a structure allocated by the Toolkit in this call.) This function returns the requested information in the structure. In cases where the result is not needed, the application can pass in NULL. The application must release this structure using the Vtk_ValRespDetailsDelete function.
certDetails	Address to the pointer to the detailed revocation information for a single certificate response. (It points to a structure allocated by the Toolkit in this call.) This function returns the requested information in the structure. In cases where the result is not needed, the application can pass in NULL. The application must release this structure using the Vtk_ValRespSingleCertDetailsDelete function.

Return Value

VTK_OK	The function has completed successfully and if specified, returns the Vtk_ValRespDetails or Vtk_ValRespSingleCertDetail structure.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

This function is similar to the Vtk_ValHdlGetRevStatus function which allows the application to return this type of information directly for a certificate added to the validation query with a validation handle.

If the application requests detailed revocation information (either respDetails or certDetails) but this information is not available in the response, the function returns VTK_OK, but the return values in the structure are not set.

See Also

“Vtk_ValRespDetails” on page 107

“Vtk_ValRespSingleCertDetails” on page 109

“Vtk_ValRespDetailsDelete” on page 229

“Vtk_ValRespSingleCertDetailsDelete” on page 231

Vtk_ValidationGetQueries

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_ValidationGetQueries(
   const Vtk_Ctxt *ctxt,       /* input */
   const Vtk_Validation *val, /* input */
   int *valQueryCount,         /* input */
   Vtk_ValQuery ***queriesOut /* input/output */
);
```

Description

This validation query function obtains an array of validation query messages contained in the Vtk_Validation data structure. After obtaining the query messages, the application instead of the Toolkit is responsible for communication with each VA returned in the queries.

The application sets the response it obtains from the VA in the response field of the Vtk_ValQuery structure. To complete the validation of these queries the application calls the Vtk_ValidationValidateFromQueries function to check the information in the responses. It must then release the memory occupied by the response buffers.

An application might want to handle the communication with the VA and complete validation with the Vtk_ValidationFromQueries for any of the following reasons:

- user wants to use SSL for the communication with the VA
- user wants asynchronous I/O with the VA
- user has their own source of validation responses or CRLs

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
val	Pointer to the data structure that encapsulates a set of validation queries that can be sent to one or more VAs.
valQueryCount	Number of query messages that are in the Vtk_ValQuery array.
queriesOut	Pointer to the array of validation queries that the application sends to the VA for validation.

Return Value

VTK_OK	The function has completed successfully and the validation queries have been added to the array.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The application must delete the returned array using the Vtk_ValQueriesDelete function. The application is also responsible for releasing the memory allocated to Vtk_Buffer in the response, which is not released by the Vtk_ValQueriesDelete function.

See Also

“Vtk_Validation” on page 106

“Vtk_ValQuery” on page 111

“Vtk_ValidationValidateFromQueries” on page 225

“Vtk_ValQueriesDelete” on page 227

Vtk_ValidationGetValHdl

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_ValidationGetValHdl(
   const Vtk_Ctxt *ctxt,            /* input */
   const Vtk_Validation *val,       /* input */
   const Vtk_Cert *cert,            /* input */
   const Vtk_Cert *issuerCert,      /* input */
   Vtk_ValHdl **hdl                /* output */
);
```

Description

This validation function creates a validation handle for a specific certificate within a validation query. This function is useful if the application added certificates to the query using the Vtk_ValidationAddCertChain function or did not specify the Vtk_ValHdl when it added the certificate using the Vtk_ValidationAddCert or Vtk_ValidationAddCertRaw function.

Once the application obtains the Vtk_ValHdl, it can use it to obtain detailed revocation information or specify certificate extensions in the validation request.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
val	Pointer to the validation data structure that encapsulates a set of validation queries that can be sent to one or more VAs. The validation query structure is opaque to the your application.
cert	Pointer to the certificate for which a validation handle is being requested.
issuerCert	Pointer to the issuer certificate.
hdl	Pointer to a auxiliary data structure used to link individual certificate validation requests to their detail.

Return Value

VTK_OK	The function has completed successfully and if specified, returns the Vtk_ValHdl structure.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The application can use the Vtk_ValidationGetValHdl function to create a validation handle after the certificate is added. The application added through the Vtk_ValidationAddCertChain function.

If the application creates a Vtk_ValHdl structure, it can set extensions, obtain validation status, and obtain details specific to a certificate instead of as an aggregate value of all the certificates. However, since memory is allocated to this auxiliary structure, the application must call the Vtk_ValHdlDelete function to release it when the structure is no longer needed.

See Also

“Vtk_ValHdlDelete” on page 193

“Vtk_ValidationAddCert” on page 198

“Vtk_ValidationAddCertRaw” on page 200

“Vtk_ValidationAddCertChain” on page 202

“Vtk_ValidationAddReqExtForSingleCertHdl” on page 208

Vtk_ValidationNew

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_Validation* Vtk_ValidationNew(
   const Vtk_Ctxt *ctxt     /* input */
);
```

Description

This validation function initializes and allocates memory for a validation query. This structure can encapsulate one or more validation queries.

Once the validation query structure has been initialized, the application can add certificates to be validated using the Vtk_ValidationAddCert or Vtk_ValidationAddCertRaw function.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.

Return Value

Vtk_Validation	The function has completed successfully. The function returns the Vtk_Validation structure that has been newly created and initialized.
NULL	The function has failed.

Notes

This function creates an empty Vtk_Validation structure. When the application no longer needs this structure it must call the Vtk_ValidationDelete function to release memory and resources allocated to this structure.

See Also

“Vtk_ValidationAddCert” on page 198

“Vtk_ValidationAddCertRaw” on page 200

“Vtk_ValidationDelete” on page 210

“Vtk_ValidationValidate” on page 223

Vtk_ValidationSetVaInfo

```
#include <vtk_valid.h>
#include <vtk_defs.h>

Vtk_uint32 Vtk_ValidationSetVaInfo(
   const Vtk_Ctxt *ctxt,          /* input */
   Vtk_Validation *val,           /* input */
   const char *vaUrl,             /* input */
   enum Vtk_ValidationMech mech, /* input */
   const Vtk_CertStore *vaCerts   /* input */
);
```

Description

This validation function sets specific protocol and VA information for this validation operation for the certificates in the Vtk_Ctxt or Vtk_CertStore (if specified).

The information set in this function overrides the protocol and VA information currently set for the Vtk_Ctxt resulting in this Vtk_Validation structure being specific to the specified VA.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
val	Pointer to the Vtk_Validation query structure.
vaUrl	URL for the VA, for example http://ocsp.valicert.net:80.
mech	Validation mechanism type employed by the user. The possible values are: ❖ VTK_VM_CRT ❖ VTK_VM_OCSP For more information about these values, see “Vtk_ValidationMech” on page 80.

vaCerts	Pointer to the certificate store that contains the VA certificates that are to be used to validate responses from the VA. This parameter is optional. If it is not set, the settings are applied to the certificates specified in the context. See “Vtk_CertStore” on page 90.

Return Value

VTK_OK	The function has completed successfully and the VA information and protocol have been set.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

This function copies the Vtk_CertStore structure. Therefore, applications must call the Vtk_CertStoreDelete to release the memory and resources allocated to the Vtk_CertStore structure.

See Also

“Vtk_CertStore” on page 90

“Vtk_CertStoreDelete” on page 145

Vtk_ValidationValidate

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_ValidationValidate(
   const Vtk_Ctxt *ctxt,     /* input */
   Vtk_Validation *val,      /* input */
   Vtk_uint32 *valStatus     /* output */
   );
```

Description

This validation function sends the validation request(s) to the VA to perform the validation. Calling this function can result in one or more validation queries being sent to one or several VAs.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
val	Validation query structure that contains certificates to be validated.
valStatus	Aggregate validation status of all the certificates in the validation query structure. The function must return VTK_OK for the status field to be valid. All the certificates must pass validation for the validation status to be VTK_STATUS_OK.

Return Value

VTK_OK	The function has completed successfully and the validation has completed.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

Since the validation status is an aggregate value, one revoked certificate causes the value of the returned status to be revoked.

The application can obtain validation status for an individual certificate by including a Vtk_ValHdl auxiliary data structure when the certificate is added to the list of certificates to be validated. A certificate is added using the Vtk_ValidationAddCert and Vtk_ValidationAddCertRaw functions. Retrieving the validation status for the individual certificate can be done using the Vtk_ValHdlGetRevStatus or Vtk_ValidationGetRevStatus function.

See Also

“Vtk_ValHdlGetRevStatus” on page 195

“Vtk_ValidationAddCert” on page 198

“Vtk_ValidationAddCertRaw” on page 200

“Vtk_ValidationGetValHdl” on page 217

Vtk_ValidationValidateFromQueries

```
#include <vtk_valid.h>
#include <vtk_err.h>

Vtk_uint32 Vtk_ValidationValidateFromQueries(
   const Vtk_Ctxt *ctxt,       /* input */
   Vtk_Validation *val,        /* input */
   Vtk_ValQuery **valQueries, /* input/output */
   Vtk_uint32 *valStatus       /* output */
);
```

Description

This validation query function validates the validation tokens or responses supplied by the application.

This function is called only after the application has called the Vtk_ValidationGetQueries function to obtain an array of validation query messages contained in the Vtk_Validation data structure, communicates with the VA, and places the information it obtains from the VA in the response field of each query.

Unlike the Vtk_Validation function, this function does not handle the communication with the VA and does not set the response in the query structure.

Parameters

ctxt	Pointer to Toolkit context created using the Vtk_CtxtNew function and for which memory has been allocated.
val	Validation query structure that contains certificates to be validated.
valQueries	Pointer to the array of validation queries that the application wants to have validated by the VA.
valStatus	Aggregate validation status of all the certificates in the validation query structure. The function must return VTK_OK for the status field to be valid.

Return Value

VTK_OK	The function has completed successfully and validation has completed.
error code	The function has failed. For information about possible error values, refer to Appendix A, "Error and Status Codes."

Notes

The Vtk_ValQuery array is created using the Vtk_ValidationGetQueries function. Since the validation status is an aggregate value, one revoked certificate causes the value of the returned status to be revoked.

See Also

“Vtk_Validation” on page 106

“Vtk_ValQuery” on page 111

“Vtk_ValidationGetQueries” on page 215

“Vtk_ValQueriesDelete” on page 227

Vtk_ValQueriesDelete

```
#include <vtk_valid.h>
#include <vtk_err.h>

void Vtk_ValQueriesDelete(
   Vtk_ValQuery **queriesToDelete
);
```

Description

This validation query function releases memory and resources occupied by the Vtk_ValQuery array allocated by the Vtk_ValidationGetQueries function. However, the response data in each Vtk_ValQuery is not released and remains the responsibility of the application. Once this function completes successfully, the Vtk_ValQuery structure becomes invalid.

Parameters

queriesToDelete	Pointer to the Vtk_ValQuery structure for which memory and resources are to be released.

Return Value

none	The function has completed successfully and the validation query array has been deleted.

Notes

The application must call this function for each Vtk_ValQuery array created using Vtk_ValidationGetQueries. If the application does not call Vtk_ValQueriesDelete, memory leaks and other problems can occur. For more information about the Toolkit memory model, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_ValQuery” on page 111

“Vtk_ValidationGetQueries” on page 215

“Vtk_ValidationValidateFromQueries” on page 225

Vtk_ValRespDetailsDelete

```
#include <vtk_valid.h>
#include <vtk_err.h>

void Vtk_ValRespDetailsDelete(
   Vtk_ValRespDetails *details    /* input */
);
```

Description

This validation function releases memory and resources previously allocated by the Vtk_ValHdlGetRevStatus or Vtk_ValidationGetRevStatus function. Once this function completes successfully, the Vtk_ValRespDetails structure becomes invalid.

Parameters

details	Pointer to the Vtk_ValRespDetails structure for which memory and resources are to be released.

Return Value

none	The function has completed successfully and the validation context has been deleted.

Notes

The application must call this function for each detailed response created using Vtk_ValHdlGetRevStatus or Vtk_ValidationGetRevStatus. If the application does not call Vtk_ValRespDetailsDelete, memory leaks and other problems can occur. For more information about the Toolkit memory model, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_ValHdlGetRevStatus” on page 195

“Vtk_ValidationGetRevStatus” on page 212

Vtk_ValRespSingleCertDetailsDelete

```
#include <vtk_valid.h>
#include <vtk_err.h>

void Vtk_ValRespSingleCertDetailsDelete(
   Vtk_ValRespSingleCertDetails* details
                                      /* input */
);
```

Description

This validation function releases memory and resources previously allocated by the Vtk_ValHdlGetRevStatus or Vtk_ValidationGetRevStatus function. Once this function completes successfully, the Vtk_ValRespSIngleCertDetails structure becomes invalid.

Parameters

details	Pointer to the Vtk_ValRespSIngleCertDetails structure for which memory and resources are to be released

Return Value

none	The function has completed successfully and the validation context has been deleted.

Notes

The application must call this function for each detailed response created using Vtk_ValHdlGetRevStatus or Vtk_ValidationGetRevStatus. If the application does not call Vtk_ValRespSingleCertDetailsDelete, memory leaks and other problems can occur. For more information about the Toolkit memory model, see “Toolkit Memory Model” on page 9.

See Also

“Vtk_ValHdlGetRevStatus” on page 195

“Vtk_ValidationGetRevStatus” on page 212

Vtk_WriteLog

```
include <vtk_error.h>
include <vtk_defs.h>

void Vtk_WriteLog(
   const Vtk_Ctxt *pCtxt,      /* input */
   enum Vtk_CtxtLogType type,  /* input */
   const char *pMsg            /* input */
);
```

Description

This function writes a message into the log file or invokes any specified application specific WriteLog callback function.

An application can use this function to put extra information into the log file that was opened by the Vtk_OpenLog function. The default Toolkit implementation is to write a message to the log file in the following format:

LogType: Timestamp MessageDescription

For example:

```
I: [28/Feb/2000 10:58:33.172 -0800] Vtk_CertLoadFromFile:
Loading certificate from \test\test_cert.cer
D: [28/Feb/2000 10:58:33.413 -0800] CrtInit: Operation completed
successfully
I: [28/Feb/2000 10:58:33.413 -0800] CrtValidate CRT Request sent
to ci.valicert.net (port 80)
E: [28/Feb/2000 13:36:44.740 -0800] CrtValidate: Unable to
validate the response from the VA Source
File:D:\src\cryptosoft\libs\vctoolkit\vtk_qcrt.c (374)
```

Parameters

pCtxt	A pointer to the Toolkit context.
type	The type of log message (see the enumerated type Vtk_CtxtLogType for details).
pMsg	The log message.

Return Value

None

Notes

To provide an alternate log writing function define Vtk_OpenLogCallback, Vtk_CloseLogCallback, and Vtk_WriteLogCallback.

See Also

“Data Structures” on page 81

“Vtk_WriteLogCallback” on page 123

“Vtk_CtxtSetOption” on page 166

“Vtk_LogOptions” on page 99

“Vtk_OpenLog” on page 189

APPENDIX

A

Error and Status Codes

This section contains the define statements in the error.h header file. It defines the following two categories:

- Error Codes
- Status Codes

The application can translate these error codes and status codes into strings that are specific to their needs using the Vtk_ErrorToString and Vtk_StatusToStrings functions provided in the Toolkit.

Error Codes

These are the return codes for the functions.

#define VTK_OK	0
#define VTK_ERR_BAD_LEAF_COUNT	1
#define VTK_ERR_UNSUPPORTED_DATA_TYPE	2
#define VTK_ERR_BAD_SIGNATURE	3
#define VTK_ERR_OUT_OF_MEMORY	4
#define VTK_ERR_BAD_FILE	5
#define VTK_ERR_INDEX_OUT_OF_BOUNDS	6
#define VTK_ERR_INTERNAL	7
#define VTK_ERR_BAD_ASN1	8
#define VTK_ERR_BAD_LEAF_POSITION	9
#define VTK_ERR_UNABLE_TO_GET	10
#define VTK_ERR_INCOMPATIBLE_VERSION	11

```
#define VTK_ERR_BAD_NONCE                       12
#define VTK_ERR_BAD_CRL                         13
#define VTK_ERR_NO_TRUSTED_CA                   14
#define VTK_ERR_WRONG_CRL                       15
#define VTK_ERR_BAD_PARMS                       16
#define VTK_ERR_OP_NOT_SUPPORTED                17
#define VTK_ERR_NO_CERTS_TO_VALIDATE            18
#define VTK_ERR_BAD_URL                         19
#define VTK_ERR_NO_DATA                         20
#define VTK_ERR_NO_VALHDL_INFO                  21
#define VTK_ERR_BAD_VALHDL                      22
#define VTK_ERR_UNKNOWN_VA                      23
#define VTK_ERR_LDAP_SIZE_LIMIT_EXCEEDED        24
#define VTK_ERR_LDAP_SEARCH                     25
#define VTK_ERR_CRL_FORMAT                      26
#define VTK_ERR_CONNECT                         27
#define VTK_ERR_ERR_SEND_REQ                    28
#define VTK_ERR_SOCK_READ                       29
#define VTK_ERR_NULL_RESPONSE                   30
#define VTK_ERR_BAD_RESPONSE                    31
#define VTK_ERR_BAD_MIME_HDR                    32
#define VTK_ERR_VA_REFUSED                      33
#define VTK_ERR_INIT_COMM                       34
#define VTK_ERR_CERTS_DO_NOT_MATCH              35
#define VTK_ERR_VA_NO_CER_INFO                  36
#define VTK_ERR_CERT_PATH_TOO_DEEP              37
#define VTK_ERR_NO_ISSUER_CERT                  38
```

#define VTK_ERR_URL_TO_LONG	39
#define VTK_ERR_DECODING_PKCS7	40
#define VTK_ERR_CTXT_OPTION_NOT_SET	41
#define VTK_ERR_UNABLE_TO_LOAD_LDAP_LIB	42
#define VTK_ERR_UNABLE_TO_LOAD_LDAP_FUNCS	43
#define VTK_ERR_LDAP_AUTH	44
#define VTK_ERR_LDAP_NOT_FOUND	45
#define VTK_ERR_USER_CALLBACK	46
#define VTK_ERR_NOT_FUND	47
#define VTK_ERR_SRV_MALFORMED_REQUEST	48
#define VTK_ERR_SRV_INTERNAL_ERROR	49
#define VTK_ERR_SRV_TRY_LATER	50
#define VTK_ERR_SRV_SIG_REQUIRED	51
#define VTK_ERR_SRV_UNAUTHORIZED	52
#define VTK_ERR_OPEN_LOG_FILE	53
#define VTK_ERR_LOG_ALREADY_OPEN	54
#define VTK_ERR_CRYPTO_LIB_INIT	55

Status Codes

The status values are defined as bit field. This means a single status value can represent multiple status codes.

#define VTK_STATUS_OK	0x1
#define VTK_STATUS_UNKNOWN	0x2
#define VTK_STATUS_REVOKED	0x4
#define VTK_STATUS_CRL_EXPIRED	0x8
#define VTK_STATUS_CRL_NOT_YET_VALID	0x10

```
#define VTK_STATUS_RESP_EXPIRED                0x20
#define VTK_STATUS_RESP_NOT_YET_VALID          0x40
```

Index