SERENA®

PVCS® VERSION MANAGER™ 8.4

Developer's Toolkit Reference Guide

Serena Proprietary and Confidential Information

Publication date: August 2010

Table of Contents

Welcome to Version Manager

Thank you for choosing Serena PVCS Version Manager, a powerful and versatile version control system that will revolutionize the way you develop software. Version Manager helps you organize, manage, and protect your software development projects on every level—from storing and tracking changes to individual files, to managing and monitoring an entire development cycle.

Purpose of this manual

The *Serena PVCS Version Manager Developer's Toolkit Reference Guide* contains information for programmers using Developer's Toolkit to build applications that use Serena PVCS Professional Suite services.

For more information

Refer to the *Serena PVCS Version Manager Getting Started Guide* for a description of the Version Manager documentation set, a summary of the ways to work with Version Manager, and instructions for accessing the Online Help.

Using this Manual

Information for programmers

The *Serena PVCS Version Manager Developer's Toolkit Reference Guide* contains information for programmers using Developer's Toolkit to build applications that use Serena PVCS Professional Suite services.

The Developer's Toolkit Reference Guide contains these chapters:

- Chapter 1, "Introduction" on page 9, presents an overview of the Developer's Toolkit, and explains how to compile and link applications that use Serena PVCS Professional Suite services.
- Chapter 2, "Serena PVCS Version Manager Functions" on page 17, describes Version Manager functions in alphabetic order.
- Chapter 3, "Serena Configuration Builder Functions" on page 183, describes Configuration Builder functions in alphabetic order.
- Chapter 4, "Return Values" on page 195, lists values returned by Professional functions, along with an explanation of each error.
- Chapter 5, "Data Structures" on page 203, lists data structures used by Professional functions.

Information about Version Manager functions

The function descriptions in Chapters 2 and 3 include function syntax, parameters, return values, and code examples. Developer's Toolkit includes source files that contain code used in the examples.

Typographical Conventions

The following typographical conventions are used in the online manuals and online help. These typographical conventions are used to assist you when using the documentation;

they are not meant to contradict or change any standard use of typographical conventions in the various product components or the host operating system.

Convention	Explanation
italics	Introduces new terms that you may not be familiar with and occasionally indicates emphasis.
bold	Emphasizes important information and field names.
UPPERCASE	Indicates keys or key combinations that you can use. For example, press the ENTER key.
monospace	Indicates syntax examples, values that you specify, or results that you receive.
monospaced italics	Indicates names that are placeholders for values you specify; for example, *`filename`*.
monospace bold	Indicates the results of an executed command.
vertical rule |	Separates menus and their associated commands. For example, select File | Copy means to select Copy from the File menu. Also, indicates mutually exclusive choices in a command syntax line.
brackets []	Indicates optional items. For example, in the following statement: `SELECT [DISTINCT]`, DISTINCT is an optional keyword.
. . .	Indicates command arguments that can have more than one value.

Contacting Technical Support

Registered customers can log in to http://support.serena.com/.

Chapter 1
Introduction

Overview

Use Version Manager functions in your application

Serena PVCS Version Manager Developer's Toolkit provides application programming interface (API) functions for programmers who want to develop applications that use Serena PVCS Professional Suite services. You can use the functions in Developer's Toolkit to build your own interface to Serena configuration management tools, or to add configuration management capabilities to your applications.

Visual Basic and Delphi programmers can also utilize the Serena PVCS Version Manager Developer's Toolkit functionality easily without having to write external functions to manipulate certain data structures normally returned by a few Developer's Toolkit functions.

The Developer's Toolkit includes functions that implement the functionality of Serena PVCS Version Manager and Serena Configuration Builder.

You can use Developer's Toolkit to build applications that use Serena PVCS Professional Suite services on various platforms.

Components of the Toolkit

DLLs, libraries, and header files

Serena PVCS Version Manager Developer's Toolkit consists of a set of dynamic link libraries (DLLs), import libraries, shared libraries, two static libraries, and header files.

Dynamic Link Libraries

Windows DLLs

The Developer's Toolkit includes the following DLLs for Windows:

- VMWFDTK.DLL contains functions for Version Manager services for 32-bit Windows.
- CBWF51.DLL contains functions for Configuration Builder services for 32-bit Windows.

NOTE When you are using the Serena PVCS Professional Suite services, the calling application must be able to find the DLLs. Under Windows, the DLLs must be in the same directory as the calling application or in a directory included in the PATH statement.

Import Libraries

Windows libraries

The Developer's Toolkit includes the following import libraries for linking Windows programs:

- VMWFDTK.LIB is the 32-bit import library for Version Manager functions.
- CBWF51.LIB is the 32-bit import library for Configuration Builder functions.

Shared UNIX Libraries

Shared Libraries

The Developer's Toolkit includes the following shared libraries for UNIX:

- *libpvcsvm.a* is the shared UNIX object library for Version Manager for AIX.
- *libpvcsvm.sl* is the shared UNIX object library for Version Manager for HP-UX.
- *libpvcsvm.so* is the shared UNIX object library for Version Manager for Solaris.

Static Library

Static UNIX library

The Developer's Toolkit includes two static libraries for UNIX:

- *pvcscb.a* is the UNIX object library for Configuration Builder functions.
- *pvcsdtk.a* is the UNIX object library for Version Manager functions.

Header Files

The Developer's Toolkit includes the following C header files:

- *pvcs.h* contains definitions and function prototypes for the Serena PVCS Professional Suite services.
- *pvcsb.h* contains definitions and function prototypes for Configuration Builder functions. This file is automatically included by *pvcs.h*.
- *pvcsvm.h* contains definitions and function prototypes for Version Manager functions. This file is automatically included by *pvcs.h*.
- *pvcsvm.bas* contains definitions and function prototypes for use with Visual Basic applications.
- *pvcsdtk.pas* contains definitions and function prototypes for use with Delphi applications.

NOTE When compiling under Windows, your program should include *windows.h* before including *pvcs.h*.

Calling Conventions

Unless stated otherwise, any revision parameter can be a revision number, version label, or promotion group. If the parameter refers to a version label that begins with a number, you must prefix the label with a backslash (\) so that Version Manager does not recognize it as a revision number.

All functions return a value of zero if successful. Non-zero return values indicate that an error occurred. See Chapter 4, "Return Values" for descriptions of the error codes.

Developer's Toolkit uses the following conventions for Windows DLLs:

- The CDecl calling convention for 32-bit Windows.

Developer's Toolkit uses the following convention for UNIX:

- The C language calling convention.

Data Structures

Types of returned data

Many Serena PVCS Professional Suite services return data to the caller. There are three types of returned data:

- Simple data types, such as integer.
- Null-terminated ASCII text strings.

- Data structures. These data structures are listed as C structures in the header files PVCSVM.H and PVCSCB.H, and in Chapter 5, "Data Structures"

The caller must allocate memory for data returned by the Professional services. The data structures contain a fixed-length structure, which may contain pointers to variable-length strings. The buffer must be large enough to hold the fixed portion plus the variable portion.

The Developer's Toolkit returns the error code `PVCS_E_BUFFER_OVERFLOW` if the buffer is too small.

Structures use the Microsoft C convention of even-byte alignment. This means that all structure members begin on an even-byte address.

Null Parameters

Null pointer equals pointer to a null string

Unless otherwise stated, functions with parameters that are pointers to text strings treat a null pointer and a pointer to a null string as equivalent.

Compiling and Linking Windows Applications

Compiling tips

The following tips on compiling and linking applications use the Professional API functions:

- **Public names**. Professional services are case-insensitive. If you are using the Microsoft C compiler, the header files declare the functions as *Pascal* type, which means that the compiler converts all public symbols to upper case in generated object code.
- **Stack**. Allocate an additional 4K beyond your application's normal stack requirements. (Use the /Stack command-line option with the Microsoft linker.)
- **Single thread**. Professional services were not designed to be used with multiple threads. Results are unpredictable if a process has more than one active thread executing Professional functions. Multiple *processes* executing Professional functions will not cause problems.
- **Memory model**. Professional services use the flat memory model for Windows: 32-bit values are used for everything.

Examples

The following compiling and linking examples are for Windows using Microsoft win32 SDK:

```
set LIB=%LIB%;\pvcsdtk\lib
set INCLUDE=%INCLUDE%;\pvcsdtk\include
cl386 -c -Di386 -D_X86_ -G3 -Foobject_file source_file
link32 -machine:i386 -out:executable_file -subsystem:console
  object_file
  pvcsvmn.lib
  pvcscbn.lib
  crtdll.lib
  kernel32.lib
  mpr.lib
  advapi32.lib
```

MS Visual C++:

```
set LIB=\msvcnt\lib;\pvcsdtk\lib
set INCLUDE=\msvcnt\include;\pvcsdtk\include
set LINK= -machine:i386 -subsystem:console
CL /c /W3 /G3 test.c
LINK test.obj,vmwfdtk + libcmt + kernel32 + advapi32
```

Compiling and Linking UNIX Applications

Tips on compiling

To compile a module that includes Developer's Toolkit functions, add the directory containing the header files to the compiler's header file search path. For UNIX C compilers, you generally use the -I option to specify an additional directory in which to search for header files.

Examples

In the examples below, *os* is sol, hpux, or aix. To compile a module named *test.c* that includes a Developer's Toolkit function, type:

```
cc -I/usr/pvcs/vm/os/dtk -DCDECL="" -DSYS_UNIX -c test.c
```

To link programs that use Developer's Toolkit, name all of the Developer's Toolkit libraries on the command line.

The following examples link a program called *test*, which uses both Version Manager and Configuration Builder functions.

To link the application, enter:

```
cc -o test test.o \
   /usr/pvcs/vm/os/dtk/pvcsvm \
     /usr/pvcs/vm/os/dtk/pvcscb.a \
```

where *pvcsvm* specifies the shared object library for Version Manager for the operating system. The value can be any of the following:

- for *AIX*: `pvcsvm.a`
- for *HP-UX*: `pvcsvm.sl`
- for *Solaris*: `pvcsvm.so`

For programs that require the Configuration Builder functions, you must link in *pvcscb.a*. If you are building a static application, use pvcsdtk instead of pvcsvm.

Initializing the Toolkit and Setting Global Parameters

Initialize the Toolkit first

Before performing Version Manager operations, you must first use the PvcsInit function to initialize the Toolkit. If you don't initialize the Toolkit, the first function you call will call PvcsInit for you. See "PvcsInit" on page 113 for details.

Set global parameters

After you have initialized the toolkit, you should set global parameters that affect all Developer's Toolkit functions. See "PvcsSetGlobalParameter" on page 168 for details.

Reading the Configuration File

Read the configuration file first

Before performing most Version Manager operations, you should first use the **PvcsQueryConfiguration** function to read the configuration file, which defines many aspects of Version Manager's behavior.

PvcsQueryConfiguration finds the default configuration file using the usual Version Manager rules. (These rules are listed in the *Serena PVCS Version Manager Command-Line Reference Guide*.) You can specify a configuration file, which overrides the normal rules for locating the configuration file.

If you do not call **PvcsQueryConfiguration**, the Professional services use the default configuration settings. If your application changes the current directory to a directory that contains a local configuration file, you should call **PvcsQueryConfiguration** again to update the configuration settings.

Using Archive Handles

Functions to open and close archives

Before accessing an archive, you can open it with **PvcsOpenArchive**. This function returns an archive handle, which you pass to other Professional services. When you are finished with the archive, call **PvcsCloseArchive**, which applies pending updates to the archive, closes it, and frees internal resources associated with it.

Functions can open archives automatically

It is not necessary to use **PvcsOpenArchive** to open an archive before processing it. If you use the services that access archives with an archive name rather than an archive handle, they can open and close the archives themselves.

However, if you are performing multiple operations on an archive, it is more efficient to open the archive first and pass the archive handle to other services.

Close or keep archives open

If you have several operations to perform on an archive, you can close the archive after each, or you can close it after the last operation. Here are the advantages and disadvantages of each:

- **Close after each operation**. This makes each operation a separate event. The advantage is that the archive is free between operations, which allows access by other users. However, there is more overhead associated with opening and closing the archive multiple times, and another user could update the archive between operations.
- **Close after last operation**. The archive remains open throughout the operations, which has the advantage of making all operations appear as a single transaction. If you open the archive in update mode, no other users can update the archive until you close it.

Operating on multiple archives

If you will perform operations on multiple archives, make sure the you close each archive before opening another. If more than one archive is opened at a time, the results of toolkit operations will be undefined.

Changes to an archive are not saved until you call **PvcsCloseArchive**. If you need to cancel an update, call **PvcsCancelUpdate**.

Using DTK Applications with a Version Manager File Server

If an access control database or LDAP authentication is associated with the Client Name (path map) on the Version Manager File Server, you must use the PvcsLogin function or an environment variable to present a user ID and password for validation.

By default, Version Manager checks for a user ID and password authentication source in the following order:

1. **MERANT_LOGIN_PRIORITY:** An environment variable. Valid values are: "PCLI_ID", "EVENTUSERID", and "*user_id*:*user_password*".
2. **PCLI_ID:** An environment variable. Its value is in the form of "*user_id*:*user_password*".
3. **EVENTUSERID:** An environment variable in the form of "*user_id*" used in combination with the EVENTPASSWORD variable (which contains a limited-lifetime encrypted password). Values for these variables are generated when an external command is executed from an Event Trigger, a Toolbar Command, or when using the PCLI `run -e` command.

Special Considerations

- Version Manager validates against the first authentication source that contains a valid value ("PCLI_ID", "EVENTUSERID", or "*user_id*:*user_password*"). It does NOT check subsequent authentication sources even if the initial one fails.
- You can cause PCLI_ID or EVENTUSERID to be the first authentication source that is checked by assigning the value "PCLI_ID" or "EVENTUSERID" to the environment variable MERANT_LOGIN_PRIORITY.
- You can pass a "hardcoded" user ID and password by assigning a "*user_id*:*user_password*" value to the environment variable MERANT_LOGIN_PRIORITY. No other authentication sources will be checked.
  NOTE You must include the colon (:) even if no password is required.

For information on setting up client access to a Version Manager File Server, see the *Serena ChangeMan Version Manager Administrator's Guide*.

Chapter 2

Serena PVCS Version Manager Functions

IDENT

This function returns information about the version of the Serena PVCS Version Manager Developer's Toolkit. The information returned is the version string of the Toolkit. The Version Manager desktop client displays this string in the "About" box.

Syntax

```
IDENT(
   unassigned char *buffer, /* Output */
   unsigned short bufLen) /* Input */
```

Parameters

buffer	Buffer receiving ident string.
bufLen	Length of buffer.

Return Values The return value is zero if the function is successful.

PvcsAccessAddGroupGroup

This function adds a new group member to a group and requires the ViewAccessDB privilege. The group must already exist in the database.

Syntax

```
PvcsAccessAddGroupGroup(
   int DBHandle,     /* Input */
   unsigned char *group,/* Input */
   unsigned char *groupMember)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
group	Pointer to a string that contains the name of the group to which you are adding a member.
groupMember	Pointer to a string that contains the name of the group member to be added.

Return Values The return value is zero if the function is successful. Other values may be:

```
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_GROUP
PVCS_E_GROUP_EXISTS
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
    &DBHandle);

/* Add group member to group */
status = PvcsAccessAddGroupGroup(
```

```
   DBHandle,/* Database handle */
    groupName,/* Name of group */
    groupMemberName);/* Name of group to add */

if (!status)
   printf("Added group \"%s\" to group \"%s\".\n",
    groupMemberName, groupName);

/* Close the access control database */
status = PvcsAccessCloseDB(DBHandle);
```

Related Functions
PvcsAccessAddGroupUser on page 21
PvcsAccessDeleteGroupGroup on page 28
PvcsAccessDeleteGroupUser on page 29
PvcsAccessRenameGroupGroup on page 46
PvcsAccessRenameGroupUser on page 47

PvcsAccessAddGroupUser

This function adds a user to a group and requires the ViewAccessDB privilege. The user must already exist in the database.

Syntax

```
PvcsAccessAddGroupUser(
   int DBHandle,     /* Input */
   unsigned char *group,/* Input */
   unsigned char *userMember)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
group	Pointer to a string that contains the name of a group.
userMember	Pointer to a string that contains the name of a user member.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_USER
PVCS_E_UNKNOWN_GROUP
PVCS_E_USER_EXISTS

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB("access.db",PVCS_ACCOPEN_UPDATE,
   &DBHandle);

/* Add user to group */
status = PvcsAccessAddGroupUser(
   DBHandle,/* Database handle */
   groupName,/* Name of group */
   userName);/* Name of user to add */
```

```
if (!status)
   printf("Added user \"%s\" to group \"%s\".\n", userName,
   groupName);

/* Close the access control database */
status = PvcsAccessCloseDB(DBHandle);
```

Related Functions
PvcsAccessAddGroupGroup on page 20
PvcsAccessDeleteGroupGroup on page 28
PvcsAccessDeleteGroupUser on page 29
PvcsAccessRenameGroupGroup on page 46
PvcsAccessRenameGroupUser on page 47

PvcsAccessCloseDB

This function closes the access control database. It writes the file back to disk if it was opened in create mode, or if it was opened in update mode and the caller made changes to the file.

Syntax

```
PvcsAccessCloseDB(
   int DBHandle)    /* Input */
```

Parameters

DBHandle Handle returned by **PvcsAccessOpenDB**.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_INVALID_DBHANDLE

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB(
   "access.db",/* Name of access control db */
   PVCS_ACCOPEN_UPDATE,/* Type of access to db */
   &DBHandle);/* Returned database handle */

/* Close the access control database */
status = PvcsAccessCloseDB(
   DBHandle);/* Database handle */
```

Related Functions
PvcsAccessOpenDB on page 40
PvcsMakeDB on page 123
PvcsReadDB on page 152

PvcsAccessDefineGroup

This function adds a group or changes a group's definition and requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessDefineGroup(
     int DBHandle,  /* Input */
     unsigned char *group,/* Input */
     unsigned char *privList,/* Input */
     unsigned char *userList,/* Input */
     unsigned char *groupList,/* Input */
     PVCS_FLAGS flags)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
group	Pointer to a string that contains the name of the group.
privList	Pointer to a buffer that contains the group's privilege list. Each privilege must be terminated by a null character, and the end-of-buffer must be marked by one additional null character. A null parameter indicates that the group has the Unlimited privilege.
userList	Pointer to a buffer that contains the list of users. Each user must be terminated by a null character, and the end-of-buffer must be marked by one additional null character. Each user must already be defined in the database. A null parameter indicates no users in the group.
groupList	Pointer to a buffer that contains the list of groups. Each group must be terminated by a null character.
flags	Bit field that controls the operation of this function and can be combined. The default value is: PVCS_ACCDEFINE_GROUP_ADD This setting adds a new group definition.

Return Values

The return value is zero if the function is successful. Other values may be:

PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_GROUP
PVCS_E_UNKNOWN_USER
PVCS_E_UNKNOWN_PRIVILEGE

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
   &DBHandle);

/* Add group definition */
status = PvcsAccessDefineGroup(
   DBHandle,/* Database handle */
   groupName,/* Group name */
   privilegeList,/* Group privilege info */
   userList,/* Group user info */
   groupList,/* Group group info */
```

```
      PVCS_ACCDEFINE_USER_ADD);/* Add a new group */

   if (!status)
      printf("Defined group \"%s\".\n", groupName);

   /* Close the access control database */
   status = PvcsAccessCloseDB(DBHandle);
```

Related Functions

PvcsAccessDeleteGroup on page 27
PvcsAccessEnumerateGroupGroups on page 31
PvcsAccessEnumerateGroups on page 32
PvcsAccessEnumerateGroupUsers on page 34
PvcsAccessQueryGroup on page 41
PvcsAccessRenameGroup on page 45

PvcsAccessDefinePrivilege

The function adds a new custom privilege, or changes the definition of an existing custom privilege. It requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessDefinePrivilege(
   int DBHandle,     /* Input */
   unsigned char *privilegeName,/* Input */
   unsigned char *privilegeList,/* Input */
   unsigned char *promoGroupList,/* Input */
   PVCS_FLAGS flags)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
privilegeName	Pointer to a string that contains the name of the privilege to define.
privilegeList	Pointer to a buffer that contains the custom privilege's privilege list. Each privilege is null-terminated, and the end-of-buffer must be marked by one additional null character. A null parameter indicates that the custom privilege has the Unlimited privilege.
promoGroupList	Pointer to a buffer that contains the custom privilege's promotion group list. Each promotion group is null-terminated, and the end-of-buffer must be marked by one additional null character. A null parameter indicates the customer privilege has no promotion group restrictions.
flags	Bit field that controls how the function operates. Values include: ■ PVCS_ACCDEFINE_PRIV_ADD This is the default setting; it adds a new privilege definition. ■ PVCS_ACCDEFINE_PRIV_REPLACE This setting replaces an existing privilege definition.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_PRIVILEGE
PVCS_E_BUFFER_OVERFLOW
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
   &DBHandle);

/* Add privilege definition */
status =  PvcsAccessDefinePrivilege(
   DBHandle,/* I: Handle from PvcsAccessOpenDB */
   privilegeName,/* I: Name of privilege */
   privilegeList,/* I: Buf containing privilege list */
   promoGroupList,/* I: Buf containing promo group list*/
   flags);            /* I: Function behavior flags */

if (!status)
    printf("Defined privilege \"%s\".\n", privilegeName);

/* Close the access control database */
status = PvcsAccessCloseDB(DBHandle);
```

Related Functions

PvcsAccessDeletePrivilege on page 29
PvcsAccessEnumeratePrivilege on page 35
PvcsAccessEnumeratePrivileges on page 36
PvcsAccessQueryPrivilege on page 42
PvcsAccessRenamePrivilege on page 48

PvcsAccessDefineUser

This function adds a user or changes the user's definition and requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessDefineUser(
   int DBHandle,     /* Input */
   unsigned char *userName,/* Input */
   unsigned char *password,/* Input */
   unsigned char *privList,/* Input */
   unsigned char *expDate,/* Input */
   PVCS_FLAGS flags)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
userName	Pointer to a string that contains the user name.
password	Pointer to a string that contains the user password. A null parameter indicates no password.

privList Pointer to a buffer that contains the user's privilege list. Each privilege is null-terminated, and the end-of-buffer must be marked by one additional null character. A null parameter indicates the user has the Unlimited privilege.

expDate Pointer to a string that contains the user expiration date. The date format is *mm-dd-yy hh:mm:ss*. A null parameter indicates no expiration date.

flags Bit field that controls the operation of this function and can be combined. Values include:

- PVCS_ACCDEFINE_USER_ADD
  Adds a new user definition. This is the default setting.
- PVCS_ACCDEFINE_USER_REPLACE
  Replaces an existing user definition.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_USER
PVCS_E_USER_EXISTS
PVCS_E_UNKNOWN_PRIVILEGE

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *userName = "DAVEE";
char *password = "GOSEAHAWKS";
char *privilegeList = "Get\0Put\0Lock\0";
char *expDate = "01-01-96 12:00:00";

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB('access.db', PVCS_ACCOPEN_UPDATE,
   &DBHandle);

/* Add user definition */
status = PvcsAccessDefineUser(
   DBHandle,/* Database handle */
   userName,/* User name */
   password,/* User password */
   privilegeList,/* User privilege info */
   expDate, /* User expiration date */
   PVCS_ACCDEFINE_USER_ADD);/* Add a new user */

if (!status)
   printf("Defined user \"%s\".\n", userName);

/* Close the access control database */
status = PvcsAccessCloseDB(DBHandle);
```

Related Functions

PvcsAccessDeleteUser on page 30
PvcsAccessEnumerateUserGroups on page 37
PvcsAccessEnumerateUsers on page 39

PvcsAccessQueryUser on page 43
PvcsAccessRenameUser on page 49

PvcsAccessDeleteGroup

This function deletes a group from the access control database and requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessDeleteGroup(
   int DBHandle,     /* Input */
   unsigned char *group)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
group	Name of the group to delete.

Return Value

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_GROUP

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
   &DBHandle);

/* Delete the group */
status = PvcsAccessDeleteGroup(
   DBHandle,/* Database handle */
   groupName);/* Name of group */

if (!status)
   printf("Deleted \"%s\".\n", groupName);

/* Close the access control database */
status = PvcsAccessCloseDB(DBHandle);
```

Related Functions

PvcsAccessDefineGroup on page 23
PvcsAccessDeleteGroup on page 27
PvcsAccessEnumerateGroupGroups on page 31
PvcsAccessEnumerateGroups on page 32
PvcsAccessEnumerateGroupUsers on page 34
PvcsAccessQueryGroup on page 41
PvcsAccessRenameGroup on page 45

PvcsAccessDeleteGroupGroup

This function deletes a group member from within a group where the member is itself a group. It requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessDeleteGroupGroup(
   int DBHandle,     /* Input */
   unsigned char *group,/* Input */
   unsigned char *groupMember)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
group	Pointer to a string that contains the name of the group.
groupMember	Pointer to a string that contains the name of the group member to be deleted.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_INVALID_PARAMETER
    PVCS_E_UNKNOWN_USER
    PVCS_E_UNKNOWN_GROUP
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
   &DBHandle);

/* Delete group from group */
status = PvcsAccessDeleteGroupGroup(
   DBHandle,/* Database handle */
   groupName,/* Name of group */
   groupMemberName);/* Name of group to delete */

if (!status)
   printf("Deleted group \"%s\" from group \"%s\".\n",
            groupMemberName, groupName);
/* Close the access control database */
status = PvcsAccessCloseDB(DBHandle);
```

Related Functions

PvcsAccessAddGroupGroup on page 20
PvcsAccessAddGroupUser on page 21
PvcsAccessDeleteGroupUser on page 29
PvcsAccessRenameGroupGroup on page 46
PvcsAccessRenameGroupUser on page 47

PvcsAccessDeleteGroupUser

This function deletes a user from a group member list. It requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessDeleteGroupUser(
   int DBHandle,     /* Input */
   unsigned char *group,/* Input */
   unsigned char *userName)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
groupName	Pointer to a string that contains the name of the group.
userName	Pointer to a string that contains the name of the user to delete.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_INVALID_PARAMETER
    PVCS_E_UNKNOWN_USER
    PVCS_E_UNKNOWN_GROUP
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
   &DBHandle);

/* Delete user from group */
status = PvcsAccessDeleteGroupUser(
   DBHandle,/* Database handle */
   groupName,/* Name of group *
   userName);/* Name of user to delete */

if (!status)
   printf("Deleted user \"%s\" from group \"%s\".\n",
            userName, groupName);

/* Close the access control database */
status = PvcsAccessCloseDB(DBHandle);
```

Related Functions

PvcsAccessAddGroupGroup on page 20
PvcsAccessAddGroupUser on page 21
PvcsAccessDeleteGroupGroup on page 28
PvcsAccessRenameGroupGroup on page 46
PvcsAccessRenameGroupUser on page 47

PvcsAccessDeletePrivilege

This function deletes a privilege and requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessDeletePrivilege(
   int DBHandle,     /* Input */
   unsigned char *privilegeName)/* Input */
```

Parameters

DBHandle	Handle from **PvcsAccessOpenDB**.
privilegeName	Pointer to a string that contains the name of the privilege to delete.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_PRIVILEGE

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
   &DBHandle);

/* Delete the privilege */
status = PvcsAccessDeletePrivilege(
   DBHandle,/* Database handle */
   privilegeName);/* Name of privilege */
if (!status)
   printf("Deleted \"%s\".\n", privilegeName);

/* Close the access control database */
status = PvcsAccessCloseDB(DBHandle);
```

Related Functions

PvcsAccessDefinePrivilege on page 24
PvcsAccessEnumeratePrivilege on page 35
PvcsAccessEnumeratePrivileges on page 36
PvcsAccessQueryPrivilege on page 42
PvcsAccessRenamePrivilege on page 48

PvcsAccessDeleteUser

This function deletes a user from the access control database and requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessDeleteUser(
   int DBHandle,     /* Input */
   unsigned char *user)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
user	Pointer to a string that contains the name of the user to delete.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_USER
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *userName = 'DAVEE';

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
   &DBHandle);

/* Delete the user */
status = PvcsAccessDeleteUser(
   DBHandle,/* Database handle */
   userName);/* Name of user */
if (!status)
   printf("Deleted \"%s\".\n", userName);

/* Close the access control database */
status = PvcsAccessCloseDB(DBHandle);
```

Related Functions

PvcsAccessDefineUser on page 25
PvcsAccessEnumerateUserGroups on page 37
PvcsAccessEnumerateUsers on page 39
PvcsAccessQueryUser on page 43
PvcsAccessRenameUser on page 49

PvcsAccessEnumerateGroupGroups

This function returns a list of all groups within a group registered in the access control database. It requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessEnumerateGroupGroups(
   int DBHandle,     /* Input */
   unsigned char *group,/* Input */
   unsigned char *groupList,/* Output */
   unsigned groupListLen)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
group	Pointer to a string that contains the name of a group.
groupList	Pointer to a buffer that receives the group list. Each group is null-terminated, and the end-of-buffer is marked by an additional null character. If no users are present, Developer's Toolkit returns a null string.
groupListLen	Length of the buffer.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_BUFFER_OVERFLOW

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* Initialize configuration settings */
rc = PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open access database */
rc = PvcsAccessOpenDB(szAccessDatabase, PVCS_ACCOPEN_RDONLY,
   &hAccessDatabase);

/*
 * PvcsAccessEnumerateGroupGroups: get a list of all
 * groups inside given group.
 */

szGroupGroups = (PVCS_PUCHAR)malloc(256);
rc = PvcsAccessEnumerateGroupGroups(
   hAccessDatabase,
   szGroup,
   szGroupGroups,
   256);

/* Print result buffer */
    if (!rc)
    {
    printf ("groups within group %s:\n",szGroup);
    p = szGroupGroups;
    while (p && *p)
    {
            printf ("     %s\n",p);
            p = p + strlen(p) + 1;
    }
    }

/* Close access database */
    rc = PvcsAccessCloseDB(hAccessDatabase);
```

Related Functions

PvcsAccessDefineGroup on page 23
PvcsAccessDeleteGroup on page 27
PvcsAccessEnumerateGroups on page 32
PvcsAccessEnumerateGroupUsers on page 34
PvcsAccessQueryGroup on page 41
PvcsAccessRenameGroup on page 45

PvcsAccessEnumerateGroups

This function returns a list of all groups registered in the access control database. It requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessEnumerateGroups(
   int DBHandle,    /* Input */
```

```
   unsigned char *groupList,/* Output */
   unsigned groupListLen)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
groupList	Pointer to a buffer that receives the group list. Each group is null-terminated, and the end-of-buffer is marked by an additional null character. If no users are present, Developer's Toolkit returns a null string.
groupListLen	Length of the buffer.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_BUFFER_OVERFLOW

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *groupList;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
    &DBHandle);

/* Allocate buffer to hold database information */
groupList = (unsigned char *)malloc(1024);

/* Get a list of all groups registered in the database */
status = PvcsAccessEnumerateGroups(
   DBHandle,/* Database handle */
   groupList,/* Buffer receiving group list */
   1024);   /* Length of buffer */
if (!status) {
   printf('Groups:\n');
   print_list(groupList);
   }

/* Close the access control database */
PvcsAccessCloseDB(DBHandle);
free(groupList);
```

Related Functions

PvcsAccessDefineGroup on page 23
PvcsAccessDeleteGroup on page 27
PvcsAccessEnumerateGroupGroups on page 31
PvcsAccessEnumerateGroupUsers on page 34
PvcsAccessQueryGroup on page 41
PvcsAccessRenameGroup on page 45

PvcsAccessEnumerateGroupUsers

This function returns a list of all users that belong to a group. It requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessEnumerateGroupUsers(
   int DBHandle,     /* Input */
   unsigned char *group,/* Input */
   unsigned char *userList,/* Output */
   unsigned userListLen)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
group	Pointer to a string that contains the name of a group.
userList	Pointer to a buffer that receives the user list. Each user is null-terminated, and the end-of-buffer is marked by an additional null character. If no users are present, then Developer's Toolkit returns a null string.
userListLen	Length of the buffer.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_GROUP
PVCS_E_BUFFER_OVERFLOW

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *groupName = "SOFTDEV";
char *userList;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
    &DBHandle);

/* Allocate buffer to hold database information */
userList = (unsigned char *)malloc(1024);

/* Get a list of all users belonging to specified group */
status = PvcsAccessEnumerateGroupUsers(
   DBHandle,/* Database handle */
   groupName,/* Name of group */
   userList,/* Buffer receiving user list */
   1024);   /* Length of buffer */
if (!status) {
   printf("The following users belong to group \"%s\":\n",
            groupName);
   print_list(userList);
   }
```

```
/* Close the access control database */
PvcsAccessCloseDB(DBHandle);
free(userList);
```

Related Functions

PvcsAccessDefineGroup on page 23
PvcsAccessDeleteGroup on page 27
PvcsAccessEnumerateGroupGroups on page 31
PvcsAccessEnumerateGroups on page 32
PvcsAccessQueryGroup on page 41
PvcsAccessRenameGroup on page 45

PvcsAccessEnumeratePrivilege

This function returns a list of promotion groups assigned to a privilege. It requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessEnumeratePrivilege(
   int DBHandle,     /* Input */
   unsigned char *privName,/* Input */
   unsigned char *promoGroupList,/* Output */
   unsigned char *promoGroupListLen./* Input */
   PVCS_FLAGS flags)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
privName	The name of the privilege to enumerate.
promoGroupList	Pointer to a buffer that receives the promotion group list. Each promotion group is null-terminated, and the end-of-buffer is marked by an additional null character. If no promotion groups have been assigned, then Developer's Toolkit returns a null string.
promoGroupListLen	Length of the buffer.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_PRIVILEGE

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *privilegeName = "SoftDevPriv";
char *promoGroupList;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
    &DBHandle);
```

```
/* Allocate buffer to hold promotion group information */
promoGroupList = (unsigned char *)malloc(4096);

/* Get list of all promo groups associated with privilege */
status = PvcsAccessEnumeratePrivilege(
   DBHandle,/* Database handle */
   privilegeName,/* Name of privilege to enumerate */
   promoGroupList,/* Buffer receiving promo group list */
   4096);   /* Length of buffer */

if (!status) {
   printf("Promotion Groups:\n");
   print_list(promoGroupList);
   }

/* Close the access control database */
PvcsAccessCloseDB(DBHandle);
free(promoGroupList);
```

Related Functions

PvcsAccessDefinePrivilege on page 24
PvcsAccessDeletePrivilege on page 29
PvcsAccessEnumeratePrivileges on page 36
PvcsAccessQueryPrivilege on page 42
PvcsAccessRenamePrivilege on page 48

PvcsAccessEnumeratePrivileges

This function returns a list of privilege names from the access control database and requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessEnumeratePrivileges(
   int DBHandle,    /* Input */
   char *privilegeList,/* Output */
   unsigned *privilegeListLen,/* Input */
   PVCS_FLAGS flags)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
privilegeList	Pointer to a buffer that receives the privilege list. Each privilege is null-terminated, and the end-of-buffer is marked by an additional null character. If no privileges are present, Developer's Toolkit returns a null string.
privilegeListLen	Length of the buffer.
flags	Bit field that controls how the function operates. Values include: ■ PVCS_ACCENUM_BASE_PRIV Returns the list of base privileges. This is the default setting. ■ PVCS_ACCENUM_USER_DEFINED_PRIV Returns the list of custom privileges.

- PVCS_ACCENUM_COMPOSITE_PRIV
  Returns the list of composite privileges.
- PVCS_ACCENUM_GUI_PRIV
  Returns the list of GUI-specific menu item privileges.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_BUFFER_OVERFLOW

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *privilegeList;
int flags = PVCS_ACCENUM_BASE_PRIV;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE, &DBHandle);

/* Allocate buffer to hold database information */
privilegeList = (unsigned char *)malloc(2048);

/* Get a list of all base privileges */
status = PvcsAccessEnumeratePrivileges(
   DBHandle,/* Database handle */
   privilegeList,/* Buf receiving privilege list */
   2048,    /* Length of buffer */
   flags);  /* Type of privileges to return */

if (!status) {
    printf("Privileges:\n");
    print_list(privilegeList);
    }

/* Close the access control database */
PvcsAccessCloseDB(DBHandle);
free(privilegeList);
```

Related Functions

PvcsAccessDefinePrivilege on page 24
PvcsAccessDeletePrivilege on page 29
PvcsAccessEnumeratePrivilege on page 35
PvcsAccessQueryPrivilege on page 42
PvcsAccessRenamePrivilege on page 48

PvcsAccessEnumerateUserGroups

This function returns a list of all groups to which a specified user belongs. It requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessEnumerateUserGroups(
```

```
   int DBHandle,     /* Input */
   unsigned char *user,/* Input */
   unsigned char *groupList,/* Output */
   unsigned listLen)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
user	Pointer to a string that contains the name of the user.
groupList	Pointer to a buffer that receives the group list. Each group is null-terminated, and the end-of-buffer is marked by an additional null character. If no users are present, then Developer's Toolkit returns a null string.
listLen	Length of the buffer.

Return Values The return value is zero if the function is successful. Other values may be:

```
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_USER
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *userName = "DAVEE";
char *groupList;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
    &DBHandle);

/* Allocate buffer to hold database information */
groupList = (unsigned char *)malloc(1024);

/* Get list of groups to which the specified user belongs */
status = PvcsAccessEnumerateUserGroups(
   DBHandle,/* Database handle */
   userName,/* Name of user */
   groupList,/* Buf receiving group list */
   1024);   /* Length of buffer */

if (!status) {
   printf("User \"%s\" belongs to the following groups:\n",
   userName);
   print_list(groupList);
}

/* Close the access control database */
PvcsAccessCloseDB(DBHandle);
free(groupList);
```

Related Functions
PvcsAccessDefineUser on page 25
PvcsAccessDeleteUser on page 30
PvcsAccessEnumerateUsers on page 39
PvcsAccessQueryUser on page 43
PvcsAccessRenameUser on page 49

PvcsAccessEnumerateUsers

This function returns a list of all users registered in the access control database. It requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessEnumerateUsers(
   int DBHandle,     /* Input */
   unsigned char *userList,/* Output */
   unsigned listLen)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
userList	Pointer to a buffer that receives the user list. Each user is null-terminated, and the end-of-buffer is marked by an additional null character. If no users are present, then Developer's Toolkit returns a null string.
listLen	Length of the buffer.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *userList;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
    &DBHandle);

/* Allocate buffer to hold database information */
userList = (unsigned char *)malloc(1024);

/* Get a list of all users registered in the database */
status = PvcsAccessEnumerateUsers(
   DBHandle,/* Database handle */
   userList,/* Buf receiving user list */
   1024);   /* Length of buffer */
if (!status) {
   printf("Users:\n");
   print_list(userList);
```

```
   }

/* Close the access control database */
PvcsAccessCloseDB(DBHandle);
free(userList);
```

Related Functions
PvcsAccessDefineUser on page 25
PvcsAccessDeleteUser on page 30
PvcsAccessEnumerateUserGroups on page 37
PvcsAccessQueryUser on page 43
PvcsAccessRenameUser on page 49

PvcsAccessOpenDB

This function opens the access control database.

Syntax

```
PvcsAccessOpenDB(
   unsigned char *path,/* Input */
        int mode,    /* Input */
        int *pDBHandle)/* Output */
```

Parameters

path Pointer to a string that contains the name of the access control database.

mode Integer that specifies the type of access to the access control database. Values include:

- PVCS_ACCOPEN_RDONLY
  Opens the existing access control database for read-only access.
- PVCS_ACCOPEN_UPDATE
  Opens the existing access control database for read-write access. All other authorized users are locked out during this function.
- PVCS_ACCOPEN_CREATE
  Creates a new access control database. If a database exists already, it is overwritten. All other authorized users are locked out during this function.

pDBHandle Pointer to a variable that receives the access control database handle. This handle is used in other Serena PVCS Professional Suite services.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_FILE_NOT_FOUND
PVCS_ACCOPEN_RDONLY
PVCS_ACCOPEN_UPDATE
PVCS_E_CANT_OPEN_ACCESSDB
PVCS_E_INVALID_PARAMETER
PVCS_E_NO_HANDLES
PVCS_E_FILE_BUSY
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
```

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB(
   "access.db",/* Name of access control db */
   PVCS_ACCOPEN_UPDATE,/* Type of access to database */
   &DBHandle);/* Returned database handle */

/* Close the access control database */
status = PvcsAccessCloseDB(
   DBHandle);/* Database handle */
```

PvcsAccessQueryGroup

This function returns information about a group (privileges) and requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessQueryGroup(
   int DBHandle,     /* Input */
   unsigned char *groupName,/* Input */
   unsigned char *privilegeList,/* Output */
   unsigned privilegeListLen)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
groupName	Pointer to a string that contains the name of the group.
privilegeList	Pointer to a buffer that receives the privilege list. Each privilege is null-terminated, and the end-of-buffer is marked by an additional null character.
privilegeListLen	Length of the buffer.

Return Values

The return value is zero if the function is successful. Other values may be:

PVCS_E_INVALID PARAMETER
PVCS_E_BUFFER_OVERFLOW
PVCS_E_UNKNOWN_GROUP

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *group = "SOFTDEV";
char *privilegeList;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
    &DBHandle);
```

```
/* Allocate buffers to hold database information */
privilegeList = (unsigned char *)malloc(4096);

/* Get access control information for a specific group */
status = PvcsAccessQueryGroup(
   DBHandle,/* Database handle */
   group,   /* Name of group */
   privilegeList,/* Buf to receive privilege info */
   4096);   /* Length of buffer */

if (!status) {
   printf("Group:\n  %s\n", group);
   printf("Privileges:\n");
   print_list(privilegeList);
   }

/* Close the access control database */
PvcsAccessCloseDB(DBHandle);
free(privilegeList);
```

Related Functions

PvcsAccessDefineGroup on page 23
PvcsAccessDeleteGroup on page 27
PvcsAccessEnumerateGroups on page 32
PvcsAccessEnumerateGroupUsers on page 34
PvcsAccessRenameGroup on page 45

PvcsAccessQueryPrivilege

This function returns the list of privileges that make up a composite privilege. It requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessQueryPrivilege(
   int DBHandle,     /* Input */
   unsigned char *privName,/* Input */
   unsigned char *privList,/* Output */
   unsigned privListLen)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
privName	Pointer to a string that contains the name of the privilege.
privList	Pointer to a buffer that receives the privilege list. Each privilege is null-terminated, and the end-of-buffer is marked by an additional null character.
privListLen	Length of the buffer.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID PARAMETER
PVCS_E_UNKNOWN_PRIVILEGE

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *privilegeName = "SoftDevPriv";
char *privilegeList;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
    &DBHandle);

/* Allocate buffer to hold database information */
privilegeList = (unsigned char *)malloc(4096);

/* Get a list of all base privileges making up a given privilege */
status = PvcsAccessQueryPrivilege(
   DBHandle,/* Database handle */
   privilegeName,/* Name of privilege to enumerate */
   privilegeList,/* Buffer receiving privilege list */
   4096);   /* Length of buffer */

if (!status) {
   printf("Privileges:\n");
   print_list(privilegeList);
   }

/* Close the access control database */
PvcsAccessCloseDB(DBHandle);
free(privilegeList);
```

Related Functions

PvcsAccessDefinePrivilege on page 24
PvcsAccessDeletePrivilege on page 29
PvcsAccessEnumeratePrivilege on page 35
PvcsAccessEnumeratePrivileges on page 36
PvcsAccessRenamePrivilege on page 48

PvcsAccessQueryUser

This function returns information about a specific user (password, privilege, and expiration date). This function requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessQueryUser(
   int DBHandle,     /* Input */
   unsigned char *userName,/* Input */
   unsigned char *password,/* Output */
   unsigned passwordLen,/* Input */
   unsigned char *privilegeList,/* Output */
   unsigned privilegeListLen,/* Input */
   char *expDate,    /* Output */
   unsigned expDateLen)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
userName	Pointer to a string containing the user name.
password	Pointer to a buffer that receives the user password. If no password is present, then Developer's Toolkit returns a null string.
passwordLen	Length of the buffer.
privilegeList	Pointer to a buffer that receives the privilege list. Each privilege is null-terminated, and the end-of-buffer is marked by an additional null character.
privilegeListLen	Length of the buffer.
expDate	Pointer to a buffer that receives the user expiration data. The date format is: `mm dd yyyy hh:mm:ss` If no expiration date is present, then Developer's Toolkit returns a null string.
expDateLen	Length of the buffer.

Return Values The return value is zero if the function is successful. Other values may be:
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_USER

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *user = "DAVEE";
char *password;
char *privilegeList;
char *expDate;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE, &DBHandle);

/* Allocate buffers to hold database information */
password = (unsigned char *)malloc(32);
privilegeList = (unsigned char *)malloc(4096);
expDate = (unsigned char *)malloc(32);

/* Get access control information for a specific user */
status = PvcsAccessQueryUser(
   DBHandle,/* Database handle */
   user,    /* Name of user */
   password,/* Buf to receive user password */
   32,      /* Length of buffer */
   privilegeList,/* Buf to receive privilege info */
   4096,    /* Length of buffer */
   expDate, /* Buf to user expiration date */
```

```
   32);     /* Length of buffer */

if (!status) {
   printf("User:\n  %s\n", user);
   printf("Password:\n  %s\n",
            strlen(password) ? password : "<none>");
   printf("Privileges:\n");
   print_list(privilegeList);
   printf("Expiration date:\n  %s\n",
            strlen(expDate) ? expDate : "<none>");
   }

/* Close the access control database */
PvcsAccessCloseDB(DBHandle);
free(password);
free(privilegeList);
free(expDate);
```

Related Functions
PvcsAccessDefineUser on page 25
PvcsAccessDeleteUser on page 30
PvcsAccessEnumerateUsers on page 39
PvcsAccessEnumerateUserGroups on page 37
PvcsAccessRenameUser on page 49

PvcsAccessRenameGroup

This function renames a group and requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessRenameGroup(
   int DBHandle,     /* Input */
   unsigned char *oldGroup,/* Input */
   unsigned char *newGroup)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
oldGroup	Pointer to a string that contains the original name of the group.
newGroup	Pointer to a string that contains the new name of the group.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_GROUP
PVCS_E_USER_EXISTS

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *oldName = "SOFTDEV";
char *newName = "SOFTWARE_DEVELOPMENT";

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);
```

```
/* Open the access control database */
status = PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
    &DBHandle);

status = PvcsAccessRenameGroup(
   DBHandle,/* Database handle */
   oldName, /* Original name of group */
   newName);/* New name of group */
if (!status)
   printf("Renamed \"%s\" to \"%s\".\n", oldName, newName);
/* Close the access control database */
status = PvcsAccessCloseDB(DBHandle);
```

Related Functions

PvcsAccessDefineGroup on page 23
PvcsAccessDeleteGroup on page 27
PvcsAccessEnumerateGroupGroups on page 31
PvcsAccessEnumerateGroups on page 32
PvcsAccessEnumerateGroupUsers on page 34
PvcsAccessQueryGroup on page 41

PvcsAccessRenameGroupGroup

This function renames group members in a group. It requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessRenameGroupGroup(
   int DBHandle,     /* Input */
   unsigned char *group./* Input */
   unsigned char *oldGroupMember,/* Input */
   unsigned char *newGroupMember)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
group	Pointer to a string that contains the name of the group.
oldGroupMember	Pointer to a string that contains the original name of the group member.
newGroupMember	Pointer to a string that contains the new name of the group member.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_GROUP
PVCS_E_GROUP_EXISTS

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* Initialize configuration settings */
rc = PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open access database */
rc = PvcsAccessOpenDB(szAccessDatabase, PVCS_ACCOPEN_RDONLY,
```

```
   &hAccessDatabase);

/*
 * PvcsAccessEnumerateGroupGroups: get a list of all
 * groups inside given group.
 */
szGroupGroups = (PVCS_PUCHAR)malloc(256);
rc = PvcsAccessEnumerateGroupGroups(
   hAccessDatabase,
   szGroup,
   szGroupGroups,
   256);

/* print result buffer */
if (!rc)
{
   printf("Groups within group %s:\n",szGroup);
   {
            p = szGroupGroups;
            while (p && *p)
            {
            printf("    %s\n",p);
            p = p + strlen(p) + 1;
            }
   }
}

/* Close access control database */
rc = PvcsAccessCloseDB(hAccessDatabase);
free(szGroupGroups);
```

Related Functions
PvcsAccessAddGroupGroup on page 20
PvcsAccessAddGroupUser on page 21
PvcsAccessDeleteGroupGroup on page 28
PvcsAccessDeleteGroupUser on page 29
PvcsAccessRenameGroupUser on page 47

PvcsAccessRenameGroupUser

This function changes the name of a user in a group member list. It requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessRenameGroupUser(
   int DBHandle,     /* Input */
   unsigned char *group,/* Input */
   unsigned char *oldUserMember,/* Input */
   unsigned char *newUserMember)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
group	Pointer to a string that contains the name of the group.

oldUserMember	Pointer to a string that contains the original name of the user.
newUserMember	Pointer to a string that contains the new name of the user.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_USER
PVCS_E_UNKNOWN_GROUP
PVCS_E_USER_EXISTS
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *groupName = "SOFTDEV";
char *oldUserName = "DAVEE";
char *newUserName = "FREDF";

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
   &DBHandle);

/* Change the name of user within group */
status = PvcsAccessRenameGroupUser(
   DBHandle,/* Database handle */
   groupName,/* Name of group */
   oldUserName,/* Original name of user */
   newUserName);/* New name of user */

if (!status)
   printf("Renamed user \"%s\" in group \"%s\" to \"%s\".\n",
            oldUserName, groupName, newUserName);

/* Close the access control database */
status = PvcsAccessCloseDB(DBHandle);
```

Related Functions

PvcsAccessAddGroupGroup on page 20
PvcsAccessAddGroupUser on page 21
PvcsAccessRenameGroupGroup on page 46
PvcsAccessDeleteGroupGroup on page 28
PvcsAccessDeleteGroupUser on page 29

PvcsAccessRenamePrivilege

This function renames a custom privilege. It requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessRenamePrivilege(
   int DBHandle,    /* Input */
   unsigned char *oldName,/* Input */
   unsigned char *newName)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsAccessOpenDB**.
oldName	Pointer to a string that contains the original name of the privilege.
newName	Pointer to a string that contains the new name of the privilege.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_PRIVILEGE
PVCS_E_PRIVILEGE_EXISTS
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *oldName = "SOFTDEV";
char *newName = "SOFTWARE_DEVELOPMENT";

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
   &DBHandle);

status = PvcsAccessRenamePrivilege(
   DBHandle,/* Database handle */
   oldName, /* Original name of privilege */
   newName);/* New name of privilege */
if (!status)
   printf("Renamed \"%s\" to \"%s\".\n", oldName, newName);

/* Close the access control database */
status = PvcsAccessCloseDB(DBHandle);
```

Related Functions

PvcsAccessQueryPrivilege on page 42
PvcsAccessEnumeratePrivilege on page 35
PvcsAccessEnumeratePrivileges on page 36
PvcsAccessDefinePrivilege on page 24
PvcsAccessDeletePrivilege on page 29

PvcsAccessRenameUser

This function changes a user name and requires the ViewAccessDB privilege.

Syntax

```
PvcsAccessRenameUser(
   int DBHandle,     /* Input */
   unsigned char *oldUser,/* Input */
   unsigned char *newUser)/* Input */
```

Parameters

DBHandle	Handle returned by **PvcsOpenAccessDB**.
oldUser	Pointer to a string that contains the original name of the user.
newUser	Pointer to a string that contains the new name of the user.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_USER
PVCS_E_USER_EXISTS
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
int DBHandle;
char *oldName = "DAVEE";
char *newName = "CLARKR";

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open the access control database */
status = PvcsAccessOpenDB("access.db", PVCS_ACCOPEN_UPDATE,
   &DBHandle);

status = PvcsAccessRenameUser(
   DBHandle,/* Database handle */
   oldName, /* Original name of user */
   newName; /* New name of user */
if (!status)
   printf("Renamed \"%s\" to \"%s\".\n", oldName, newName);

/* Close the access control database */
status = PvcsAccessCloseDB(DBHandle);
```

Related Functions

PvcsAccessEnumerateUsers on page 39
PvcsAccessEnumerateUserGroups on page 37
PvcsAccessQueryUser on page 43
PvcsAccessDefineUser on page 25
PvcsAccessDeleteUser on page 30

PvcsAddAlias

This function adds an alias definition to the alias list. It is equivalent to the Alias directive.

Syntax

```
PvcsAddAlias(
   unsigned char *aliasName,/* Input */
   unsigned char *aliasValue)/* Input */
```

Parameters

`aliasName`	Pointer to the name of the alias to define.
`aliasValue`	Pointer to the character string that is assigned to *`aliasName`*.

Return Values This function returns zero if successful. Otherwise the value is `PVCS_E_INVALID_PARAMETER`. See Chapter 4, "Return Values" for descriptions of return values.

Example

```
char *name = "SourceFiles";
char *value = "alpha.c beta.c gamma.c";
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Add alias definition to alias list */
status = PvcsAddAlias(
   name,    /* Alias name */
   value);  /* Alias value */

/* Display newly defined alias */
if (!status)
   printf("Alias name \"%s\" is defined to be \"%s\".\n",
        name, value);
```

Related Function PvcsQueryAlias on page 138

Related Topic For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
How to use aliases	*Using Aliases*

PvcsAddPromoteTreeNode

This function adds a group to a promotion model by defining the relationship between each group and its parent group. This is equivalent to the Promote directive.

Syntax

```
PvcsAddPromoteTreeNode(
   unsigned char *fromGroup,/* Input */
   unsigned char *toGroup)/* Input */
```

Parameters

`fromGroup`	Name of promotion group.
`toGroup`	Name of the parent of group *`fromGroup`*.

Return Values The return value is zero if the function is successful. Other values may be:
`PVCS_E_INVALID_PARAMETER`
`PVCS_E_INVALID_PROMO`

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

To define a complete promotion model, call this function once for every promotion group, except the highest group in the model, called the *production group*. Then call **PvcsVerifyPromoTree** to verify that the model has exactly one production group and that every group except the production group promotes to exactly one parent group.

Example

```
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Build the following promotion model */
/*                                     */
/*              PRODUCTION             */
/*                  |                  */
/*                 QA                  */
/*          _______|_______           */
/*         |       |       |          */
/*       DEV1     DEV2     DEV3       */
/*                                     */
status = PvcsAddPromoteTreeNode("QA", "PRODUCTION");
if (!status)
   status = PvcsAddPromoteTreeNode("DEV1", "QA");
if (!status)
   status = PvcsAddPromoteTreeNode("DEV2", "QA");
if (!status)
   status = PvcsAddPromoteTreeNode("DEV3", "QA");

/* Verify the promotion model */
if (!status)
   status = PvcsVerifyPromoTree();
if (!status)
   printf("Promotion model defined successfully.\n");
```

Related Functions

PvcsGetPromoParent on page 99
PvcsGroupToRevision on page 111
PvcsPromoteRevision on page 131
PvcsVerifyPromoTree on page 177
PvcsVerifyPromoTreeNodeExist on page 178

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Promotion models	*Promotion*
Defining a promotion group	*Promote directive*
Promoting revisions	*VPROMOTE command*

PvcsAssignPromoGroup

This function assigns or deletes a promotion group. It is equivalent to the VCS -G command.

Syntax

```
PvcsAssignPromoGroup(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *group,/* Input */
   unsigned char *revarg,/* Input */
   PVCS_FLAGS grpFlags)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to a string that contains the name of the archive or workfile. Required only if the archive is not open.
group	Pointer to a string that contains the promotion group that you want to revise.
revarg	Pointer to a string that contains a revision number, version label, or promotion group. If the version label or promotion group begins with a number, precede it with a backslash (\). A null parameter defaults to the tip revision on the trunk.
grpFlags	Bit field that controls how the function operates. Values include: ■ PVCS_AG_REPLACE_GROUP Moves an existing promotion group to a new revision. ■ PVCS_AG_NO_REPLACE_GROUP Does not move an existing promotion group. This is the default. ■ PVCS_AG_RENAME_GROUP Renames the promotion group using *revarg* as the old name and *group* as the new name. ■ PVCS_AG_DELETE_GROUP Deletes the group.

Return Values

The return value is zero if the function is successful. Other values may be:

PVCS_E_GROUP_EXISTS
PVCS_E_LOCKED_REVISION
PVCS_E_ARCHIVE_EMPTY
PVCS_E_NO_GROUP
PVCS_E_NO_REVISION
PVCS_E_NO_VERSION
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* Initialize configuration settings */
rc = PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* PvcsAssignPromoGroup: */
rc = PvcsAssignPromoGroup(
   ARCHIVEHANDLE_NOT_OPEN,
```

```
   szArchive,
   szGroup,
   szRevision,
   flags);

/*
 * Call PvcsGetRevisionInfo2 to get a list of the promotion
 * groups assigned to the given revision.
 */
bufSize = 256;
pRevInfo2Buffer = (PVCS_PUCHAR)malloc(bufSize);
rc = PvcsGetRevisionInfo2(
   ARCHIVEHANDLE_NOT_OPEN,
   szArchive,
   szRevision,
   &szAuthor,
   &szVersions,
   &szPromoGroups,
   &szLockers,
   &szDescription,
   pRevInfo2Buffer,
   bufSize,
   NULL);

if (!rc)
{
   printf("    Rev: %s\n",szRevision);
   printf("        PromoGroups:   ");
   print_string_list(szPromoGroups);
}
if (pRevInfo2Buffer) free(pRevInfo2Buffer);
```

Related Functions
PvcsGroupToRevision on page 111
PvcsGetPromoParent on page 99
PvcsPromoteRevision on page 131

PvcsAssignVersion

This function assigns or modifies a version label. It is equivalent to the VCS -V command.

Syntax

```
PvcsAssignVersion(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *version,/* Input */
   unsigned char *revarg,/* Input */
   PVCS_FLAGS verFlags)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of the archive or workfile. This parameter is required only if the archive is not open.

`version` Pointer to a string that contains the version label to modify.

`revarg` Pointer to a string that contains a revision number or version label. If the version label begins with a number, precede it with a backslash (\). A null parameter defaults to the tip revision on the trunk.

`verFlags` Bit field that controls the operation of this function. Values include:

- `PVCS_AV_REPLACE_VERS`
  Moves an existing version label to a new revision.
- `PVCS_AV_NO_REPLACE_VERS`
  Does not move an existing version label. This is the default.
- `PVCS_AV_BRANCH_VERS`
  Makes *`version`* a floating version label on the same branch as *`revarg`*.
- `PVCS_AV_RENAME_VERS`
  Renames the version label, using *`revarg`* as the old name and *`version`* as the new name.
- `PVCS_AV_DELETE_VERS`
  Deletes the version label.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_BAD_ARCHIVE_HANDLE
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
PVCS_E_NO_REVISION
PVCS_E_NO_VERSION
PVCS_E_USER_ABORTED
PVCS_E_VERSION_EXISTS
PVCS_E_VERSION_EXISTS
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *`fileName`* refers to a workfile, the program infers the archive name. If *`fileName`* contains wildcards, the program expands it according to the usual Version Manager rules.
- If *`revarg`* is a revision number, *`version`* is assigned to it. If *`revarg`* is a version label, *`version`* is assigned to the revision number to which *`revarg`* is currently assigned.
- If you specify `PVCS_AV_DELETE_VERSION` for *`verFlags`* to delete a version label, *`revarg`* is ignored.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Assign a version label */
PvcsAssignVersion(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",/* Name of archive */
   "Beta Release 1.0",/* Version label to assign */
   NULL,    /* Assign to tip revision */
   PVCS_AV_NO_REPLACE_VERS);/* Don't move existing label */
```

Related Functions
PvcsOpenArchive on page 129
PvcsPutRevision on page 134
PvcsVersionToRevision on page 179

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Version labels	*Version Labels*
Assigning version labels	*PUT command* *VCS command*
Specifying archives and workfiles	*File Specification*
Wildcard file expansion	*Wildcards*

PvcsCancelUpdate

This function cancels an update to an open archive.

Syntax
```
PvcsCancelUpdate(
   ARCHIVEHANDLE hArchive)/* Input */
```

Parameter

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.

Return Value This function returns zero if successful. Otherwise it returns PVCS_E_INVALID_PARAMETER. See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- You must open the archive with **PvcsOpenArchive** before calling this function.
- Changes to an archive are not saved until you call **PvcsCloseArchive** or **PvcsCloseAll**. To cancel changes made since you opened the archive, call **PvcsCancelUpdate**. You do not need to call **PvcsCloseArchive** after calling **PvcsCancelUpdate**.
- If the NoArchiveWork directive is in effect, you cannot use this function to cancel changes to an archive. When NoArchiveWork is in effect, Version Manager applies changes directly to the archive, rather than to a copy of it.

  When ArchiveWork is in effect, Version Manager applies updates to a working copy of the archive. When you call **PvcsCancelUpdate**, Version Manager deletes the working copy and leaves the archive in its original state.

  You can query the ArchiveWork directive from your program by using the *useArchiveWork* value of the CONFIG structure after calling **PvcsQueryConfiguration**. For details on the CONFIG data structure, see Chapter 5, "Data Structures"

Example
```
ARCHIVEHANDLE handle;
int status;
```

```
/* Initialize configuration structure */
config = (CONFIG *)malloc(sizeof(CONFIG));
PvcsQueryConfiguration(NULL, NULL, 0,
   PVCS_CONFIG_OVERWRITE);

/* Open archive for update */
PvcsOpenArchive("foo.c_v", NULL, 0, NULL,
   0, PVCS_OPEN_UPDATE, &handle);

/* Perform an operation on archive */
status = PvcsUnLockRevision(handle, NULL, NULL, "DAVEE");

/* Cancel update and close archive */
if (status)
   PvcsCancelUpdate(handle);
```

Related Functions
PvcsCloseAll on page 62
PvcsCloseArchive on page 63
PvcsOpenArchive on page 129

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Using the ArchiveWork directive	*ArchiveWork directive*

PvcsChangeAccessList

This function modifies an archive access list. It is equivalent to the VCS -A command.

Syntax

```
PvcsChangeAccessList(
   ARCHIVEHANDLE archiveHandle,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *accessList,/* Input */
   PVCS_FLAGS accessListflags)/* Input */
```

Parameters

`archiveHandle`	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify `ARCHIVEHANDLE_NOT_OPEN`.
`fileName`	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open. The file name can contain wildcards.
`accessList`	Pointer to a string that contains a list of user IDs separated by spaces or commas.
`accessListflags`	Bit field that controls the operation of this function. Values include: ■ `PVCS_ACCLIST_ADD` Adds name(s) to an existing list.

- PVCS_ACCLIST_DELETE
  Deletes name(s) from an existing list.
- PVCS_ACCLIST_REPLACE
  Replaces an existing list.
- PVCS_ACCLIST_CLEAR
  Deletes all names from a list.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_ACCESS_DENIED
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_BAD_ARCHIVE_HANDLE
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Add "SOFTDEV" to AccessList of foo.c_v */
PvcsChangeAccessList(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",
   "SOFTDEV",
   PVCS_ACCLIST_ADD);

/* Delete "QA" from AccessList of foo.c_v */
PvcsChangeAccessList(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",
   "QA",
   PVCS_ACCLIST_DELETE);

/* Replace AccessList of foo.c_v with DEVGROUP1,DEVGROUP2,DEVGROUP3 */
PvcsChangeAccessList(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",
   "DEVGROUP1,DEVGROUP2,DEVGROUP3",
   PVCS_ACCLIST_REPLACE);

/* Delete all names from AccessList of foo.c_v */
PvcsChangeAccessList(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",
   NULL,
   PVCS_ACCLIST_CLEAR);
```

Related Functions

PvcsChangeArchiveInfo on page 59
PvcsGetArchiveInfo on page 85
PvcsQueryArchiveAccess on page 139

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Changing an access list	*VCS command*
Controlling user access	*Access Control*

PvcsChangeArchiveInfo

This function changes archive header information. It is equivalent to certain VCS command options.

Syntax

```
PvcsChangeArchiveInfo(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned short attributes,/* Input */
   unsigned short reclen,/* Input */
   unsigned short renum_start_col,/* Input */
   unsigned short renum_end_col,/* Input */
   long renum_start_val,/* Input */
   long renum_step_val,/* Input */
   unsigned char *workfile,/* Input */
   unsigned char *owner,/* Input */
   unsigned char *access,/* Input */
   unsigned char *cmt_str,/* Input */
   unsigned char *newline,/* Input */
   unsigned char *diffmask)/* Input */
```

Parameters

hArchive — Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.

fileName — Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open.

attributes — Bit field with values that toggle archive attributes. Values include:

- PVCS_ATTR_CHK_LOCK
  Enables or disables lock checking. This flag is equivalent to the VCS +|-PL command.
- PVCS_ATTR_CMPRS_DELTA
  Enables or disables delta compression. This flag is equivalent to the VCS +|-PD command.
- PVCS_ATTR_CMPRS_TEXT
  Enables or disables compression of workfile images. This flag is equivalent to the VCS +|-PC command.
- PVCS_ATTR_EXCL_LOCK
  Enables or disables exclusive locking. This flag is equivalent to the VCS +|-PE command.

- PVCS_ATTR_EXP_KEYS
  Enables or disables keyword expansion. This flag is equivalent to the VCS +|-PK command.
- PVCS_ATTR_TRANSLATE
  Enables or disables file translation. This flag is equivalent to the VCS +|-PT command.
- PVCS_ATTR_WRT_PROT
  Enables or disables write protection of archives. This flag is equivalent to the VCS +|-PW command.

reclen	Workfile record length. Use 0 to indicate that the workfile does not contain fixed-length records. This parameter is equivalent to the VCS -XRecordLength command.
renum_start_col	Starting column number of the line number field. Use 0 to disable renumbering. This parameter is equivalent to the *column_start* parameter to the VCS -XRenumber command.
renum_end_col	Ending column number of the line number field. This parameter is equivalent to the *column_end* parameter to the VCS -XRenumber command.
renum_start_val	The number for the first line in the workfile. This parameter is equivalent to the *start* parameter to the VCS -XRenumber command.
renum_step_val	The value used to increment the line numbers. This parameter is equivalent to the *number* parameter to the VCS -XRenumber command.
workfile	Pointer to a string that contains the name of the workfile. The program uses this name as the workfile name when it only knows the archive name. This parameter is equivalent to the VCS -W command.
owner	Pointer to a string that contains the ID of the archive's owner. This parameter is equivalent to the VCS -O command.
access	Pointer to a string that contains the archive access list, which is a comma-separated list of user IDs or group names. This parameter is equivalent to the VCS -A command.
cmt_str	Pointer to a string that contains the comment prefix string, used when expanding the Log keyword. This parameter is equivalent to the VCS -EC command.
newline	Pointer to a string that contains the end-of-line characters used when expanding the Log keyword. The default is `\r\n`. This parameter is equivalent to the VCS -EN command.
diffmask	Pointer to a string that contains the column mask specification used when computing differences. A null parameter indicates that columns are not masked when computing differences. This parameter is equivalent to the VCS -X ColumnMask command.

Return Values The return value is zero if the function is successful. Other values may be:
PVCS_E_ACCESS_VIOLATION

```
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Consideration

If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.

Example

```
#define ARCHIVEINFO_PAD 256
ARCHIVEINFO *archinfo;
char newAccessList[256];
unsigned short newAttributes;
char *fileName = "foo.c_v";

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Obtain archive information */
archinfo = (ARCHIVEINFO *)malloc(sizeof(ARCHIVEINFO) +
    ARCHIVEINFO_PAD);
PvcsGetArchiveInfo(ARCHIVEHANDLE_NOT_OPEN,
   Filename,
   archinfo,
   sizeof(ARCHIVEINFO) + ARCHIVEINFO_PAD);

/* Add names to the AccessList */
if (strlen(archinfo->info + archinfo->access))
   strcpy(newAccessList, archinfo->info + archinfo->access);
strcat(newAccessList, "SOFTDEV,TECHPUBS,QA");

/* Turn off keyword expansion */
newAttributes = archinfo->attributes &= ~PVCS_ATTR_EXP_KEYS;

/* Change archive information */
PvcsChangeArchiveInfo(ARCHIVEHANDLE_NOT_OPEN,
   "Filename",
   newAttributes,
   archinfo->reclen,
   archinfo->renum_start_col,
   archinfo->renum_end_col,
   archinfo->renum_start_val,
   archinfo->renum_step_val,
   archinfo->info + archinfo->workfile,
   archinfo->info + archinfo->owner,
     newAccessList,
   archinfo->info + archinfo->cmt_str,
   archinfo->info + archinfo->newline,
   archinfo->info + archinfo->diffmask);
```

Related Functions

PvcsChangeAccessList on page 57
PvcsGetArchiveInfo on page 85

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide.*

For information about...	See...
Changing archive information	*VCS command*
Archive attributes	*Archive Attributes*
Specifying archive and workfile names	*File Specification*

PvcsCloseAll

This function closes all open archives and deletes semaphores and temporary files.

Syntax
```
PvcsCloseAll(
   void)
```

Parameters There are no parameters to this function.

Special Consideration You can call this function as a cleanup function when your program exits. It does not return an error if no archives are open.

Return Value This function returns zero.

Example
```
ARCHIVEHANDLE handle[3];
char workfile[32];
char archive[32];
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open several archives for update */
PvcsOpenArchive("alpha.c_v", NULL, 0, NULL, 0, PVCS_OPEN_UPDATE,
   &handle[0]);
PvcsOpenArchive("beta.c_v", NULL, 0, NULL, 0, PVCS_OPEN_UPDATE,
   &handle[1]);
PvcsOpenArchive("gamma.c_v", NULL, 0, NULL, 0, PVCS_OPEN_UPDATE,
   &handle[2]);

/* Close all archives */
status = PvcsCloseAll();
if (!status)
   printf("Closed all archives\n");
```

Related Functions
PvcsCancelUpdate on page 56
PvcsCloseArchive on page 63
PvcsOpenArchive on page 129

PvcsCloseArchive

This function closes an archive and frees resources associated with it. It also closes the archive semaphore, if it exists.

Syntax

```
PvcsCloseArchive(
   ARCHIVEHANDLE hArchive)/* Input */
```

Parameter

hArchive	Handle returned by **PvcsOpenArchive**.

Return Values

The return value is zero if the function is successful. Otherwise it returns PVCS_E_INVALID_PARAMETER. See Chapter 4, "Return Values" for descriptions of return values.

Example

```
ARCHIVEHANDLE handle;
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open archive for update */
PvcsOpenArchive(
   "foo.c_v",/* Name of archive or workfile */
   NULL,    /* Buf receiving name of workfile */
   0,       /* Length of workfile buffer */
   NULL,    /* Buf receiving name of archive */
   0,       /* Length of archive buffer */
   PVCS_OPEN_UPDATE,/* Open archive for modification */
   &handle);/* Returned archive handle */

if (!status)
   printf("Opened \"%s\"\n", archive);

/* Close archive */
status = PvcsCloseArchive(handle);
if (!status)
   printf("Closed \"%s\"\n", archive);
```

Related Functions

PvcsCancelUpdate on page 56
PvcsCloseAll on page 62
PvcsOpenArchive on page 129

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Archive semaphores	*Semaphores*

PvcsComputeArchiveName

This function computes the name of an archive for a specified workfile.

Syntax

```
PvcsComputeArchiveName(
   unsigned char *workfileName,/* Input */
   unsigned char *archiveName,/* Output */
   unsigned short archiveNameLen)/* Input */
```

Parameters

workfileName	Pointer to the name of a workfile. The file name must be valid, but does not have to exist.
archiveName	Pointer to the buffer to receive the name of the archive. The buffer must be long enough to contain the full path name.
archiveNameLen	Length of the archive buffer.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_BAD_FILENAME
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- This function uses the current value of the ArchiveSuffix directive to compute the archive name.
- This function neither requires nor verifies the existence of the workfile or archive.
- Use **PvcsOpenArchive** to perform the inverse operation: computing the workfile name for a known archive.

Example

```
char archiveName[128];
char workfileName[128] = "foo.c";
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Compute name of archive for "foo.c" */
status = PvcsComputeArchiveName(workfileName,
   archiveName,
   sizeof(archiveName));

/* Display name of archive */
if (!status)
   printf("Archive name for \"%s\" is \"%s\"\n", workfileName,
            archiveName);
```

Related Function

PvcsOpenArchive on page 129

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
How workfile names are translated into archive names	*Suffix Translation*
Specifying archive translation	*ArchiveSuffix directive*
Specifying archive and workfile names	*File Specification*

PvcsCreateArchive

This function creates a new archive that contains no revisions. It is equivalent to the VCS -I command.

Syntax
```
PvcsCreateArchive(
   unsigned char *fileName,/* Input */
   unsigned char *workfileName,/* Input */
   unsigned char *fileDesc,/* Input */
   unsigned char *accessList,/* Input */
   unsigned char *owner,/* Input */
   PVCS_FLAGS createFlags)/* Input */
```

Parameters

fileName	Pointer to the archive name. See *Special Considerations* for the effects of various types of path specifications.
workfileName	Pointer to the name of the workfile, which is stored in the archive. The name may contain a path. This parameter is equivalent to the *file_name* parameter to the VCS -I command.
fileDesc	Pointer to a string that contains the description of the workfile. The string may contain *@fileName*, which causes the program to read the description from *fileName*. A null parameter causes the program to prompt the user for a description. This parameter is equivalent to the -T option used with the VCS -I command.
accessList	Pointer to a string that contains the comma- or space-separated archive access list. A null parameter indicates an empty access list. This parameter is equivalent to the -A option used with the VCS -I command.
owner	Pointer to a string that contains the owner of the archive. If the parameter is null, the program uses the current user ID. This parameter is equivalent to the -O option used with the VCS -I command.
createFlags	Bit field that controls the operation of this function. Values include: ■ PVCS_CREATE_IGN_PATH Ignores the path of the workfile and creates the archive in the first path in the VCSDir list. This flag is equivalent to having the IgnorePath directive in effect when you use the VCS -I command.

- PVCS_CREATE_NO_OVERWRITE

  Does not overwrite the archive if it exists. This flag is equivalent to the -N option used with the VCS -I command.

- PVCS_CREATE_OVERWRITE

  Overwrites the archive if it exists. This flag is equivalent to the -Y option used with the VCS -I command.

- PVCS_CREATE_WRITABLE

  Makes the archive writable. The default is read-only. This flag is equivalent to having the NoWriteProtect directive in effect when you use the VCS -I command.

Return Values

The return value is zero if the function is successful. Other values may be:

PVCS_E_ACCESS_DENIED
PVCS_E_ACCESS_VIOLATION
PVCS_E_ALREADY_EXISTS
PVCS_E_BAD_FILENAME
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
PVCS_E_USER_ABORTED

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- Archive attributes, such as Compress and ExpandKeywords, are taken from the directives in the CONFIG structure. If the CONFIG structure is null, the Developer's Toolkit uses the default configuration. For details on the CONFIG structure, see Chapter 5, "Data Structures"
- There are several ways to specify the *fileName* parameter:
  - **Fully qualified path**. The program creates the archive in the specified location. The specified drive and directory must exist.
  - **Path without a file name**. The program derives the archive name from the workfile name and creates the archive in the specified location. The specified drive and directory must exist.
  - **Null**. The program derives the archive name from the workfile name and creates the archive in the first path specified for the VCSDir directive. If the VCSDir list is empty, it creates the archive in the current directory.
- If *fileName* is null and *workfileName* contains a path, the program creates the archive in the workfile directory, unless you specify the flag PVCS_CREATE_IGN_PATH.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Create archive */
PvcsCreateArchive(
   NULL,    /* Path only; force deriv of */
            /* Archive name from workfile */
   "foo.c", /* Workfile name */
   "@proj1.msg",/* Use msg file for desc */
   NULL,    /* Empty access list */
   "DAVEE", /* Archive owner */
   PVCS_CREATE_NO_OVERWRITE);/* Don't overwrite existing */
```

Related Functions
PvcsOpenArchive on page 129
PvcsPutRevision on page 134

Related Topics
For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Creating archives	*VCS command*
Directives that are related to this function	*IgnorePath directive* *WriteProtect directive* *VCSDir directive*

PvcsDeleteRevision

This function deletes revisions from an archive. It is equivalent to the VDEL command.

Syntax

```
PvcsDeleteRevision(
   ARCHIVEHANDLE archiveHandle,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *revisionRange,/* Input */
   PVCS_FLAGS flags)/* Input */
```

Parameters

archiveHandle	Use ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to a string that contains the archive or workfile name.
revisionRange	Pointer to a string specifying revisions to delete. To specify a revision range, separate the revisions with an asterisk (*). Append a plus sign (+) to the range to delete branches as well as the trunk. If you do not use a plus sign and there are branches, the operation fails with the error PVCS_E_BRANCH_REVISION. The range can be open-ended on either end.
flags	Not currently used.

Return Values
The return value is zero if the function is successful. Other values may be:
PVCS_E_BRANCH_REVISION
PVCS_E_GROUP_EXISTS
PVCS_E_INVALID_PARAMETER
PVCS_E_LOCKED_REVISION

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- This function does not currently support archive handles. You can specify the name of the archive to be modified, and **PvcsDeleteRevision** opens and closes the archive. If the archive is open and you specify an archive handle, the function returns PVCS_E_INVALID_PARAMETER.

- You cannot delete a revision if it contains locks or promotion groups. If a revision contains locks or promotion groups, the function does not delete any revision and returns PVCS_LOCKED_REVISION or PVCS_GROUP_EXISTS.

Example

```
char *version = "Beta Release 1.0";
char range[32];
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Build a ver range that specifies all revisions up to, but */
/* excluding the ver label "Beta Release 1.0" */
sprintf(range, "%s%s%s", "*", version, "-1");

/* Delete a revision range */
status = PvcsDeleteRevision(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",/* Archive name */
   range,   /* Version range */
   0);      /* Not currently used */

if (!status)
   printf("Deleted revisions prior to vers. label \"%s\".\n",
            version);
```

Related Functions

PvcsGetRevisionInfo on page 103
PvcsUnLockRevision on page 172

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Deleting revisions	*VDEL command*
Specifying revision ranges	*Revision Specification*

PvcsDiagnosticEnable

This function enables the Developer's Toolkit debug trace to write diagnostic information to a file that you specify. Call this function once to enable the debug trace for all future Developer's Toolkit functions.

Call this function with the PVCS_DIAG_CLOSE flag to end the debug trace and close the file.

Syntax

```
PvcsdiagnosticEnable(
   unsigned char *diagnosticFile,/* Input */
   unsigned short diagnosticFileMode,/* Input */
   PVCS_FLAGS flags) /* Input */
```

Parameters

diagnosticFile	Pointer to a string that names the output file to which Developer's Toolkit writes diagnostic information. The type of file access is determined by the *diagnosticFileMode* parameter.
diagnosticFileMode	Integer that specifies the type of access to the output file. It is also used to close the output file. Values include: ■ PVCS_DIAG_MODE_OVERWRITE Developer's Toolkit opens the output file in overwrite mode. If the diagnostic information file exists, then it is overwritten. This is the default. ■ PVCS_DIAG_MODE_APPEND Developer's Toolkit opens the output file in append mode. If the file exists, then information is appended to the end of the file. Without this flag, the file is overwritten. ■ PVCS_DIAG_MODE_CLOSE Developer's Toolkit closes the output file and stops all diagnostic reporting.

flags	Bit field that controls how this function operates. Values include PVCS_DIAG_CONFIG, which enables configuration file tracing using **PvcsQueryConfiguration** to echo a copy of each line it reads from the configuration file. It is equivalent to the -#1 command-line flag. It must be enabled before you call **PvcsQueryConfiguration**. Output is:
	`(nesting_level) config_file_name [line_number]: config_file_line`
	
	Example:
	`Contents of vcs.cfg:`
	`Login = vcsid`
	`VcsID = cyne`
	`AccessDB = access.db`
	`AccessList = cyne, lauraf, skipr`
	`Include = local.cfg`
	`Contents of local.cfg:`
	`NoDeleteWork`
	`NoExpandKeywords`
	`NoArchiveWork`
	`Output:`
	`( 1) vcs.cfg[ 1]: Login = vcsid`
	`( 1) vcs.cfg[ 3]: Vcsid = cyne`
	`( 1) vcs.cfg[ 4]: AccessDB = access.db`
	`( 1) vcs.cfg[ 5]: AccessList = cyne, lauraf, skipr`
	`( 1) vcs.cfg[ 6]: Include = local.cfg`
	`( 2) local.cfg[ 1]: NoDeleteWork`
	`( 2) local.cfg[ 2]: NoExpandKeywords`
	`( 2) local.cfg[ 3]: NoArchiveWork`

- PVCS_DIAG_FUNCTION

Enables tracing. Every Developer's Toolkit function writes the name of the function and the value of the function parameters. There
is no command-line equivalent. Output is:
`function_name,date_and_time_when_called`

Example:

```
PvcsDiagnosticEnable, 18 Feb 1995
14:59:40
  1: "tkdig.out"
  2: 1
  3: 7
PvcsQueryConfiguration, 19 Feb 1995
14:59:40
  1:0000:0000
  2:0000:0000
  3:0
  4:0
PvcsCreateArchive, 18 Feb 1994 14:59:42
  1: 0000:0000
  2: "foo.c"
  3: "message text"
  4: 0000:0000
  5: "DAVEE"
  6: 2
PvcsOpenArchive, 19 Feb 1995 14:59:42
  1: "foo.c_v"
  2: 002F:195C
  3: 32
  4: 002F:197C
  5: 32
  6: 2
  7: 002F:1958
```

```
PvcsPutRevision, 18 Feb 1995 14:59:42
 1: 0
 2: "DAVEE"
 3: 0000:0000
 4: 0000:0000
 5: 0000:0000
 6: "Fixed a killer bug."
 7: 0000:0000
 8: "Beta Release 1.0"
 9: 0000:0000
10: 0000:0000
11: 0
12: 32
PvcsCloseArchive, 18 Feb 1995 14:59:42
 1: 0
PvcsDiagnostic Enable, 18 Feb 1995
14:59:42
 1: 0000:0000
 2: 3
 3: 0
```

- PVCS_DIAG_ACCESS_CONTROL

Enables access control privilege tracing. It writes access control information about the current user. This flag must be enabled before you call any Developer's Toolkit functions that open archives. It is equivalent to the -#400 command-line flag. Output is:

```
owner:archive_owner,access_list:
archive_access_list
Opening access_control_database_name
User name:user_name
privilege:[base_priv_1, base_priv_2,...,
base_priv_n]
```

Example:

```
Contents of access control database named
"access.db":
USER davee (Put, Get, Lock, Unlock,
AddVersion, ViewArchive)

Output:
owner: davee, access list:
Opening access.db
User name: davee
privilege:
[LT,LN,UN,bl,GT,GN,PT,PB,sb,ca,co,cp,cc,cw,m
w,mc,AV,dv,mv,il,dt,dn,vd,VH,VR,va,pr,ag,mg,
dg]
```

Base Privilege Codes

Every base privilege is represented by a two-character mnemonic value. Values in lower case are turned off for the current user; values in upper case are active for the current user (as shown in the example above).

Base Privilege Codes

AV	AddVersion	LN	LockNonTip
AG	AddGroup	LT	LockTip
BL	BreakLock	MC	ModifyChangeDescription
CA	ChangeAccessList	MG	ModifyGroup
CC	ChangeCommentDelimiter	MV	ModifyVersion
CO	ChangeOwner	MW	ModifyWorkfileDescription
CP	ChangeProtection	PB	PutBranch
CW	ChangeWorkfileName	PR	Promote
DG	DeleteGroup	PT	PutTrunk
DN	DeleteRevNonTip	SB	StartBranch
DT	DeleteRevTip	UN	Unlock
DV	DeleteVersion	VA	ViewAccessDB
GN	GetNonTip	VD	ViewDelta
GT	GetTip	VH	ViewArchiveHeader
IL	InitArchive	VR	ViewArchiveRev

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_DIAG_FILE

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
ARCHIVEHANDLE handle;

/* Enable debug trace */
PvcsDiagnosticEnable(
   "diag.out",/* Diagnostic file name */
   PVCS_DIAG_MODE_OVERWRITE,/* Overwrite existing file */
   PVCS_DIAG_FUNCTION);/* Write toolkit func param */

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);
/* Create archive */
PvcsCreateArchive(NULL, "foo.c", "message text", NULL,
   "DAVEE", PVCS_CREATE_OVERWRITE);

/* Open archive for update */
PvcsOpenArchive("foo.c_v", NULL, 0, NULL,
   0, PVCS_OPEN_UPDATE, &handle);

/* Check in a revision */
PvcsPutRevision(handle, "DAVEE", NULL, NULL, NULL, "Fixed a
   bug.", NULL, "Beta Release 1.0", NULL, NULL, 0,
   PVCS_PUT_RELOCK);

/* Close archive */
PvcsCloseArchive(handle);
```

```
/* Disable diagnostic trace */
PvcsDiagnosticEnable(
   (char *)0,
   PVCS_DIAG_MODE_CLOSE,/* Close diagnostic file */
   0);
```

PvcsEndArchiveSearch

This function ends an archive search that was initiated by **PvcsFindFirstArchive** and frees memory associated with the file search.

Syntax

```
PvcsEndArchiveSearch(
   PVCSSEARCHHANDLE srchHandle)/* Input */
```

Parameter

srchHandle	Pointer to a search handle. This handle must have been initialized by a call to **PvcsFindFirstArchive**

Return Value

The return value is zero if the function is successful. Otherwise the value is PVCS_E_INVALID_PARAMETER. See Chapter 4, "Return Values" for descriptions of return values.

Example

```
PVCSSEARCHHANDLE srchHandle;
char archive[128];
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Find first file, set up search handle for next file */
status = PvcsFindFirstArchive(
   &srchHandle,/* Receives search hdl used on subsequent */
            /* calls to PvcsFindNextArchive */
   "*.c_v", /* File pattern to match */
   archive, /* Buffer receiving first matching name */
   sizeof(archive));    /* Length of buffer */

if (!status) {

   /* Iterate over all matching files */
            do {

   /* Perform an operation on the archive */
   PvcsAssignVersion(ARCHIVEHANDLE_NOT_OPEN, archive,
            "Beta Release 1.0", NULL, PVCS_AV_REPLACE_VERS);

   /* Obtain next file name */
   status = PvcsFindNextArchive(
            srchHandle,/* Search handle */
            archive,/* Buf receiving next matching name */
   sizeof(archive));/* Length of buffer */
```

```
   if ((status) && (status != PVCS_E_NO_MORE_FILES)) {
            printf("Error finding next archive.\n");
            break;
   }
            } while (status != PVCS_E_NO_MORE_FILES);

   /* End the search */
   PvcsEndArchiveSearch(srchHandle);
}
else
   printf("Error finding first archive.\n");
```

Related Functions
PvcsFindFirstArchive on page 79
PvcsFindNextArchive on page 81

PvcsExport

This function exports archive data to text files in the specified format. It duplicates the functionality of the VSQL program.

Syntax

```
PVCS_APIENTRY PvcsExport(
   unsigned char *DDLTimestamp,/* Input */
   unsigned char *DDLVarchar,/* Input */
   unsigned char *DDLInteger,/* Input */
   unsigned char *DDLFileName,/* Input */
   unsigned char *DMLFileName,/* Input */
   unsigned char *TimeFormat,/* Input */
   unsigned char *Delimiter,/* Input */
   unsigned char *EmbDelimiter,/* Input */
   unsigned char *Separator,/* Input */
   unsigned char *EmbSeparator,/* Input */
   unsigned char *LineCont,/* Input */
   unsigned char *EmbLineCont,/* Input */
   unsigned char *EmbNewLine,/* Input */
   unsigned char *SQLTerminator,/* Input */
   unsigned short CharMaxLen,/* Input */
   unsigned short LineMaxLen,/* Input */
   unsigned char *FileName,/* Input */
   PVCS_FLAGS exportFlags)/* Input */
```

Parameters

DDLTimestamp	Pointer to a string that contains the TimeStamp column type definition, depending on the requirements of your SQL interpreter. The default is `TIMESTAMP`. This parameter is equivalent to the VSQL -DT command.
DDLVarchar	Pointer to a string that contains the variable-length character string column type definition. The default is `VARCHAR(1500)`. This parameter is equivalent to the VSQL -DV command.

DDLInteger — Pointer to a string that contains the integer column type definition, depending on the requirements of your SQL interpreter. The default is `INTEGER`. This parameter is equivalent to the VSQL -DI command.

DDLFileName — Pointer to a string that contains the file name where the DDL statements are to be written. DDL file contains DROPTABLE, CREATETABLE, and CREATEVIEW commands to initiate tables for holding Version Manager archive data and views used for updating these tables. This parameter is equivalent to the VSQL -DL command.

DMLFileName — Pointer to a string that contains the file name where the DML statements are to be written. The DML statements will insert, update, and/or delete information from the tables created by the DDL statements.

TimeFormat — Pointer to a string that contains the format for the timestamp fields, or the fields specified by *DDLTimestamp*.

TimeFormat may contain the following special formatting strings:

- `yyyy`: A four-digit number representing the year portion of the date. You can also use `yy` to represent the last two digits of the year.
- `mm`: (First occurrence.) A two-digit number between 01 and 12 representing the month. If you enter more than two consecutive `m` characters, the Developer's Toolkit interprets them as an abbreviation of the month. For example, `mmm`=Dec; `mmmm`=Dece
- `dd`: A two-digit number between 01 and 31 (the maximum depends on the month and year) representing the day of the month.
- `hh`: A two-digit number between 00 and 24 representing the hour portion of the time (using a 24-hour clock).
- `mm`: (Second occurrence.) A two-digit number between 00 and 59 representing the minutes portion of the time.
- `ss`: A two-digit number from 00 to 59 representing the seconds portion of the time. The default is: `TimeFormat yyyy-mm-dd-hh.mm.ss`

This parameter is equivalent to the VSQL -FT command.

Delimiter — Pointer to a string that contains the string delimiter character. Most CSV formats are processed with the least conflict using double quotation marks (") as the string delimiter. The default, however, is a single quotation mark (') to accommodate the DML format.

To use a single quotation mark or double quotation mark, preface it with a backslash (\). To use a backslash, enter two backslashes. This parameter is equivalent to the VSQL -O command.

EmbDelimiter	Pointer to a string that contains the character that the Developer's Toolkit substitutes for *Delimiter* when it is embedded within a string. The default *Delimiter* is a single quotation mark. To use a single quotation mark or double quotation mark, preface it with a backslash. To use a backslash, enter two backslashes. This parameter is equivalent to the VSQL -E command. For example, if *Delimiter* is a single quotation mark, and you expect to use single quotation marks within strings, you can use backslash, double quotation mark (\") to ensure that the Developer's Toolkit won't confuse *Delimiter* with characters within strings.
Separator	Pointer to a string that contains the field separator character. The default Separator is a comma (,). This parameter is equivalent to the VSQL -G command.
EmbSeparator	Pointer to string that contains the character that the Developer's Toolkit substitutes for *Separator* when it is embedded in a field. This parameter is usually needed only in CSV format files if no *Delimiter* is used. This parameter is equivalent to the VSQL -I command. For example, if *Separator* is a comma, and you expect to use commas within strings, use a semicolon (;), or another non-conflicting character to change the treatment of commas within a field.
LineCont	Pointer to string that contains the line continuation character. The default is an empty string. This parameter is equivalent to the VSQL -J command.
EmbLineCont	Pointer to a string that contains the character that the Developer's Toolkit substitutes for *LineCont* when it is embedded in a string. The default is an empty string. This parameter is equivalent to the VSQL -L command.
EmbNewLine	Pointer to a string that contains the character that the Developer's Toolkit substitutes for the newline character when it is embedded in a string. By default, the Developer's Toolkit substitutes a space. If you don't specify this parameter, strings cannot contain newline characters. This parameter is equivalent to the VSQL -N command. For example, to preserve newline characters in strings, use one of the following values: "\n," "<lf>".
SQLTerminator	Pointer to a string that contains the SQL statement termination character. If your SQL interpreter requires no statement terminator, use a null string (" ") as the parameter value. The default is a semicolon (;). This parameter is equivalent to the VSQL -T command.
CharMaxLen	Maximum length of a variable-length character string, as named in the *DDLVarchar* parameter (above).
LineMaxLen	Maximum line length for output command files. The minimum line length is 64 characters. If you use a number smaller than 64, the line length will default to 64 characters. The default is an unlimited length. This parameter is equivalent to the VSQL -K command. For example, use 250 to limit the line length to 250 characters, including delimiters and modification characters.

FileName Pointer to a string that contains an archive file specification. The specification allows wildcards.

exportFlags Bit field that controls how the function operates. Values include:

- PVCS_EXPORT_SQLDML_FORMAT
- PVCS_EXPORT_CSV_FORMAT
- PVCS_EXPORT_CSV_FORMAT_HDRS

If PVCS_EXPORT_SQLDML_FORMAT is set, then the Developer's Toolkit uses the file name specified by *DMLFileName*. This file contains SQL DML format insert statements.
If none of the above mentioned flags is set, then the default is PVCS_EXPORT_SQLDML_FORMAT. This parameter is equivalent to the VSQL -MH, VSQL -MV, or VSQL -ML command.

- PVCS_EXPORT_GEN_VIEW_STMTS
  Generates DROPTABLE and DROPVIEW commands if used in conjunction with PVCS_EXPORT_GEN_DROP_STMTS. You can use this command to re-execute the command file. It deletes and recreates the old tables and views.
- PVCS_EXPORT_GEN_DROP_STMTS
  Generates DROP statements in the DDL command file if used in conjunction with PVCS_EXPORT_GEN_VIEW_STMTS. DROP commands delete old tables and views from the database before generating new ones. You can use this command if you need to rerun the DDL command file with the old tables still defined in the database. This command is ignored if you have not set PVCS_EXPORT_GEN_VIEW_STMTS.
- PVCS_EXPORT_GEN_UPDATE_STMTS
  Generates update statements. This command puts the Delete statements in a separate file with the name specified by *DMLFileName*, if used in conjunction with PVCS_EXPORT_CSV_FORMAT, PVCS_EXPORT_CSV_FORMAT_HDRS, or PVCS_EXPORT_SQLDML_FORMAT.
  You can use this command to maintain incremental changes to an existing database of archives when the tables cannot be easily regenerated; or, where the number of new/updated rows is expected to be small compared to the total number of rows.
- PVCS_EXPORT_CONTINUE
  Use in conjunction with wildcard archive file name specifications.
- PVCS_EXPORT_SQLDML_FORMAT
  Generates SQL DML format insert statements. This flag is equivalent to the VSQL -ML command.

- PVCS_EXPORT_CSV_FORMAT
  Generates a set of CSV format files that contain data from specified archives. The Developer's Toolkit creates all script files; however, they may be empty. This flag is equivalent to the VSQL -MV command.
- PVCS_EXPORT_CSV_FORMAT_HDRS
  Generates a set of CSV format files that contain data from specified archives. The column names are the first record of each file. This flag is equivalent to the VSQL -MH command.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID PARAMETER
PVCS_E_DISK_FULL
PVCS_E_BAD_ARCHIVE
PVCS_E_ACCESS_VIOLATION
PVCS_E_ACCESS_DENIED

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- The *DDLTimeStamp*, *DDLVarchar*, and *DDLInteger* parameters will be the default if they are specified as null pointers. All other pointers must be are required and will return a PVCS_E_INVALID_PARAMETER error if they are null.
- Each invocation of **PvcsExport** will process one archive file specification (which may contain wildcards) and create new output files. To process additional wildcard specifications, use the PVCS_EXPORT_CONTINUE flag.

IMPORTANT! PvcsExport must always be called first if the PVCS_EXPORT_CONTINUE flag is not set.

- The status returned is for the last archive processed. For this reason, if the caller wants to process warning-level errors (anything but PVCS_E_DISK_FULL), then the caller should do wildcard expansion using **PvcsFindFirstArchive**/ **PvcsFindNextArchive**, and then call **PvcsExport( )** for each archive.
- Archive IDs are calculated from the current time. To avoid duplicate IDs, do not run multiple instances of PVCSexport in quick succession.

Example

The following example demonstrates with pseudo-code how to create one set of output files that contains archive information for all .C and .H archives:

```
for each fileSpecification in *.c_v *.h_v
    PvcsExport(..., fileSpecification, exportFlags);
   exportFlags |= PVCS_EXPORT_CONTINUE;
    endfor
char *fileName = "foo.c_v";
PVCS_FLAGSexportFlags =PVCS_EXPORT_CSV_FORMAT_HDRS |
   PVCS_EXPORT_GEN_VIEW_STMTS | PVCS_EXPORT_GEN_DROP_STMTS |
   PVCS_EXPORT_GEN_UPDATE_STMTS;
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Export archive information for use by relational database */
```

```
status = PvcsExport(
   NULL,    /* I: -DT, defs to "TIMESTAMP" */
   NULL,    /* I: -DV, defs to "VARCHAR(1500)" */
   NULL,    /* I: -DI, defs to "INTEGER" */
   "ddl",   /* I: -DL + DDLGenerate */
   "vsql",  /* I: -MH, -MV, -ML */
   "YYYY-MM-DD-HH.MM.SS",
   "\"",    /* I: -O  String delimiter */
   "\'\'",  /* I: -E  ...when embedded */
   ",",     /* I: -G  Field separator */
   ",",     /* I: -I  ...when embedded */
   "",      /* I: -J  Line continuation */
   "",      /* I: -L  ...when embedded */
   " ",     /* I: -N  ...when embedded */
   ";",     /* I: -T  SQL Termination  */
   1500,    /* I: -DV */
   10000,   /* I: -K  */
   fileName,/* I: Archive file specification */
   exportFlags);/* I: */

if (!status)
   printf("Exported information from \"%s\".\n",fileName);
```

PvcsFindFirstArchive

This function searches for files matching a file specification. It returns the name of the first matching file and a search handle that **PvcsFindNextArchive** can use to locate additional files that match the wildcard specification. This is the function used by Version Manager to find files that are specified on the command line.

IMPORTANT! This function finds files that may or may not be archives.

Syntax

```
PvcsFindFirstArchive(
   PVCSSEARCHHANDLE *srchHandle,/* Output */
   unsigned char *filePattern,/* Input */
   unsigned char *resultBuf,/* Output */
   unsigned short bufLen)/* Input */
```

Parameters

srchHandle	Pointer to a search handle. This handle is used on subsequent calls to **PvcsFindNextArchive**.
filePattern	Pointer to a file pattern, which may include a path and wildcards.
resultBuf	Pointer to a buffer to receive the matching file name. The buffer must be long enough to contain the full path name.
bufLen	Length of the result buffer.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_ARCHIVE_NOT_FOUND

```
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- The function searches the directories specified by the VCSDir directive. If VCSDir is null, it searches the current directory.

  You can also specify a directory as part of the *filePattern* parameter to ignore the VCSDir directive. For example:

  ```
  I:\source\vmgui\logfiles\*.c_v
  ```

- After processing data from **PvcsFindFirstArchive**, call **PvcsEndArchiveSearch** to free memory allocated by **PvcsFindFirstArchive**.

Example

```
PVCSSEARCHHANDLE srchHandle;
char archive[128];
int status;
int is_archive;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Find first file, set up search handle for next file */
status = PvcsFindFirstArchive(
   &srchHandle,/* Receives search hdl used on subsequent */
            /* Calls to PvcsFindNextArchive */
   "*.c_v", /* File pattern to match */
   archive, /* Buffer receiving first matching name */
   sizeof(archive));    /* Length of buffer */

if (!status)
{
   /* Iterate over all matching files */
   do
   {
   /* Only do operation if file is Pvcs Archive */
   status = PvcsIsArchive (archive, &is_archive);
   if (!status && is_archive)
   {
            /* Perform an operation on the archive */
            PvcsAssignVersion(ARCHIVEHANDLE_NOT_OPEN,
            archive, "Beta Release 1.0", NULL,
            PVCS_AV_REPLACE_VERS);
    }
   /* Obtain next file name */
   status = PvcsFindNextArchive(
            srchHandle,/* Search handle */
            archive,/* Buf receiving next matching name */
            sizeof(archive));  /* Length of buffer */
   if ((status) && (status != PVCS_E_NO_MORE_FILES))
            {
            printf("Error finding next archive.\n");
            break;
            }
   } while (status != PVCS_E_NO_MORE_FILES);
```

```
   /* End the search */
   PvcsEndArchiveSearch(srchHandle);
}
else
   printf("Error finding first archive.\n");
```

Related Functions
PvcsEndArchiveSearch on page 73
PvcsFindNextArchive on page 81

Related Topics
For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Wildcard expansion	*Wildcards*
How Version Manager finds archives	*Search Paths*
Specifying archive locations	*VCSDir directive*

PvcsFindNextArchive

This function searches a directory for the next file that matches a file specification. It returns the name of the next matching file.

Syntax

```
PvcsFindNextArchive(
   PVCSSEARCHHANDLE srchHandle,/* Input */
   unsigned char *resultBuf,/* Output */
   unsigned short bufLen)/* Input */
```

Parameters

srchHandle	A search handle.
resultBuf	Pointer to a buffer to receive the matching file name. The buffer must be long enough to contain the full path name.
bufLen	Length of the result buffer.

Return Values
The return value is zero if the function is successful. Other values may be:
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER
PVCS_E_NO_MORE_FILES

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations
- The search handle must have been initialized by a prior call to **PvcsFindFirstArchive**.
- The function searches the directories specified by the VCSDir directive. If VCSDir is null, it searches the current directory.

Example

```
PVCSSEARCHHANDLE srchHandle;
char archive[128];
int status;

/* Initialize configuration settings */
```

```
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Find first file, set up search handle for next file */
status = PvcsFindFirstArchive(
   &srchHandle, /* Receives search hdl used on subsequent */
            /* calls to PvcsFindNextArchive */
   "*.c_v", /* File pattern to match */
   archive, /* Buffer receiving first matching name */
   sizeof(archive));     /* Length of buffer */

if (!status) {

   /* Iterate over all matching files */
            do {

            /* Perform an operation on the archive */
            PvcsAssignVersion(ARCHIVEHANDLE_NOT_OPEN,
            archive, "Beta Release 1.0", NULL,
            PVCS_AV_REPLACE_VERS);

   /* Obtain next file name */
   status = PvcsFindNextArchive(
            srchHandle,/* Search handle */
            archive,/* Buf receiving next matching name */
            sizeof(archive));/* Length of buffer */
   if ((status) && (status != PVCS_E_NO_MORE_FILES)) {
   printf("Error finding next archive.\n");
   break;}
   } while (status != PVCS_E_NO_MORE_FILES);

   /* End the search */
   PvcsEndArchiveSearch(srchHandle);
}
else
   printf("Error finding first archive.\n");
```

Related Functions
PvcsEndArchiveSearch on page 73
PvcsFindFirstArchive on page 79

PvcsGenDeltaFile

This function compares two files or revisions and generates a delta file, which contains editing commands used by the Version Manager REGEN command to generate the target file from the reference file. This function is equivalent to the VDIFF -D command.

Syntax

```
PvcsGenDeltaFile(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *refFile,/* Input */
   unsigned char *refRev,/* Input */
   unsigned char *tgtFile,/* Input */
   unsigned char *tgtRev,/* Input */
   unsigned short deltaFormat,/* Input */
   unsigned char *deltaCfg,/* Input */
```

```
unsigned char *outFile,/* Input */
unsigned short recordLength,/* Input */
unsigned char *columnMask,/* Input */
PVCS_FLAGS flags) /* Input */
```

Parameters

hArchive Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.

refFile Pointer to a string that contains the name of the reference file. Specify a null pointer if the reference file is an archive whose archive handle is specified in the *hArchive* parameter.

refRev Pointer to a string that contains the revision number or version label of the reference file. If the version label begins with a number, precede it with a backslash (\). If the reference file is not an archive, or if it is the tip revision, specify a null pointer.

tgtFile Pointer to a string that contains the name of the target file. Specify a null pointer if the target file is an archive with an archive handle specified in the *hArchive* parameter.

tgtRev Pointer to a string that contains the revision number or version label of the target file. If the version label begins with a number, precede it with a backslash (\). If the target file is not an archive, or if it is the tip revision, specify a null pointer.

deltaFormat Integer that specifies the type of delta file to create. Values include:

- PVCS_DELTA_BINARY
  Generates a binary delta file in Version Manager format. This flag is equivalent to the VDIFF -DS command.
- PVCS_DELTA_LIBRARIAN
  Generates a CA-LIBRARIAN delta file. This flag is equivalent to the VDIFF -DL command.
- PVCS_DELTA_PANVALET
  Generates a CA-PANVALET delta file. This flag is equivalent to the VDIFF -DP command.
- PVCS_DELTA_PREDEFINED
  Uses the file specified by the *deltaCfg* parameter.
- PVCS_DELTA_UNKNOWN
  Uses the file specified by the *deltaCfg* parameter. If the *deltaCfg* parameter is null, this flag defaults to the PVCS_DELTA_BINARY *deltaformat*.

deltaCfg Pointer to a string that contains the name of a file that contains delta configuration directives. The PVCS_DELTA_PREDEFINED *deltaformat* must be specified to generate the delta file in this format. If this parameter is null, the function generates a delta file in the format specified by the *deltaFormat* parameter. This parameter is equivalent to the VDIFF -D command.

outFile Pointer to a string that contains the name of a file to which this function writes the delta file. The specified file will be overwritten if it exists. If this pointer is null, the report is sent to standard output. This parameter is equivalent to the VDIFF -XO command.

recordLength	Integer specifying the record length of the files to be compared. Specify 0 to indicate that the files use newline characters to indicate the ends of lines. This parameter is equivalent to the VDIFF -XRecordLength command.
columnMask	String specifying the columns to ignore when comparing files. This parameter is equivalent to the VDIFF -XColumnMask command.
flags	Bit field that controls how this function operates. Values include: ■ PVCS_DIFF_APPEND Appends the report to an existing file. ■ PVCS_DIFF_IGN_WHITE Ignores leading and trailing white space during comparison. This flag is equivalent to the VDIFF -B command. ■ PVCS_DIFF_NOMOVE Records moved text as deletions and insertions.

Return Values

The return value is zero if the function is successful. Other values may be:

PVCS_E_ACCESS_DENIED
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_BAD_FILENAME
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
PVCS_E_NO_REVISION
PVCS_E_USER_ABORTED

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
char *outFile = "foo.dlt";
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Generate delta for differences between two revisions */
/* in an archive */
status = PvcsGenDeltaFile(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",/* Use archive as reference file */
   NULL,    /* Use tip as reference rev */
   "foo.c_v",/* Use archive as target file */
   "Beta Release 1.0",/* Use ver label as target rev */
   PVCS_DELTA_BINARY,/* Generate Version Manager delta */
   NULL,    /* Use format specified above */
   outFile, /* Delta output file */
   0,       /* Newlines are end of line */
   NULL,    /* No columnmask */
   PVCS_DIFF_IGN_WHITE);/* Ignore whitespace

if (!status)
   printf("Created delta in \"%s\".\n", outFile);
```

Related Functions

PvcsReportDifferences on page 165
PvcsTestDifferences on page 170

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Delta file format	*Delta Files*
Generating delta files	*VDIFF command*
Directives that affect delta generation	*Delta Delete directive* *Delta Insert directive* *Delta Replace directive* *Delta Seq directive*
Recreating a file from its delta file	*REGEN command*

PvcsGetArchiveInfo

This function returns archive header information. It is equivalent to the VLOG -B command.

Syntax

```
PvcsGetArchiveInfo(
   ARCHIVEHANDLE hArchive,/* Input  */
   unsigned char *fileName,/* Input */
   ARCHIVEINFO *archInfo,/* Output */
   unsigned short archInfoLen)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open.
archInfo	Pointer to the structure receiving archive information. The last element of this structure contains buffer information, which is a character array of variable length.
archInfoLen	Length of the information buffer.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_BUFFER_OVERFLOW
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- To get archive information in a format suitable for user reports, use **PvcsLog** instead.

- You must allocate the ARCHIVEINFO structure. It should be long enough to contain the string fields copied into the information buffer. The location of each string is given by an offset into information. There are seven strings, each with a maximum length of 256 bytes, so you should pad the structure with 1792 bytes when you allocate it.

  See the definition of the ARCHIVEINFO structure in Chapter 5, "Data Structures" for a description of the fields returned by this function.

- If you don't need any of the information string fields, you dont need to provide extra space for the information buffer. You will get a buffer overflow error, which you can ignore.

- If PVCS_E_BUFFER_OVERFLOW is returned, a string offset value of -1 indicates that the corresponding information string would not fit in the buffer.

Example

```
#define ARCHIVEINFO_PAD = 1792
ARCHIVEINFO *archinfo;
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Allocate buffer to hold archive information */
archinfo = (ARCHIVEINFO *)malloc(sizeof(ARCHIVEINFO) +
   ARCHIVEINFO_PAD);

/* Obtain archive information */
status = PvcsGetArchiveInfo(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",
   archinfo,
   sizeof(ARCHIVEINFO) + ARCHIVEINFO_PAD);

/* Display archive information */
if (!status) {
   printf("Number of revs:    %d\n", archinfo->revcnt);
   printf("Number of locks:   %d\n", archinfo->lockers);
   printf("CheckLock:         %s\n",
            (archinfo->attributes & PVCS_ATTR_CHK_LOCK) ?
            "on": "off");
   printf("WriteProtect:      %s\n",
            (archinfo->attributes & PVCS_ATTR_WRT_PROT) ? "on" : "off");
   printf("ExclusiveLock:     %s\n",
            (archinfo->attributes & PVCS_ATTR_EXCL_LOCK) ? "on" :
     "off");
   printf("ExpandKeywords:    %s\n",
            (archinfo->attributes & PVCS_ATTR_EXP_KEYS) ? "on" : "off");
   printf("Translate:         %s\n",
            (archinfo->attributes & PVCS_ATTR_TRANSLATE) ? "on" :
     "off");
   printf("CompressDelta:     %s\n",
            (archinfo->attributes & PVCS_ATTR_CMPRS_DELTA) ? "on" :
     "off");
   printf("CompressWorkImage: %s\n",
            (archinfo->attributes & PVCS_ATTR_CMPRS_TEXT) ? "on" :
     "off");
   printf("RecordLength:      %d\n", archinfo->reclen);
   printf("Created:           %d/%d/%d %d:%d:%d\n",
```

```
                archinfo->create_time.month,
                archinfo->create_time.day, archinfo->create_time.year,
                archinfo->create_time.hours, archinfo->create_time.minutes,
                archinfo->create_time.twosecs * 2);
        printf("Renum start col: %d\n", archinfo->renum_start_col);
        printf("Renum end col:   %d\n", archinfo->renum_end_col);
        printf("Renum start val: %ld\n", archinfo->renum_start_val);
        printf("Renum step val:  %ld\n", archinfo->renum_step_val);
        printf("Archive:      %s\n", archinfo->info + archinfo->archive);
        printf("Workfile:     %s\n", archinfo->info + archinfo->workfile);
        printf("Owner:        %s\n", archinfo->info + archinfo->owner);
        printf("AccessList:   %s\n", archinfo->info + archinfo->access);
        printf("CommentPrefix: %s\n", archinfo->info + archinfo->cmt_str);
        printf("NewLine:      %s\n", archinfo->info + archinfo->newline);
        printf("ColumnMask:   %s\n", archinfo->info + archinfo->diffmask);
```

Related Functions

PvcsChangeArchiveInfo on page 59
PvcsGetLockInfo on page 95
PvcsGetRevisionInfo on page 103
PvcsLog on page 119
PvcsOpenArchive on page 129

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Viewing archive information	*VLOG command*
Specifying archive and workfile names	*File Specification*

PvcsGetArchiveInfoVB1

This function, specifically for Visual Basic users, returns the most commonly used values from the archive header information.

Syntax

```
PvcsGetArchiveInfoVB1(
     ARCHIVEHANDLE  hArchive,/* Input */
     unsigned char  *fileName,/* Input */
     unsigned short *revcnt,/* Output */
     unsigned short *lockers,/* Output */
     unsigned char  *archive,/* Output */
     unsigned char  *workfile,/* Output */
     unsigned char  *owner,/* Output */
     unsigned char  *access,/* Output */
     unsigned char  *create_time,/* Output */
     unsigned short *attribute_chk_lock,/* Output */
     unsigned short *attribute_wrt_prot,/* Output */
     unsigned short *attribute_excl_lock,/* Output */
     unsigned short *attribute_exp_keys,/* Output */
     unsigned short *attribute_translate,/* Output */
     unsigned short *attribute_cmprs_delta,/* Output */
     unsigned short *attribute_cmprs_text,/* Output */
       PVCS_FLAGS flags)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of the archive or workfile. This parameter is required only if the archive is not open.
revcnt	Pointer to a string that receives the number of revisions in the archive.
lockers	Pointer to an unsigned short that receives the number of locks on the archive.
archive	Pointer to a string that receives the name of the archive.
workfile	Pointer to a string that receives the name of the workfile.
owner	Pointer to a string that receives the owner of the archive.
access	Pointer to a string that receives information about which users have access to the archive.
create_time	Pointer to a string that receives the time the archive was created.
attribute_chk_lock *attribute_wrt_prot* *attribute_excl_lock* *attribute_exp_keys* *attribute_translate* *attribute_cmprs_delta* *attribute_cmprs_text*	These parameters return a value of TRUE if the corresponding bit in the ARCHIVEINFO attributes variable is set.
flags	Bit field that controls the operation of this function. Values include: PVCS_ARCHINFO_DATETIME_FORMAT (0X0001) This flag causes the *create_time* string to be formatted with the currently configured date/time format. If this flag is not set, then the create_time string will be returned with the default format mm/dd/yyyy hh:mm:ss.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_ACCESS_VIOLATION
    PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_BUFFER_OVERFLOW
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Consideration

Any of the non-input parameters may be passed a NULL value if no return value for that parameter is desired.

Example

```
/* PvcsGetArchiveInfoVB1 example */

int         status = 0;
unsigned shortrevcnt;
unsigned shortlockers;
unsigned chararchive[256];
```

```
unsigned charworkfile[256];
unsigned charowner[64];
unsigned characcess[256];
unsigned charcreate_time[32];
unsigned shortattribute_chk_lock;
unsigned shortattribute_wrt_prot;
unsigned shortattribute_excl_lock;
unsigned shortattribute_exp_keys;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/*
 * PvcsGetArchiveInfoVB1: Get archive info into separate
 * variables instead of an ARCHIVEINFO structure. If you don't
 * want the information for a specific field, then pass NULL
 * (i.e. see the last three attribute parameters).
 */
status = PvcsGetArchiveInfoVB1(
    ARCHIVEHANDLE_NOT_OPEN, /* I: Archive handle */
    "foo.c_v",              /* I: Name of archive or workfile */
    &revcnt,                /* O: Number of revisions */
    &lockers,               /* O: Mumber of locks */
    archive,                /* O: Name of archive */
    workfile,              /* O: Name of workfile */
    owner,                 /* O: Name of archive owner */
    access,                /* O: Archive access list */
    create_time,           /* O: Text version of workfile
             creation time */
    &attribute_chk_lock,   /* O: Attrib item check lock on
           check in bit */
    &attribute_wrt_prot,   /* O: Attribute item archive write
                                 protection bit */
    &attribute_excl_lock,  /* O: Attribute item single lock
           bit */
    &attribute_exp_keys,   /* O: Attrib item expand keywords
           bit */
    NULL,   /* O: Attrib item do eol trans bit */
    NULL,   /* O: Attrib item compress deltas bit */
    NULL,   /* O: Attrib item compress full text
                    revisions bit */
    flags   /* I: Bit field */
   );
if (!status)
{
    printf("Archive name: %s\n",archive);
    printf("Number of revisions: %d\n",revcnt);
    printf("Number of locks: %d\n",lockers);
    printf("Workfile name: %s\n",workfile);
    printf("Owner: %s\n",owner);
    printf("Access list: %s\n",access);
    printf("Creation time: %s\n",create_time);
    printf("Check Lock attribute is:
           %s\n",attribute_chk_lock ? "on" : "off");
   printf("    Write Protect attribute is:
           %s\n",attribute_wrt_prot ? "on" : "off");
```

```
   printf("    Exclusive Lock attribute is:
           %s\n",attribute_excl_lock ? "on" : "off");
   printf("    Expand Keywords attribute is:
           %s\n",attribute_exp_keys ? "on" : "off");
}
```

Related Functions
PvcsGetArchiveInfo on page 85
PvcsGetArchiveInfoVB2 on page 90
PvcsGetLockInfoVB on page 97
PvcsGetRevisionInfoVB on page 107

PvcsGetArchiveInfoVB2

This function, specifically for Visual Basic users, returns the less commonly used values from the archive header information.

Syntax

```
PvcsGetArchiveInfoVB2(
   ARCHIVEHANDLEhArchive,/* Input */
   unsigned char*fileName,/* Input */
   unsigned short*reclen,/* Output */
   unsigned short*renum_start_col,/* Output */
   unsigned short*renum_end_col,/* Output */
   unsigned long*renum_start_val,/* Output */
   unsigned long*renum_step_val,/* Output */
   unsigned char*cmt_str,/* Output */
   unsigned char*newline,/* Output */
   unsigned char*diffmask)/* Output */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open
reclen	Pointer to an integer that receives the workfile fixed record length.
renum_start_col	The renumber start column short.
renum_end_col	The renumber end column short.
renum_start_val	The renumber start value short.
renum_step_val	The renumber step value short.
cmt_str	Pointer to a comment prefix string.
newline	Pointer to a newline string.
diffmask	Pointer to the column mask range string.

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	String that contains the name of an archive or a workfile. This parameter is required only if the archive is not open

reclen	Integer that receives the workfile fixed record length.
renum_start_col	Integer that receives the renumber start column value.
renum_end_col	Integer that receives the renumber end column value.
renum_start_val	Integer that receives the renumber start value.
renum_step_val	Integer that receives the renumber step value.
cmt_str	String that receives the comment prefix string.
newline	String that receives the newline string.
diffmask	String that receives the column mask range.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_BUFFER_OVERFLOW
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Consideration

Any of the non-input parameters may be passed a NULL value if no return value for that parameter is desired.

Example

```
/* PvcsGetArchiveInfoVB2 example */

int         status = 0;
unsigned shortreclen;
unsigned shortrenum_start_col;
unsigned shortrenum_end_col;
unsigned longrenum_start_val;
unsigned longrenum_step_val;
unsigned charcmt_str[32];

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/*
 * PvcsGetArchiveInfoVB2: Get archive info into separate
    variables
 * instead of an ARCHIVEINFO structure. If you don't want the
 * information for a specific field, then pass NULL
 * (i.e. see the last two parameters).
 */
status = PvcsGetArchiveInfoVB2(
   ARCHIVEHANDLE_NOT_OPEN, /* I: Archive handle */
   "foo.c_v",              /* I: Name of archive or workfile */
   &reclen,                /* O: Workfile fixed record
                length */
   &renum_start_col        /* O: Renumber start column */
   &renum_end_col,         /* O: Renumber end column */
   &renum_start_val,       /* O: Renumber starting value */
   &renum_step_val,        /* O: Renumber increment */
   cmt_str,                /* O: Comment prefix string */
   NULL,                   /* O: Newline string */
   NULL                    /* O: Column mask range */
```

```
   );

if (!status)
{
    printf("Archive name: %s\n","foo.c_v");
    printf("Workfile record length: %d\n",reclen);
    printf("Renumber start column: %d\n",renum_start_col);
    printf("  Renumber end column: %d\n",renum_end_col);
    printf("  Renumber starting value:
           %ld\n",renum_start_val);
    printf("Renumber increment value:
           %ld\n",renum_step_val);
    printf("Comment prefix string: %s\n",cmt_str);
}
```

Related Functions

PvcsGetArchiveInfo on page 85
PvcsGetArchiveInfoVB1 on page 87
PvcsGetLockInfoVB on page 97
PvcsGetRevisionInfoVB on page 107

PvcsGetErrorMessage

This function retrieves the text message that corresponds to a return code and copies it into a buffer that the caller provides.

Syntax

```
PvcsGetErrorMessage(
   unsigned int errorCode,/* Input */
   unsigned char *msgBuffer,/* Output */
   unsigned int bufLen)/* Input */
```

Parameters

errorCode	An error code returned by a Developer's Toolkit function.
msgBuffer	Pointer to the buffer that receives the message string corresponding to the error code. The messages returned are the return values listed in Chapter 4, "Return Values"
bufLen	Length of the buffer that receives the message string.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

This function returns the name of the error, not a description. For example, it returns the name PVCS_E_INTERNAL instead of the description, "An internal toolkit error has occurred."

Example

```
char msgBuffer[128];
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);
```

```
/* Force an error */
status = PvcsCreateArchive(
   NULL,
   "^+=",   /* Invalid file name */
   NULL,
   NULL,
   NULL,
   PVCS_CREATE_NO_OVERWRITE);

if (status) {

   /* Retrieve error message */
   status = PvcsGetErrorMessage(
            status,/* Error code */
            msgBuffer,/* Buf to receive msg */
            sizeof(msgBuffer));/* Length of buffer */

   /* Display error message */
   if (!status)
   printf("PvcsComputeArchiveName error: %s\n", msgBuffer);
}
```

Related Function PvcsQueryConfigurationError on page 143

PvcsGetExtRevAttribute

This function retrieves an extended attribute that was previously attached to a revision using **PvcsPutExtRevAttribute**. Extended revision attributes are used to attach free-form binary or text data to a revision. The attribute contains a user-defined keyword, which you use to access the attribute data.

Syntax

```
PvcsGetExtRevAttribute(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *revarg,/* Input */
   unsigned char *keyword,/* Input */
   unsigned char *extAttribute,/* Output */
   unsigned short bufLen)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of the archive or workfile. This parameter is required only if the archive is not open.
revarg	Pointer to a string that contains a revision number or version label. The function retrieves the extended attribute associated with this revision. If the version label begins with a number, precede it with a backslash (\). A null parameter defaults to the tip revision on the trunk.

keyword	Pointer to a buffer that contains the user-defined value used to retrieve this extended attribute.
extAttribute	Pointer to a buffer to receive the extended revision attribute record. See below for the buffer format.
bufLen	Length of the extended attribute buffer.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_BUFFER_OVERFLOW
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
PVCS_E_NO_ATTRIBUTE
PVCS_E_USER_ABORTED
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- You can only retrieve one attribute per function call.
- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- The format of the *extAttribute* buffer is as follows:

Length
Keyword
Null
User_data

Length	Two-byte unsigned value that is the length of the keyword and null terminator, plus the length of the user data. The length does not include the length field itself.
Keyword	Null-terminated string that contains a user-defined keyword.
User_data	Free-format data, which may be text or binary.

- The maximum length of the extended attribute record (including the length field) is 64K.
- If PVCS_E_BUFFER_OVERFLOW is returned, the first two bytes of the attribute buffer contain the actual length that was required.

Example

```
char *keyword = "BetaTextKey";
char *userdata = "This Beta release was sent out to 500 customers \
as a precursor to the General Availability release.";
int extAttCount = 1;
char *archive = "foo.c_v";
char *revision = "Beta Release 1.0";
char getExtAttBuf[256];
char *bufPtr;
char *ptr;
int status;
```

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Add an extended attribute */
extAttBuf.length = strlen(keyword) + strlen(userdata) + 2;
strcpy(extAttBuf.data, keyword);
ptr = strchr(extAttBuf.data, '\0');
strcpy(ptr + 1, userdata);
bufPtr = (char *)&extAttBuf;
status = PvcsPutExtRevAttribute(ARCHIVEHANDLE_NOT_OPEN, archive,
   revision, bufPtr, extAttCount);
/* Now retrieve the extended attribute that was stored */
if (!status) {
   /* Retrieve extended attribute */
   status = PvcsGetExtRevAttribute(ARCHIVEHANDLE_NOT_OPEN,
            archive,
            revision,
            keyword,
            getExtAttBuf,
            sizeof(getExtAttBuf));
   /* Display extended attribute */
   if (!status){
            printf("User data associated with keyword
    \"%s\"is:\n\"%s\"\n",
            keyword, getExtAttBuf);
   }
    }
```

Related Functions
PvcsOpenArchive on page 129
PvcsPutExtRevAttribute on page 132

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Wildcard expansion	*Wildcards*
Specifying archive and workfile names	*File Specification*

PvcsGetLockInfo

This function returns information about locked revisions. It is equivalent to the VLOG -BL command.

Syntax

```
PvcsGetLockInfo(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *revision,/* Input */
   unsigned char *lockers,/* Input */
   unsigned short *pLockCount,/* Input / Output */
   LOCK *pLockInfo, /* Output */
   unsigned short buflen)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open.
revision	Pointer to a string that contains a revision number. If the parameter is null, the function returns information about all locks in the archive. This parameter is equivalent to the -R option used with the VLOG -BL command.
lockers	Pointer to a string that contains a comma-separated list of user IDs used to limit the search for locks. If the parameter is null, this function returns information about locks owned by any user. This parameter is equivalent to the *user_id* parameter to the VLOG -BL command.
pLockCount	Pointer to a variable that specifies the number of LOCK structures to retrieve. The actual number retrieved is returned. Specify -1 to retrieve all LOCK structures.
pLockInfo	Pointer to a buffer to receive lock information.
buflen	The length of the *pLockInfo* buffer.

Return Values

The return value is zero if the function is successful. Other values may be:

PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_EMPTY
PVCS_E_BAD_ARCHIVE_HANDLE
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER
PVCS_E_NOT_LOCKED

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- This function organizes the buffer as an array of LOCK structures, followed by a variable-length data area which contains revision numbers and user IDs. The *pLockInfo* buffer must be large enough to contain the LOCK structures and variable data area. For details on the LOCK structure, see Chapter 5, "Data Structures"
- Use the NoCase directive to conduct a case-insensitive search for user IDs that match the *lockers* list.

Example

```
LOCK lockInfoBuf[256];
    LOCK *pLock;
short lockCount = -1;

PvcsGetLockInfo(
   ARCHIVEHANDLE_NOT_OPEN,/* Archive handle */
   archive, /* Archive name */
   NULL,    /* Revision (any revision) */
   NULL,    /* VCSID (any) */
   &lockCount,/* Count (unlimited) */
   lockInfoBuf,/* Buffer */
   sizeof(lockInfoBuf));/* Buffer length */
```

```
for (pLock = lockInfoBuf; lockCount-- != 0; pLock++) {
   printf("Revision: %s\n", pLock->revision);
   printf("New rev: %s\n", pLock->newRevision);
   printf("Locked by: %s\n", pLock->locker);
}
```

Related Functions
PvcsLockRevisionGroup on page 117
PvcsOpenArchive on page 129

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Viewing lock information	*VLOG command*
Revision locking	*Revision Locking*
Controlling case-sensitivity of user IDs	*Case directive*

PvcsGetLockInfoVB

This function, specific for Visual Basic users, returns information about locked revisions.

Syntax

```
PvcsGetLockInfoVB(
   ARCHIVEHANDLEhArchive,/* Input */
   unsigned char*fileName,/* Input */
   unsigned char*revision,/* Input */
   unsigned char*lockers,/* Input */
   unsigned char*oldRevision,/* Output */
   unsigned char*newRevision,/* Output */
   unsigned char*lockers,/* Output */
   unsigned shortlock_info_index,/* Input */
   PVCS_FLAGSflags)/* Input */
```

Parameters

hArchive	Handle return by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NO_OPEN
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open.
revision	Pointer to a string that contains the revision if the parameter is NULL (see *oldRevision*).
lockers	Pointer to a string that contains a comma-separated list of user IDs used to limit the search for locks. If the parameter is null, this function returns information about locks owned by any user. This parameter is equivalent to the *user_id* parameter to the VLOG -BL command.
oldRevision	Pointer to a string that that receives the name of the locked revision.
newRevision	Pointer to the string that receives the new revision number to be created when the workfile is checked in.

lockers	Pointer to the string to receive the user ID of the locker.
lock_info_index	Specifies the requested number of lock information structures on which to return information.
flags	This parameter is reserved for future use.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_EMPTY
PVCS_E_BAD_ARCHIVE_HANDLE
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER
PVCS_E_NO_REVISION
PVCS_E_NOT_LOCKED
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- Any of the non-input parameters may be passed a null value if no return value for that parameter is desired.
- You can determine how many locks are in an archive by calling **PvcsGetArchiveInfoVB1** and looking at the *lockers* parameters.

Example

```
/* PvcsGetLockInfoVB example */

int         status = 0;
unsigned shortindex = 0;
unsigned charlockRevision[64];
unsigned charnewRevision[64];
unsigned charlocker[64];

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWIRTE);

/*
 * PvcsGetLockInfoVB: Gets the lock information for the given
     *archive into separate variables.This example gets the
 * first lock for thearchive not limited by revision or
 */locker.

status = PvcsGetLockInfoVB(
   ARCHIVEHANDLE_NOT_OPEN, /* I: Archive Handle */
   "foo.c_v",              /* I: Name of archive or workfile */
   NULL,                   /* I: Revision */
   NULL,                   /* I: VCSID of locker */
   lockRevision,           /* O: The locked revision */
   newRevision,            /* O: Revision to create at check
                in */
   locker,                 /* O: User ID of the locker */
   index,                  /* I: Index of lock to return
            values from */
   PVCS_REVINFO_USE_DATETIME_FORMAT/* I: Bit field */
   );

if (!status)
{
   printf("Revision: %s\n",lockRevision);
```

```
   printf("New revision on Check In: %s\n",newRevision);
   printf("Locker ID: %s\n",locker);
}
```

Related Functions
PvcsGetArchiveInfoVB1 on page 87
PvcsGetArchiveInfoVB2 on page 90
PvcsGetLockInfo on page 95
PvcsGetRevisionInfoVB on page 107

PvcsGetPromoParent

This function returns the name of the parent of a promotion group.

Syntax

```
PvcsGetPromoParent(
   unsigned char *buffer,/* Output */
   unsigned short length,/* Input */
   unsigned char *node_name)/* Input */
```

Parameters

buffer	Pointer to a buffer that receives the name of the parent of *node_name*.
length	Length of the buffer.
node_name	Name of the promotion group for which to find the parent.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER
PVCS_E_INVALID_PROMO
PVCS_E_PROMO_NO_NODE

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
char *child = "SOFTDEV";
char parent[64];
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Obtain parent of specified group name */
status = PvcsGetPromoParent(
   parent,  /* Buf to receive parent group name */
   sizeof(parent),/* Length of buffer */
   child);  /* Child group name */

/* Display parent group  */
if (!status)
   printf("Parent of promotion group \"%s\" is \"%s\".\n", child,
    parent);
```

Related Functions
PvcsGroupToRevision on page 111
PvcsPromoteRevision on page 131

PvcsVerifyPromoTree on page 177
PvcsVerifyPromoTreeNodeExist on page 178

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Promotion models	*Promotion*
Defining a promotion group	*Promote directive*
Promoting revisions	*VPROMOTE command*

PvcsGetRevision

This function checks out revisions from archives. It is equivalent to the GET command.

Syntax

```
PvcsGetRevision(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *workfileName,/* Input */
   unsigned char *revarg,/* Input */
   unsigned char *date,/* Input */
   unsigned char *update,/* Input */
   unsigned char *locker,/* Input */
   unsigned char *grp_name,/* Input */
   void *reserved,  /* Input */
   PVCS_FLAGS getFlags)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open. The file name can contain wildcards.
workfileName	Pointer to the name of a workfile. If this parameter is null, the function obtains the workfile name from the archive. If you specify a workfile name, then *fileName* must contain an archive name.
revarg	Pointer to a string that contains a revision number, version label, or promotion group. If the version label begins with a number, precede it with a backslash (\). This parameter is equivalent to the GET -R command.
date	Pointer to a string that contains a date/time. The function checks out the latest revision with a check-in date on or before that date. This parameter is equivalent to the GET -D command.
update	Pointer to a string that contains a date specification. The function only checks out the revision if its check-in date is later than the specified date. If you specify a date, you must set the PVCS_GET_NEWER flag for the *getFlags* parameter.

locker — Pointer to a string that contains the user ID of the person who is locking the revision. If the parameter is null, the current user ID is used.

grp_name — Pointer to a string that contains the name of the promotion group to which the revision is drawn down. This parameter is used only if you have defined a promotion model. The group must be at the lowest level of the promotion model. This parameter is equivalent to the GET -G command.

reserved — Used internally. Pass a null pointer for this parameter.

getFlags — Bit field that controls the operation of this function. Values include:

- PVCS_GET_LOCK
  Locks the revision. This flag is equivalent to the GET -L command.
- PVCS_GET_MULTILOCK
  Adds an additional lock if the MultiLock directive is in effect.
- PVCS_GET_BRANCH_PERMITTED
  Grants permission to lock a revision that would cause a branch to be created upon check in. This flag does not do anything unless the BranchWarn directive is in effect. If the BranchWarn directive is in effect, you must give permission to lock a revision that would cause Version Manager to create a branch.
- PVCS_GET_NEWER
  Checks out the revision only if it is newer than the existing workfile. If you specify a date for the *update* parameter, using this flag checks out the revision only if it is newer than the date. This flag is equivalent to the GET -U command.
- PVCS_GET_NOBRANCH
  Denies permission to lock a revision that would cause a branch to be created upon check in. This flag does not do anything unless the BranchWarn directive is in effect. The function returns PVCS_E_BRANCHWARN if the revision that you are locking would cause a branch to be created.
- PVCS_GET_NOMULTILOCK
  Does not add an additional lock even if the MultiLock directive is in effect.
- PVCS_GET_NO_OVERWRITE
  Does not overwrite an existing workfile. This flag is equivalent to the GET -N command.
- PVCS_GET_OVERWRITE
  Overwrites an existing workfile. This flag is equivalent to the GET -Y command.
- PVCS_GET_PIPE
  Writes the revision to standard output. This flag is equivalent to the GET -P command.

- PVCS_GET_TOUCH
  Sets the workfile timestamp to the current time. This flag is equivalent to the GET -T command.
- PVCS_GET_UPDATE
  Checks out the archive only if its check-in date is later than the date specified using the *update* parameter. This flag is equivalent to the GET -U command.
- PVCS_GET_WRITABLE
  Checks out a writable copy of the workfile. This flag is equivalent to the GET -W command.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_ACCESS_DENIED
PVCS_E_ACCESS_VIOLATION
PVCS_E_ALREADY_EXISTS
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_BAD_FILENAME
PVCS_E_FILE_BUSY
PVCS_E_LOCKED_REVISION
PVCS_E_INVALID_PARAMETER
PVCS_E_NO_REVISION
PVCS_E_USER_ABORTED
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- Use the *workfileName* parameter when you are generating a workfile that has a different name than the name stored in the archive. This is equivalent to the following command-line example:

  ```
  GET foo.c_v(d:\temp\test.c)
  ```

  In this example, *fileName* is FOO.C_V and *workfileName* is D:\TEMP\TEST.C.

  The *workfileName* parameter can consist of a drive or path. This is equivalent to the following command-line example, where *fileName* is *.??V, and *workfileName* is C:\SOURCE:

Example

```
#return REVINFO_PAD = 256
REVINFO *revinfo;
unsigned short revcount = -1;
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Check out revisions by version label, overwriting workfiles */
PvcsGetRevision(ARCHIVEHANDLE_NOT_OPEN,
   "*.c",   /* Archive names */
   NULL,    /* Use default workfile names */
   "Beta Release 1.0",/* Version label to be checked
            out */
   NULL,    /* No date restriction */
   NULL,
```

```
NULL,    /* Use user's VCSID as locker */
NULL,    /* Assume no promotion hierarchy */
NULL,    /* Reserved *
PVCS_GET_OVERWRITE);/* Force overwrite of existing
                       workfiles */
```

Related Functions
PvcsGetRevisionInfo on page 103
PvcsLog on page 119
PvcsOpenArchive on page 129
PvcsPutRevision on page 134

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Checking out revisions	*GET command*
Wildcard expansion	*Wildcards*
Specifying archive and workfile names	*File Specification*
Specifying dates	*Date and Time Specification*
Drawing revisions down to a development group	*Promotion*

PvcsGetRevisionInfo

This function returns information about revisions in specified archives. It is equivalent to the VLOG -BR command.

Syntax

```
PvcsGetRevisionInfo(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *revision,/* Input */
   unsigned short *pRevCount,/* Input/Output */
   REVINFO *pRevInfo,/* Output */
   PVCS_FLAGS flags)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open.
revision	Pointer to a string that contains a revision number. If the parameter is null, the function returns information about all revisions, beginning with the tip of the trunk. This parameter can contain a version label. If the version label begins with a number, precede it with a backslash (\).
pRevCount	Pointer to a variable that specifies the number of REVINFO structures to retrieve. The parameter returns the number of structures retrieved. Specify -1 for all revisions.
pRevInfo	Pointer to a buffer to receive revision information.

flags — Bit field that controls how the function processes a range of revisions. Values include:

- PVCS_REVINFO_RECURSE
  Includes all branches emanating from specified revisions. This flag is equivalent to appending a plus sign (+) to a revision range.
- PVCS_REVINFO_NO_RECURSE
  Ignores branches emanating from specified revisions.

Return Values

The return value is zero if the function is successful. Other values may be:

PVCS_E_ARCHIVE_EMPTY
PVCS_E_BAD_ARCHIVE_HANDLE
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER
PVCS_E_ACCESS_VIOLATION

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- To retrieve information for more than one revision, specify the number of revisions in *pRevCount.* In this case, *pRevInfo* must point to an array of **pRevCount* REVINFO structures. Use the *revision* parameter to specify the revision with which to begin processing. The function processes revisions in reverse order, so you would specify the tip of the trunk as the starting point to retrieve information about all revisions.
- The *pRevInfo* buffer must be large enough to contain the REVINFO structure and space for the variable-length description strings. You should allocate 64 bytes per revision.

  See Chapter 5, "Data Structures" for details on the REVINFO structure.

Example

```
REVINFO *revinfo;
unsigned short revcount = -1;
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Allocate buffer to hold revision information */
revinfo = (REVINFO *)malloc(sizeof(REVINFO) + REVINFO_PAD);

/* Report revision number corresponding to a version label */
status = PvcsGetRevisionInfo(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",/* Archive name */
   "Beta Release 1.0",/* Ver label to search for */
   &revcount,        /* Only one rev associated w/
             label */
   revinfo, /* Buffer to receive rev info */
   PVCS_REVINFO_RECURSE);/* Search all branches */

if (!status)
printf("Revision corresponding to \"Beta Release 1.0\" is
   \"%s\".\n", revinfo->revstr);
```

Related Functions
PvcsGetArchiveInfo on page 85
PvcsOpenArchive on page 129

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Specifying archive and workfile names	*File Specification*
Specifying revisions	*Revision Specification*

PvcsGetRevisionInfo2

This function returns the following information associated with a revision:

- Version labels assigned to a revision
- Promotion groups assigned to a revision
- Locks on a revision
- Author of a revision
- Change description of a revision

Syntax

```
PvcsGetRevisionInfo2(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *revision,/* Input */
   unsigned char **author,/* Output */
   unsigned char **versions,/* Output */
   unsigned char **promoGroups, /* Output */
   unsigned char **lockers,/* Output */
   unsigned char **descr,/* Output */
   unsigned char *resultBuf,/* Input */
   unsigned bufLen, /* Input */
   void *reserved)  /* Reserved */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open.
revision	Pointer to a string that contains the revision for which you want to retrieve information. If the parameter is null, Developer's Toolkit returns information about the tip revision.
author	(Output). Pointer to a pointer to a string: the user ID of the user who created this revision.
versions	(Output). Pointer to a pointer to a list of strings. Each is a version label associated with this revision. If Developer's Toolkit returns a null pointer, then no version labels are present.

promoGroups	(Output). Pointer to a pointer to a list of strings. Each string is a promotion group associated with this revision. The strings are null-separated with a double-null pointer at the end of the list. Developer's Toolkit returns a null pointer if no promotion groups are present.
lockers	(Output). Pointer to a pointer to a list of strings. Each is a user ID that owns a lock on the revision. The strings for these are null-separated with a double-null pointer at the end of the list. Developer's Toolkit returns a null pointer if no locks are present.
descr	(Output). Pointer to a pointer to a string: the revision change description. If Developer's Toolkit returns a null pointer, then no change description is present.
resultBuf	Pointer to a buffer that this function uses to store the result data. Your program must allocate the buffer. The pointers referenced by *author*, *versions*, *promoGroups*, *lockers*, and *description* will point to memory in *resultBuffer*.
bufLen	Length of the result buffer.
reserved	Reserved for future use. Must be null.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_ARCHIVE_EMPTY
PVDS_E_NO_REVISION
PVCS_E_BAD_ARCHIVE_HANDLE
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER
PVCS_E_ACCESS_VIOLATION
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- The result buffer must be large enough to hold all of the variable-length data. This can get quite large if you have a number of version labels and lengthy change descriptions.

Example

```
int         rc;
PVCS_PUCHAR szFilename2 = NULL;
PVCS_PUCHAR szRevision2 = NULL;
PVCS_PUCHAR szAuthor = NULL;
PVCS_PUCHAR szVersions = NULL;
PVCS_PUCHAR szPromoGroups = NULL;
PVCS_PUCHAR szLockers = NULL;
PVCS_PUCHAR szDescription = NULL;
PVCS_PUCHAR pRevInfo2Buffer = NULL;
unsigned shortbufSize;

/* Initialize configuration settings */
rc = PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/*
 * PvcsGetRevisionInfo2 only for single archives and single revisions
 */
bufSize = 256;
```

```
pRevInfo2Buffer = (PVCS_PUCHAR)malloc(bufSize);

rc =PvcsGetRevisionInfo2(
   ARCHIVEHANDLE_NOT_OPEN,
   szFilename2,
   szRevision2,
   &szAuthor,
   &szVersions,
   &szPromoGroups,
   &szLockers,
   &szDescription,
   pRevInfo2Buffer,
   bufSize,
   NULL
   );

if (!rc) return(rc){
    printf(" Rev: %s\n",pRevInfoBuffer->revstr);
    printf("        Author:        %s\n",szAuthor);
    printf("        Versions:      ");
    print_string_list(szVersions);
    printf("        PromoGroups:   ");
    print_string_list(szPromoGroups);
    printf("        Lockers:       ");
    print_string_list(szLockers);
    printf("        Decription:    %s\n", szDescription);
}
```

Related Functions
PvcsGetArchiveInfo on page 85
PvcsOpenArchive on page 129

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Specifying archive and workfile names	*File Specification*
Specifying revisions	*Revision Specification*

PvcsGetRevisionInfoVB

This function, specifically for Visual Basic users, returns information about a revision in a specified archive.

Syntax

```
PvcsGetRevisionInfoVB(
   ARCHIVEHANDLEhArchive,/* Input */
   unsigned char  *fileName,/* Input */
   unsigned char*revision,/* Input */
   unsigned short*branch_count,/* Output */
   unsigned short*lock_count,/* Output */
   unsigned short  *level,/* Output */
   unsigned char   *date,/* Output */
   unsigned char   *mdate,/* Output */
```

```
unsigned short*ord,/* Output */
unsigned char*revstr,/* Output */
unsigned shortrev_info_index,/* Input*/
PVCS_FLAGSflags)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	String that contains the name of an archive or a workfile. This parameter is required only if the archive is not open.
revision	String that contains the revision number to retrieve.
branch_count	Integer that receives the number of branches off this revision.
lock_count	Integer that receives the number of locks on this revision.
level	Integer that receives the revision level.
date	String that receives the date the revision was checked in.
mdate	String that receives the date the revision was modified.
ord	Integer that receives the ordinal number.
revstr	String that receives the revision.
rev_info_index	Specifies the revision information structure about which to return information. The first structure has an index of 0 (the tip of the trunk) and the last index is one less than the value of *revcnt* returned by **PvcsGetArchiveInfoVB1** or **PvcsGetArchiveInfo**.
flags	Indicates how to modify the behavior of this function. Values include: ■ PVCS_REVINFO_RECURSE Includes all branches emanating from specified revisions. this flag is equivalent to appending a plus sign (+) to a revision range ■ PVCS_REVINFO_NO_RECURSE Ignores branches emanating from specified revisions. ■ PVCS_REVINFO_DATETIME_FORMAT Causes the *create_time* string to be formated with the currently configured date/time format. If this flag is not set, then the *create_time* string will be returned with the default format mm/dd/yyyy hh:mm:ss.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_ARCHIVE_EMPTY
PVCS_E_ACCESS_VIOLATION
PVCS_E_BAD_ARCHIVE_HANDLE
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER
PVDS_E_NO_REVISION

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- Any of the non-input parameters may be passed a NULL value if no return value for that parameter is desired.

- You can determine how many revisions are in an archive by calling **PvcsGetArchiveInfoVB1** and looking at the *revcnt* parameter.
- If the *revisions* parameter is not NULL, it will override the index passed in the *rev_info_index* parameter.

Example

```
/* PvcsGetRevisionInfoVB example */

int         status = 0;
unsigned shortindex = 0;
unsigned shortbranch_count;
unsigned shortlock_count;
unsigned shortlevel;
unsigned shortord;
unsigned chardate[32];
unsigned charmdate[32];
unsigned charrevstr[64];

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWIRTE);

/*
 * PvcsGetRevisionInfoVB: Get revision info into separate
 * variables instead of a REVINFO structure. This example
 * gets information from the tip revision. If you don't
 * want the information for a specific field, then pass NULL
 * (i.e. see the modification date parameter).
 */
status = PvcsGetRevisionInfoVB(
   ARCHIVEHANDLE_NOT_OPEN,/* I: Archive handle */
   "foo.c_v",/* I: Name of archive or workfile */
   NULL,       /* I: Revision to retrieve */
   &branch_count,   /* O: Number of branches from this
            rev */
   &lock_count,/* O: Number of locks on this
            revision */
   &level,  /* O: Revision tree level, 0 = trunk */
   date,    /* O: Text version of check-in rev
            date */
   NULL,    /* O: Text version of rev mod date */
   &ord,    /* O: Ordinal number in archive */
   revstr,  /* O: Revision number as text */
   index,   /* I: Which revision to get info for */
   PVCS_REVINFO_USE_DATETIME_FORMAT  /* I: Bit field */
   );

if (!status)
{
   printf(" Revision: %s\n",revstr);
   printf(" Branch Count: %d\n",branch_count);
   printf(" Lock Count: %d\n",lock_count);
   printf(" Level: %d\n",level);
   printf(" Check In Date: %s\n",date);
   printf(" Ordinal Number: %d\n",ord);
}
```

Related Functions
PvcsGetRevisionInfo on page 103
PvcsGetLockInfoVB on page 97
PvcsGetArchiveInfoVB1 on page 87
PvcsGetArchiveInfoVB2 on page 90

PvcsGetUserInfo

This function returns the current user ID and login source.

Syntax

```
PvcsGetUserInfo(
   unsigned char *userID,/* Output */
   unsigned short userIDLen,/* Input */
   unsigned char *loginSource,/* Output */
   unsigned short loginSourceLen,/* Input */
   unsigned short *loginSourceValue)/* Output */
```

Parameters

userID	Pointer to a buffer that receives the current user ID.
userIDLen	Variable that contains the length of the *userID* buffer.
loginSource	Pointer to a buffer that receives a string that contains the login source. This string contains one of the following values: HOST NETWARE LANMAN VLOGIN VCSID UNKNOWN WNET
loginSourceLen	Variable that contains the length of the *loginSource* buffer.
loginSourceValue	Pointer to a variable that receives a number that identifies the login source. This number is one of the following values: PVCS_LOGSRC_HOST PVCS_LOGSRC_NETWARE PVCS_LOGSRC_LANMAN PVCS_LOGSRC_NOT_ESTABLISHED PVCS_LOGSRC_VLOGIN PVCS_LOGSRC_VCSID PVCS_LOGSRC_UNKNOWN PVCS_LOGSRC_WNET

Return Values

The return value is zero if the function is successful. Other values may be:

PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Special Consideration

If you call this function before calling **PvcsQueryConfiguration**, you may not receive the actual login name or source. The LogIn directive affects where Version Manager looks for the user ID.

Example

```
char userBuf[64];
char sourceBuf[16];
int sourceValue;
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);
status = PvcsGetUserInfo(
   userBuf, /* Buffer receiving user ID */
   sizeof(userBuf),/* Length of buffer */
   sourceBuf,/* Buffer receiving login source */
   sizeof(sourceBuf),/* Length of buffer */
   &sourceValue);/* Receives login source ident ID */

if (!status)
   printf("Current user ID is \"%s\", login source is
       \"%s\".\n", userBuf, sourceBuf);
```

Related Function PvcsQueryConfiguration on page 140

PvcsGroupToRevision

This function returns the revision numbers that are assigned to a specified promotion group. It is equivalent to the VLOG -BG command.

Syntax

```
PvcsGroupToRevision(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *grpName,/* Input */
   unsigned short *pGrpCount,/* Input/Output */
   unsigned char *groupTable,/* Output */
   unsigned short bufLen)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open. This parameter cannot contain wildcards.
grpName	Pointer to a string that contains a promotion group. The promotion group must be assigned to a revision in the archive. If *grpName* is null, the function returns revision/group pairs up to the limit specified in **pGrpCount* or until the *groupTable* buffer is full.
pGrpCount	Pointer to a variable that specifies the number of revision/group pairs to retrieve. The parameter returns the actual number that were retrieved.
groupTable	Pointer to a buffer that receives the revision/group pairs.
bufLen	The length of the *groupTable* buffer.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER
PVCS_E_PROMO_NO_NODE

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- The format of the *groupTable* buffer is as follows:

revisionString
null
groupString
null
. . .
null

Each revision string and group string is null-terminated, and the end-of-buffer is marked by an additional null character. The revision number is returned as a printable string—for example, 1.15.

Example

```
unsigned short grpCount = USHORT_MAX;
char grpBuffer[512];
char *bufPtr;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Obtain revisions associated with group names */
PvcsGroupToRevision(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",/* Name of archive or workfile */
   NULL,    /* All revision/group pairs */
   &grpCount,/* Returns number of pairs retrieved */
   grpBuffer,/* Buf to receive revision/group pairs
   sizeof(grpBuffer));/* Length of the buffer */

/* Display all revisions with group names */
bufPtr = grpBuffer;
while (grpCount--) {
   printf("Revision: %s", bufPtr);
   bufPtr += strlen(bufPtr) + 1;/* Skip over revision */
   printf("  Group: %s\n", bufPtr);
   bufPtr += strlen(bufPtr) + 1;/* Skip over group */
     }
```

Related Functions

PvcsGetPromoParent on page 99
PvcsGroupToRevision on page 111
PvcsOpenArchive on page 129
PvcsPromoteRevision on page 131

PvcsVerifyPromoTree on page 177
PvcsVerifyPromoTreeNodeExist on page 178

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Specifying archive and workfile names	*File Specification*
Promotion models	*Promotion*
Defining a promotion group	*Promote directive*
Promoting revisions	*VPROMOTE command*

PvcsInit

This function initializes internal variables, allocating memory for file buffers, and verifying the license. You can call this function in your program's initialization routine to verify that you will be able to use Developer's Toolkit services later in your program.

Syntax

```
PvcsInit(
   unsigned char *progname)/* Input */
```

Parameter

progname	Pointer to a string that names the calling program. Developer's Toolkit prefixes warning and error messages with this string. If this string is null, Developer's Toolkit uses the string `Serena API`.

Return Value

The return value is zero if the function is successful. Otherwise it returns `PVCS_E_NO_MEMORY`. See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

Developer's Toolkit calls this function if you do not call it in advance.

Example

```
int status;

/* Initialize Developer's Toolkit */
status = PvcsInit("My Application");

/* Display error message upon failure */
if (status)
   printf("Error, unable to initialize application.\n");
```

Related Functions

PvcsCloseArchive on page 63
PvcsOpenArchive on page 129
PvcsCancelUpdate on page 56

PvcsIsArchive

This function determines whether a file name is an archive by returning a status code. Developer's Toolkit does not perform any VCSDir checking or ArchiveSuffix translation on the file name.

Syntax

```
PvcsIsArchive(
   unsigned char *fileName,/* Input */
   unsigned short *is_archive)/* Output */
```

Parameters

fileName	Pointer to the name of the file to process. The file name cannot contain wildcards and must be a fully qualified path name.
is_archive	Pointer to a short that will be set to TRUE (1) if the file is an archive or FALSE (0) if the file is not an archive.

Return Values

This function returns zero if successful. Other values include:
PVCS_E_ARCHIVE_NOT_FOUND
PVD_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
unsigned char *file = "c:\vcsdir\foo.c_v";
unsigned short is_archive;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Determine if file is an archive */
status = PvcsIsArchive(
   file,    /* Name of file to check */
   &is_archive);/* Indicates whether */
                /* the file is an archive */
if (status)
   printf("%s %s an archive.\n", file, (is_archive)? "is":
        "is not");
```

PvcsIsUserInDatabase

This function searches the access control database for a specific user. This function requires the ViewAccessDB privilege.

Syntax

```
PvcsIsUserInDatabase(
   unsigned char *user,/* Input */
   unsigned short *found)/* Output */
```

Parameters

user	Pointer to the name of the user.
found	Pointer to a short that is set to TRUE if the user is in the database or FALSE if the user cannot be found.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_BAD_ACCESS_USER
PVCS_E_CANT_OPEN_ACCESSDB
    PVCS_E_INVALID_PARAMETER
    PVCS_E_UNKNOWN_USER
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/*
 * PvcsIsUserInDatabase: See if user is in access database.
 * Return result in 2nd parameter, TRUE for yes, FALSE for no.
 */

rc = PvcsIsUserInDatabase(
   szUser,
   &found);

if (found)
   printf ("User %s is in the database.\n", szUser);
```

Related Functions

PvcsLogin on page 122
PvcsGetUserInfo on page 110
PvcsQueryUserAccess on page 150

PvcsListJournal

This function creates a journal report from a journal file. It is equivalent to the VJOURNAL command.

Syntax

```
PvcsListJournal(
   unsigned char *journalFile,/* Input */
   unsigned char *date,/* Input */
   unsigned char *archiveList,/* Input */
   unsigned char *users,/* Input */
   unsigned char *operations,/* Input */
   unsigned char *reportFile,/* Input */
   PVCS_FLAGS  flags)/* Input */
```

Parameters

journalFile	Pointer to a string that contains the name of the journal file to process. If the pointer is null or points to a null string, the name of the journal file is obtained from the Journal directive in the configuration file.
date	Pointer to a string that contains a date or a date range. This parameter limits the report to operations that occurred within the date range.
archiveList	Pointer to a string that contains a list of archive names separated by spaces. This parameter is *not* case-sensitive and accepts workfiles, partial directories, and wildcards. It will list all like names across directories. This parameter limits the report to operations on these archives.
users	Pointer to a string that contains a list of user IDs separated by spaces. This parameter limits the report to operations by these users.
operations	Pointer to a string that contains a list of operations separated by spaces. For example, PUT, PUT -R. This parameter limits the report to these operations.
reportFile	Pointer to a string that contains the name of a file to receive the journal report. An existing file will be overwritten unless you specify the flag PVCS_JOURNAL_APPEND.
flags	Bit field that controls the operation of this function. Values include: ■ PVCS_JOURNAL_APPEND Appends to the report file rather than overwriting it. ■ PVCS_JOURNAL_LOCKS Generates the lock list. This is equivalent to the VJOURNAL -XL command.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_BAD_DATE
PVCS_E_BAD_REDIRECT
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Special Consideration

If you do not want to limit the report with one of the limiting parameters, specify a null pointer for that parameter.

Example

```
char *outFile = "davee.jnl";
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);
   status = PvcsListJournal(
   "journal.vcs",/* Name of journal file */
   "93",    /* Limit to 1993 */
   NULL,    /* No archive restrictions */
   "DAVEE", /* Limit to actions by DAVEE */
   NULL,          /* No operation restrictions */
   outFile, /* File to receive report */
   (PVCS_FLAGS)0);/* Overwrite existing, no lock list */
```

```
if (!status)
   printf("Created journal report in \"%s\".\n", outFile);
```

Related Function: PvcsGetArchiveInfo on page 85

Related Topics: For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Viewing journal reports	*VJOURNAL command*
Contents and format of journal files	*Journal Files*
Specifying date ranges	*Date and Time Specification*
Enabling journal entries	*Journal directive*

PvcsLockRevisionGroup

This function locks a revision in an archive. It is equivalent to the VCS -L command.

Syntax

```
PvcsLockRevisionGroup(
   ARCHIVEHANDLE archiveHandle,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *revarg,/* Input */
   unsigned char *locker,/* Input */
   unsigned char *group,/* Input */
   PVCS_FLAGS flags)/* Input */
```

Parameters

archiveHandle	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open.
revarg	Pointer to a string that contains a revision number or version label. Use a null string to specify the tip revision. If the version label begins with a number, precede it with a backslash (\). This parameter is equivalent to the VCS -R command.
locker	Pointer to a string that contains the user ID of the locker. If the parameter is null, the current user ID is used.
group	Pointer to a string that contains the promotion group that is assigned to the locked revision. This group must be a development group. If you do not specify a group, and more than one development group is defined, the function returns PVCS_E_NO_GROUP. This parameter is equivalent to the VCS -G command.

flags — Bit field that controls the operation of this function. Values include:

- PVCS_LOCK_MULTILOCK
  Adds an additional lock if the MultiLock directive is in effect.
- PVCS_LOCK_NOMULTILOCK
  Does not add an additional lock even if MultiLock is in effect.
- PVCS_LOCK_BRANCH_PERMITTED
  Grants permission to lock a revision that would cause a branch to be created upon check in. This flag does not do anything unless the BranchWarn directive is in effect. If the BranchWarn directive is in effect, you must give permission to lock revisions that would cause Version Manager to create a branch.
- PVCS_LOCK_NOBRANCH
  Denies permission to lock a revision that would cause a branch to be created upon check in. This flag does not do anything unless the BranchWarn directive is in effect. The function returns PVCS_E_BRANCHWARN if the revision that you are locking would cause a branch to be created.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_NO_REVISION
PVCS_E_GROUP_LOCKED
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- You can use this function to lock a revision even if there is no promotion group by specifying a null pointer for the group parameter.
- This function replaces **PvcsLockRevision**. **PvcsLockRevisionGroup** is similar to **PvcsLockRevision**, but takes the *group* and *flags* arguments. For compatibility with previous versions, Developer's Toolkit still supports **PvcsLockRevision**.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Lock a revision */
PvcsLockRevisionGroup(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",/* Archive name */
   "1.12",  /* Revision to lock */
   "DAVEE", /* User ID of locker */
   "SOFTDEV",/* Group name associated with
            lock */
   PVCS_LOCK_NOMULTILOCK);/* Only one lock per revision */
```

Related Functions

PvcsGetLockInfo on page 95
PvcsGetRevision on page 100

PvcsUnLockRevision on page 172
PvcsOpenArchive on page 129

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Locking revisions	*Revision Locking VCS command*
Specifying archive and workfile names	*File Specification*
Wildcard expansion	*Wildcards*

PvcsLog

This function returns information about archives in a format suitable for reporting. It is equivalent to the VLOG command.

Syntax

```
PvcsLog(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *revision,/* Input */
   unsigned char *version,/* Input */
   unsigned char *owners,/* Input */
   unsigned char *authors,/* Input */
   unsigned char *lockers,/* Input */
   unsigned char *groups,/* Input */
   unsigned char *date,/* Input */
   unsigned char *outFile,/* Input */
   PVCS_FLAGS logFlags)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open. The file name can contain wildcards.
revision	Pointer to a string that contains a revision number or revision range. The function displays only revisions that match this specification. This parameter is equivalent to the VLOG -R command.
version	Pointer to a string that contains a version label. The function displays information only for revisions with this version label. This parameter is equivalent to the VLOG -V command.
owners	Pointer to a string that specifies a comma-separated list of user IDs. The function displays information only for archives that were created by these user IDs. This parameter is equivalent to the VLOG -O command.

authors Pointer to a string that specifies a comma-separated list of user IDs. The function displays information only for revisions modified by these users. This parameter is equivalent to the VLOG -A command.

lockers Pointer to a string that specifies a comma-separated list of user IDs. The function displays information only for revisions that are locked by these users. If the parameter is null, the function displays revisions that are locked by any user. This parameter is equivalent to the VLOG -L command.

groups Pointer to a string that contains the name of a promotion group. The function displays information only about archives that contain a revision associated with this group.

date Pointer to a string that specifies a date or date range. The function displays information only for revisions that were checked in within this range. This parameter is equivalent to the VLOG -D command.

outFile Pointer to a string that names an output file. If this parameter is null, the function displays the report. This parameter is equivalent to the VLOG -XO command.

logFlags Bit field that controls the operation of this function. Values include:

- PVCS_LOG_APPEND
  Appends to the output file rather than overwriting it.
- PVCS_LOG_CHECK_TIP
  Lists each archive whose tip revision is not the same as *revision* or *version*. This flag is equivalent to the VLOG -BC command.
- PVCS_LOG_GROUPS
  Displays revisions associated with the promotion group specified using the *groups* parameter. This flag is equivalent to the VLOG -BG command.
- PVCS_LOG_LOCKERS
  Displays a list of locked revisions. If the *lockers* parameter is specified, displays only revisions locked by those users. This flag is equivalent to the VLOG -BL command.
- PVCS_LOG_NEWEST
  Displays the newest revision on the trunk. This flag is equivalent to the VLOG -BN command.
- PVCS_LOG_NO_HEADER
  Does not display the archive header. This flag is equivalent to the VLOG -BR command.
- PVCS_LOG_NO_INDENT
  Does not indent the revision history. This flag is equivalent to the VLOG -I command.

- PVCS_LOG_NO_REVISIONS
  Does not display the revision history. This flag is equivalent to the VLOG -B command.
- PVCS_LOG_VERSIONS
  Displays version labels only. If the *version* parameter is specified, displays only that version. This flag is equivalent to the VLOG -BV command.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values"Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- To get information in a format suitable for program manipulation, use **PvcsGetArchiveInfo** or **PvcsGetRevisionInfo** instead.
- To specify a range for the *revision* or *date* parameter, use the form *first_value*last_value*.
- If you use both the *revision* and *version* parameters, the *revision* parameter supersedes the *version* parameter.

Example

```
char *outFile = "davee.lck";
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Generate report of all locks by "DAVEE" on revs checked */
/* in during 1993 for archives matching "*.c_v" */
status = PvcsLog(ARCHIVEHANDLE_NOT_OPEN,
   "*.c_v", /* Archive name */
   NULL,    /* No revision restrictions */
   NULL,    /* No version restrictions */
   NULL,    /* No ownership restrictions */
   NULL,    /* No author restrictions */
   "DAVEE", /* Limit to revisions locked by
               DAVEE */
   NULL,    /* No group restrictions */
   "93",    /* Revisions checked in in 1993 */
   outFile, /* File to receive report */
   PVCS_LOG_LOCKERS);  /* Display list of locked revisions */

if (!status)
   printf("Created lock report in \"%s\".\n", outFile);
```

Related Functions

PvcsGetArchiveInfo on page 85
PvcsGetRevisionInfo on page 103
PvcsOpenArchive on page 129

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Viewing revision information	*VLOG command*
Specifying revision ranges	*Revision Specification*
Specifying date ranges	*Date and Time Specification*
Wildcard expansion	*Wildcards*
Specifying archive and workfile names	*File Specification*

PvcsLogin

This function validates user IDs and passwords against the access control database. If the login source is VLOGIN, then the user ID becomes the VCSID. If the login source is not VLOGIN, then **PvcsLogin** does not affect Developer's Toolkit behavior. This function does not replace the VLOGIN command, but sets the user ID for the life of the Developer's Toolkit DLL.

NOTE If an access control database or LDAP authentication is associated with the Client Name (path map) on the Version Manager File Server, you must use the PvcsLogin function or an environment variable to present a user ID and password for validation. See, Chapter 1, "Using DTK Applications with a Version Manager File Server" on page 15.

NOTE The login source and access control database path can be set using VCONFIG or by using directives in a configuration file. If you set these using the configuration file, you must call **PvcsQueryConfiguration** before calling **PvcsLogin**.

Syntax

```
PvcsLogin(
   unsigned char *userid,/* Input */
   unsigned char *passwd,/* Input */
   void *reserved)  /* Input */
```

Parameters

userid	Pointer to a string that contains the user ID as defined in the access control database.
passwd	Pointer to a string that contains the password as defined in the access control database.
reserved	Reserved for future use.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_NO_ACCESS_DB
PVCS_E_UNKNOWN_USER
PVCS_E_INVALID_PASSWORD
PVCS_E_CANT_OPEN_ACCESSDB
PVCS_E_BAD_ACCESS_DB
PVCS_E_USER_EXPIRED
PVCS_E_INVALID_PARAMETER
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
char *name = "HANKB";
char *password = "BOBO";
char msgBuffer[128];
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Set the user ID */
   status = PvcsLogin(
   name,    /* User ID */
   password,/* User password */
   NULL);   /* Reserved */

if (!status)
   printf("User \"%s\" has been validated.\n", name);
```

Related Functions

PvcsGetUserInfo on page 110
PvcsIsUserInDatabase on page 114
PvcsQueryUserAccess on page 150

PvcsMakeDB

This function creates an access control database file. It is equivalent to the MAKEDB command.

Syntax

```
PvcsMakeDB(
   unsigned char *databaseName,/* Input */
   unsigned char *file,/* Input */
   void *reserved)  /* Input */
```

Parameters

databaseName	Pointer to a string that contains the name of the resulting access control database. If this parameter is null, then Developer's Toolkit uses the file name embedded by the VCONFIG -A command or specified by the AccessDB directive.
file	Pointer to a string that contains the name of the input text file.
reserved	Reserved for future use. Must be null.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVCS_E_FILE_NOT_FOUND
PVCS_E_CANT_OPEN_ACCESSDB

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
   char *databaseName = "access.db";
   char *listFile = "access.txt";
   void *reserved = (void *)0;
```

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Create access control database */
status = PvcsMakeDB(
   databaseName,/* Name of access control database */
   listFile,/* Name of input text file */
   reserved);/* Reserved for future use */

if (!status)
    printf("Created access control database \"%s\".\n",
        databaseName);
```

Related Functions
PvcsAccessOpenDB on page 40
PvcsAccessCloseDB on page 22
PvcsReadDB on page 152

PvcsMerge

This function merges revisions that diverge from a common ancestor. It is equivalent to the VMRG command.

Syntax

```
PvcsMerge(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *baseFile,/* Input */
   unsigned char *baseRev,/* Input */
   unsigned char *branch1File,/* Input */
   unsigned char *branch1Rev,/* Input */
   unsigned char *branch2File,/* Input */
   unsigned char *branch2Rev,/* Input */
   unsigned char *outFile,/* Input */
   PVCS_FLAGS mergeFlags)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
baseFile	Pointer to a string that contains the file name of the base point of the merge, which is usually the archive. If the parameter is a workfile and *baseRev* is not null, the function infers the archive name.
baseRev	Pointer to a string that contains the revision number or version label of the base point. This parameter is equivalent to the VMRG -R or VMRG -V command.
branch1File	Pointer to a string that contains the name of the archive or workfile that contains the first branch revision. If this string is null, the function uses the base file name.
branch1Rev	Pointer to a string that contains the revision number or version label of the revision on the first branch. If this string is null, the function uses the tip revision. This parameter is equivalent to the VMRG -R or VMRG -V command.

branch2File — Pointer to a string that contains the name of the archive or workfile of the second branch revision. If this string is null, the function uses the base file name.

branch2Rev — Pointer to a string that contains the revision number or version label of the revision on the second branch. If this string is null, the function uses the tip revision. This parameter is equivalent to the VMRG -R or VMRG -V command.

outFile — Pointer to a string that contains the file name that the function uses to create the merged file. If this parameter is null, the function uses *branch2File*. If *branch2File* is an archive, the function uses the workfile name. This parameter is equivalent to the VMRG -O command.

mergeFlags — Bit field that controls the operation of this function. Values include:

- PVCS_MERGE_AUTO
  Performs an automatic merge. This flag is equivalent to the VMRG -A command.
- PVCS_MERGE_NO_OVERWRITE
  Disallows overwriting if *outFile* exists. The function returns the value PVCS_E_OVERWRITE. This flag is equivalent to the VMRG -N command.
- PVCS_MERGE_OVERWRITE
  Allows overwriting if *outFile* exists. This flag is equivalent to the VMRG -Y command.
- PVCS_MERGE_STDOUT
  Writes the merge file to standard output. Do not use this flag in conjunction with the *outFile* parameter. This flag is equivalent to the VMRG -XO command.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- The *baseFile*, *branch1File*, and *branch2File* parameters correspond to the VMRG command-line parameters of the same names.
- If *branch1Rev* and *branch2Rev* are revisions in the archive *baseFile*, then *branch1File*, and *branch2File* can be null.

Example

```
char *outFile = "foo.mrg";
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Merge the following revisions from archive "foo.c_v": */
/* Base Rev - rev with version label "Beta Release 1.0" */
/* Branch 1 Rev - tip revision of archive */
/* Branch 2 Rev - rev with version label "Daves_Branch" */
```

```
/* Create merge output file "foo.mrg"*/
status = PvcsMerge(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",/* File name of the merge base */
   "Beta Release 1.0", /* Version of the merge base */
   NULL,    /* Use base file name for Branch 1 file
   NULL,    /* Use tip as Branch 1 revision */
   NULL,    /* Use base file name for Branch 2 file
   "Daves_Branch",/* Use ver label as Branch 2 rev */
   outFile, /* Merge output file */
   PVCS_MERGE_OVERWRITE);/* Overwrite existing output file */

if (!status)
   printf("Created merged output in \"%s\".\n", outFile);
```

Related Functions
PvcsOpenArchive on page 129
PvcsMerge2 on page 126

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Merging	*Merging*
How to merge	*VMRG command*
Specifying archive and workfile names	*File Specification*

PvcsMerge2

This function is equivalent to the VMRG command with two additional parameters. It notifies you of the percentage of completion and the number of merge conflicts it finds.

Syntax

```
PvcsMerge2(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *baseFile,/* Input */
   unsigned char *baseRev,/* Input */
   unsigned char *branch1File,/* Input */
   unsigned char *branch1Rev,/* Input */
   unsigned char *branch2File,/* Input */
   unsigned char *branch2Rev,/* Input */
   unsigned char *outFile,/* Input */
   int *pConflictCount,/* Output */
   PVCS_CALLBACKFUNC_MERGE_STATUS *pStatusFunc,
   PVCS_FLAGS mergeFlags)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
baseFile	Pointer to a string that contains the file name of the base point of the merge, which is usually the archive. If the parameter is a workfile and *baseRev* is not null, the function infers the archive name.

baseRev	Pointer to a string that contains the revision number or version label of the base point. This parameter is equivalent to the VMRG -R or VMRG -V command.
branch1File	Pointer to a string that contains the name of the archive or workfile that contains the first branch revision. If this string is null, the function uses the base file name.
branch1Rev	Pointer to a string that contains the revision number or version label of the revision on the first branch. If this string is null, the function uses the tip revision. This parameter is equivalent to the VMRG -R or VMRG -V command.
branch2File	Pointer to a string that contains the name of the archive or workfile of the second branch revision. If this string is null, the function uses the base file name.
branch2Rev	Pointer to a string that contains the revision number or version label of the revision on the second branch. If this string is null, the function uses the tip revision. This parameter is equivalent to the VMRG -R or VMRG -V command.
outFile	Pointer to a string that contains the file name that the function uses to create the merged file. If this parameter is null, the function uses *branch2File*. If *branch2File* is an archive, the function uses the workfile name. This parameter is equivalent to the VMRG -O command.
pConflictCount	Pointer to an integer that receives the number of merge conflicts.
pStatusFunc	Pointer to a callback function that displays the percent complete. Developer's Toolkit calls this function periodically, passing it an integer ranging from 0 to 100. Specify a null pointer if you don't want Developer's Toolkit to call a function.
mergeFlags	Bit field that controls the operation of this function. Values include: ■ PVCS_MERGE_AUTO Performs an automatic merge. This flag is equivalent to the VMRG -A command. ■ PVCS_MERGE_NO_OVERWRITE Disallows overwriting if *outFile* exists. The function returns the value PVCS_E_OVERWRITE. This flag is equivalent to the VMRG -N command. ■ PVCS_MERGE_OVERWRITE Allows overwriting if *outFile* exists. This flag is equivalent to the VMRG -Y command. ■ PVCS_MERGE_STDOUT Writes the merge file to standard output. Do not use this flag in conjunction with the *outFile* parameter. This flag is equivalent to the VMRG -XO command

Return Values The return value is zero if the function is successful. Other values may be:
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_FILE_BUSY

PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- The *baseFile*, *branch1File*, and *branch2File* parameters correspond to the VMRG command-line parameters of the same names.
- If *branch1Rev* and *branch2Rev* are revisions in the archive *baseFile*, then *branch1File*, and *branch2File* can be null.

Example

```
int         rc;
PVCS_PUCHAR szOutfile = NULL;
PVCS_PUCHAR szBaseFile = NULL;
PVCS_PUCHAR szBaseRev = NULL;
PVCS_PUCHAR szBranch1File = NULL;
PVCS_PUCHAR szBranch1Rev = NULL;
PVCS_PUCHAR szBranch2File = NULL;
PVCS_PUCHAR szBranch2Rev = NULL;
int         nConflictCount;

/* Initialize configuration settings */
rc = PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/*
 * PvcsMerge2: merge two files into a third given a commom
 * base. This code merges two revisions in single archive
 * from base revision in the archive. If file names for
 * branch1 or branch2 not specified, it uses the same name
 * as the base. To merge workfiles, set the revision
 * arguments to NULL. It also registers a callback function
 * to print percentage complete.
 */
rc =PvcsMerge2(
   ARCHIVEHANDLE_NOT_OPEN,
   szBaseFile,
   szBaseRev,
   szBranch1File,
   szBranch1Rev,
   szBranch2File,
   szBranch2Rev,
   szOutfile,
   &nConflictCount,
   merge2_monitor,
   0);
if (!rc)
   printf("    Conflicts:  %d\n",nConflictCount);

/* Callback procedure */
int PVCS_CALLBACK
merge2_monitor(int n, char *msg)
{
   printf("%d%% complete: %s\n",n,msg);
   return PVCS_OK;
}
```

Related Functions

PvcsOpenArchive on page 129
PvcsMerge on page 124

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Merging	*Merging*
How to merge	*VMRG command*
Specifying archive and workfile names	*File Specification*

PvcsOpenArchive

This function opens an archive and returns an archive handle to be used by subsequent operations. The archive remains open until you call **PvcsCloseArchive**, **PvcsCloseAll**, or **PvcsCancelUpdate**.

Syntax

```
PvcsOpenArchive(
   unsigned char *fileName,/* Input */
   unsigned char *workfile,/* Output */
   unsigned short workfileLen,/* Input */
   unsigned char *archive,/* Output */
   unsigned short archiveLen,/* Input */
   PVCS_FLAGS openFlags,/* Input */
   ARCHIVEHANDLE *hArchive)/* Output */
```

Parameters

fileName	Pointer to the name of an archive or a workfile. If this parameter is an archive, the function finds the corresponding workfile.
workfile	Pointer to a buffer to receive the name of the workfile.
workfileLen	Length of the *workfile* buffer.
archive	Pointer to a buffer to receive the name of the archive.
archiveLen	Length of the *archive* buffer.
openFlags	Bit field that controls the operation of this function. Values include: ■ PVCS_OPEN_RDONLY Opens the archive in read-only mode. If you do not call any functions that modify the archive, it is much more efficient to use the PVCS_OPEN_RDONLY flag. If you later attempt an update operation on the archive, you will get the *PVCS_E_BAD_ARCHIVE_HANDLE* error. This flag is the default. ■ PVCS_OPEN_UPDATE Opens the archive for modification, using normal archive semaphores. No other update access will be permitted to the archive while it is open.
hArchive	Pointer to a variable that receives the archive handle. This handle is used in other Serena PVCS Professional Suie services.

Return Values The return value is zero if the function is successful. Other values may be:
PVCS_E_ACCESS_VIOLATION

```
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_BAD_ARCHIVE_HANDLE
PVCS_E_BUFFER_OVERFLOW
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- It is not necessary to use this function to open an archive before processing it. If you use services that access archives with an archive name rather than an archive handle, they open and close the archives themselves. However, if you are performing multiple operations on an archive, it is more efficient to open the archive first and pass the archive handle to other services.
- Leaving an archive open may keep other users from accessing the archive, especially if it was opened for updating.
- The *archive* and *workfile* buffers must be long enough to contain the full path names. Specify null parameters if you do not want this information.

Example

```
ARCHIVEHANDLE handle;
char workfile[256];
char archive[256];
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Open archive for update */
PvcsOpenArchive(
   "foo.c_v",/* Name of archive or workfile */
   workfile,/* Buffer receiving name of workfile */
   (workfile),/* Length of workfile buffer */
   archive, /* Buffer receiving name of archive */
   sizeof(archive),/* Length of archive buffer */
   PVCS_OPEN_UPDATE,/* Open archive for modification */
   &handle);/* Returned archive handle */

/* Close archive */
PvcsCloseArchive(handle);
```

Related Functions

PvcsCancelUpdate on page 56
PvcsCloseAll on page 62
PvcsCloseArchive on page 63

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
File semaphores	*Semaphores*
Specifying archive locations	*VCSDir directive*

PvcsPromoteRevision

This function promotes a revision to the next highest level in the promotion model. It is equivalent to the VPROMOTE command.

Syntax

```
PvcsPromoteRevision(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *group,/* Input */
   PVCS_FLAGS flags)/* Input */
```

Parameters

hArchive — Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.

fileName — Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open.

group — Name of the group to promote from. A revision at this promotion level must exist in the archive. This is equivalent to the VPROMOTE -G command.

flags — Bit field that controls the operation of this function. Values include:

- PVCS_PROMOTE_NO_XBRANCH
  Denies permission to promote a revision across a branch boundary.
- PVCS_PROMOTE_XBRANCH
  Grants permission to promote a revision across a branch boundary.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_FILE_BUSY
PVCS_E_LOCKED_REVISION
PVCS_E_INVALID_PARAMETER
PVCS_E_INVALID_PROMO
PVCS_E_NO_GROUP
PVCS_E_PROMOTE_XBRANCH

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- If the specified promotion promotes a revision across a branch, and you don't specify one of the flags PVCS_PROMOTE_XBRANCH or PVCS_PROMOTE_NO_XBRANCH, **PvcsPromoteRevision** prompts for permission to promote across the branch. If you specify the PVCS_PROMOTE_NO_XBRANCH flag, **PvcsPromoteRevision** does not promote the revision across a branch, and returns PVCS_E_PROMOTE_XBRANCH.
- This function replaces **PvcsPromote**. **PvcsPromoteRevision** is similar to **PvcsPromote**, but takes the *flags* argument. For compatibility with previous versions, Developer's Toolkit still supports **PvcsPromote**.

Example

```
unsigned short grpCount = 1;
```

```
char grpBuffer[512];
char *archive = "foo.c_v";
char *group = "SOFTDEV";
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* See if group exists in archive */
status = PvcsGroupToRevision(ARCHIVEHANDLE_NOT_OPEN,
   archive, /* Name of archive or workfile */
   group,   /* Promotion group */
   &grpCount,/* Returns number of pairs retrieved */
   grpBuffer,/* Buf that receives
            revision/group pairs */
   sizeof(grpBuffer));/* Length of the buffer */

/* If so, then promote it */
if (status != PVCS_E_PROMO_NO_NODE)
   PvcsPromoteRevision(ARCHIVEHANDLE_NOT_OPEN,
   archive, /* Archive name */
   group,   /* Group to promote */
   PVCS_PROMOTE_NO_XBRANCH);/* Don't promote across */
            /* branch */
```

Related Functions

PvcsGetPromoParent on page 99
PvcsGroupToRevision on page 111
PvcsVerifyPromoTree on page 177
PvcsVerifyPromoTreeNodeExist on page 178

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Promotion models	*Promotion*
Defining a promotion group	*Promote directive*
Promoting revisions	*VPROMOTE command*
Specifying archive and workfile names	*File Specification*

PvcsPutExtRevAttribute

This function associates an extended attribute with a revision, or modifies an existing extended attribute. Use extended attributes to attach free-form binary or text data to a revision. The attribute contains a user-defined keyword, which you can use to access the data at a later time.

Syntax

```
PvcsPutExtRevAttribute(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *revarg,/* Input */
   unsigned char *extAttribute,/* Input */
```

```
unsigned short extAttrCount)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open.
revarg	Pointer to a string that contains a revision number or version label. The function associates the extended attribute with this revision. If the version label begins with a number, precede it with a backslash (\). A null parameter defaults to the tip revision on the trunk.
extAttribute	Pointer to a buffer that contains an extended attribute. See below for the buffer format.
extAttrCount	The number of extended attribute records present.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
PVCS_E_USER_ABORTED
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- The format of the *extAttribute* buffer is as follows:

Length
Keyword
Null
User_data

Length	Two-byte unsigned value that is the length of the keyword and null terminator, plus the length of the user data. The length does not include the length field itself.
Keyword	Null-terminated string that contains a user-defined keyword.
User_data	Free-format data, which may be text or binary.

- The maximum length of the extended attribute record (including the length field) is 64K.
- If the extended attribute keyword already exists, this function replaces it. To avoid inadvertently replacing an attribute, call **PvcsGetExtRevAttribute** before using this function to verify that the attribute does not already exist.

Example

```
char *keyword = "BetaTextKey";
```

```
char *userdata = "This Beta release was sent out to 500 customers \
as a precursor to the General Availability release.";
int extAttCount = 1;
char *archive = "foo.c_v";
char *revision = "Beta Release 1.0";
char getExtAttBuf[256];
char *bufPtr;
char *ptr;
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* See if extended attribute keyword already exists */
status = PvcsGetExtRevAttribute(ARCHIVEHANDLE_NOT_OPEN,
   archive, revision, keyword, getExtAttBuf,
   sizeof(getExtAttBuf));

/* If keyword exists, indicate that user data will be replaced */
if (!status)
   printf("Replacing data for keyword \"%s\".\n", keyword);

/* Set up extended attribute buffer */
   extAttBuf.length = strlen(keyword) + strlen(userdata) + 2;
   strcpy(extAttBuf.data, keyword);
   ptr = strchr(extAttBuf.data, '\0');
   strcpy(ptr + 1, userdata);
   bufPtr = (char *)&extAttBuf;

/* Add extended attribute */
status = PvcsPutExtRevAttribute(ARCHIVEHANDLE_NOT_OPEN,
   archive,
   revision,
   bufPtr,
   extAttCount);
```

Related Functions
PvcsGetExtRevAttribute on page 93
PvcsOpenArchive on page 129

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Specifying archive and workfile names	*File Specification*
Specifying revision numbers	*Revision Numbering*

PvcsPutRevision

This function checks a new revision into an archive. It is equivalent to the PUT command.

Syntax

```
PvcsPutRevision(
   ARCHIVEHANDLE hArchive,/* Input */
```

```
  unsigned char *locker,/* Input */
  unsigned char *fileName,/* Input */
  unsigned char *workfileName,/* Input */
  unsigned char *revarg,/* Input */
  unsigned char *changeDescr,/* Input */
  unsigned char *fileDesc,/* Input */
  unsigned char *versionLabel,/* Input */
  unsigned char *groupName,/* Input */
  unsigned char *extAttribute,/* Input */
  unsigned short extAttrCount,/* Input */
  PVCS_FLAGS putFlags)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
locker	Pointer to a string that contains the user ID of the person who locked the revision. If this parameter is null, the function uses the current user ID. If the PVCS_PUT_RELOCK flag is set, the function relocks the revision using the same user ID.
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open.
workfileName	Pointer to the name of a workfile. If this parameter is null, the function obtains the workfile name from the archive. If you specify a workfile name, then *fileName* must contain an archive name.
revarg	Pointer to a string that contains a revision number, which overrides the default revision numbering scheme. This parameter is equivalent to the PUT -R command.
changeDescr	Pointer to a string that contains the description of the change. This parameter is equivalent to the PUT -M command. *When the archive exists*, if the parameter is null or an empty string, the Toolkit prompts for a change description. *When the archive does not exist*, if the parameter is an empty string, the Toolkit prompts for a change description. *When the archive does not exist*, if the parameter is null, the Toolkit uses the "Initial Revision" for the change description.
fileDesc	Pointer to a string that contains the description of the workfile, used at initial check in only. This parameter is equivalent to the PUT -T command. *When the archive exists* the *fileDesc* parameter is ignored. *When the archive does not exist*, if the *fileDesc* parameter is null or an empty string, the Toolkit prompts for a file description.
versionLabel	Pointer to a string that contains the version label to assign to the new revision. This parameter is equivalent to the PUT -V command.
groupName	Pointer to a string that contains the promotion group of the new revision. This parameter is equivalent to the PUT -G command.
extAttribute	Pointer to a buffer that contains an extended attribute record.

extAttrCount The number of extended attribute records present. This field must contain 0 or 1.

putFlags Bit field that controls the operation of this function. Values include:

- PVCS_PUT_AUTOCREATE
  Creates an archive before checking in the workfile, if one does not exist. This flag overrides the NoAutoCreate directive.
- PVCS_PUT_BRANCH_VERS
  Makes *versionLabel* a floating version label. This flag is equivalent to the -*vfloating_version*:* variant of the PUT -V command.
- PVCS_PUT_DELETE
  Deletes the workfile after check in. This flag overrides the NoDeleteWork directive.
- PVCS_PUT_FORCE_BRANCH
  Creates a new branch. This flag is equivalent to the PUT -FB command.
- PVCS_PUT_FORCE_PUT
  Checks in the file, even if it is unchanged. This flag is equivalent to the PUT -F command.
- PVCS_PUT_KEEP_MSGFILE
  Does not delete the message file. This flag is equivalent to the NoDeleteMessageFile directive.
- PVCS_PUT_NO_REPLACE_VERS
  Does not overwrite an existing version label. This flag is equivalent to the PUT -N command.
- PVCS_PUT_NOAUTOCREATE
  Does not create an archive if one does not exist.
- PVCS_PUT_NODELETE
  Does not delete the workfile. This flag is equivalent to the PUT -U command.
- PVCS_PUT_NOFORCE_PUT
  Does not allow the workfile to be checked in if it is older than the parent, or is unchanged.
- PVCS_PUT_NOPROMPT
  Suppresses prompts. If Version Manager requires input from the user to complete the operation, the function returns the error code PVCS_E_PROMPT, and the operation fails.
- PVCS_PUT_NOWHITE
  Ignores leading and trailing white space when determining whether the file has changed. This flag is equivalent to the PUT -B command.

- PVCS_PUT_RELOCK
  Checks the revision back out with a lock. This flag is equivalent to the
  PUT -L command.
- PVCS_PUT_REPLACE_VERS
  Overwrites an existing version label. This flag is equivalent to the PUT -Y command.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_ACCESS_DENIED
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_BAD_FILENAME
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
PVCS_E_USER_ABORTED
PVCS_E_VERSION_EXISTS
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- Use the *workfileName* parameter when you are checking in a workfile with a different name than the name stored in the archive. This is equivalent to the following command-line example:

  ```
  put inv.c_v(d:\temp\test.c)
  ```

  In this example, *fileName* is INV.C_V and *workfileName* is D:\TEMP\TEST.C.
- Either the *changeDesc* or *fileDesc* parameter may contain *@fileName,* which causes the program to read the description from *fileName*. If either parameter is null, the program prompts the user for a description.
- If *versionLabel* already exists in the archive, the program prompts the user for permission to overwrite it, unless you use the PVCS_PUT_REPLACE_VERS or PVCS_PUT_NO_REPLACE_VERS flag.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Check in a revision, assign a version label, and check out */
/*  the revision with a lock                                  */
PvcsPutRevision(ARCHIVEHANDLE_NOT_OPEN,
   "DAVEE", /* User ID of locker */
   "foo.c_v",/* Archive name */
   NULL,    /* Infer workfile name */
   NULL,    /* Use default revision numbering scheme */
   "Fixed a killer bug.",/* Change description */
   NULL,    /* Assume archive exists */
   "Beta Release 1.0",/* Version label to assign */
   NULL,    /* Assume no promo hierarchy */
   NULL,    /* Assume no ext rev attrib */
   0,
   PVCS_PUT_RELOCK);/* Check out with a lock */
```

Related Functions
PvcsGetRevision on page 100
PvcsOpenArchive on page 129
PvcsPutExtRevAttribute on page 132

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Checking revisions into archives	*PUT command*
Specifying archive and workfile names	*File Specification*

PvcsQueryAlias

This function returns the current value of an alias that was defined using **PvcsAddAlias**. This is equivalent to referencing an alias that has been defined by the Alias directive.

Syntax
```
PvcsQueryAlias(
   unsigned char *aliasName,/* Input */
   unsigned char *aliasValue,/* Output */
   unsigned short bufLen)/* Input */
```

Parameters

aliasName	Pointer to the name of the alias to query.
aliasValue	Pointer to the buffer that is to contain the alias value.
bufLen	Length of the alias buffer.

Return Values This function returns zero if successful. Other values may be:
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER
PVCS_E_NO_ALIAS

See Chapter 4, "Return Values" for descriptions of return values.

Example
```
char *name = "SourceFiles";
char *value = "alpha.c beta.c gamma.c";
char aliasBuf[64];
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Add an alias definition */
PvcsAddAlias(
   name,    /* Alias name */
   value);  /* Alias value */

/* Retrieve newly defined alias */
status = PvcsQueryAlias(
   name,    /* Alias name */
   aliasBuf,/* Buffer receiving alias value */
```

```
   sizeof(aliasBuf));/* Length of buffer */

if (!status)
  printf("Alias name \"%s\" is defined to be \"%s\".\n", name,
    aliasBuf);
```

Related Function: PvcsAddAlias on page 50

Related Topics: For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
How to use aliases	*Using Aliases*

PvcsQueryArchiveAccess

This function checks to determine if a user can perform an action on a specific archive.

Syntax

```
PvcsQueryArchiveAccess(
   unsigned char *archive,/* Input */
   unsigned char *user,/* Input */
   char *privilege, /* Input */
   int *pStatus,    /* Output */
   void *reserved)  /* Reserved */
```

Parameters

archive	Pointer to a string that contains the name of the archive.
user	Pointer to a string that contains the name of the user.
privilege	Pointer to a string that contains the name of the base privilege.
pStatus	The variable that receives the access status. A non-zero value indicates that the user has access.
reserved	Reserved for future use. Must be null.

Return Values: The return value is zero if the function is successful. Other values may be:
PVCS_E_INVALID_PARAMETER
PVDS_E_UNKNOWN_USER

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* PvcsQueryArchiveAccess */
PVCS_PUCHAR szArchive=NULL;
PVCS_PUCHAR szUser=NULL;
char *szPrivilege="GETTIP";
int nAccessGranted=0;
int rc;

/* Initialize configuration settings */
rc = PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);
```

```
/*
 * PvcsQueryArchiveAccess: this procedure checks the
 * access list internal to the archive and the access db to
 * see if the given user is allowed access
 * to the archive for the specified priv. nAccessGranted is 0
 * for denied, non-0 for granted.
 */

rc = PvcsQueryArchiveAccess(
   szArchive,
   szUser,
   szPrivilege,
   &nAccessGranted,
   NULL);
```

Related Function PvcsChangeAccessList on page 57

PvcsQueryConfiguration

This function reads a Version Manager configuration file and fills in the CONFIG structure. Call this function before calling other Serena PVCS Professional Suite functions if you want to use directives from a configuration file to control the operation of the functions. For details on the CONFIG structure, see Chapter 5, "Data Structures"

Syntax

```
PvcsQueryConfiguration(
   unsigned char *configFile,/* Input */
   CONFIG *pConfig, /* Output */
   unsigned short bufLen,/* Input */
   PVCS_FLAGS configFlags)/* Input */
```

Parameters

configFile	Pointer to the name of the configuration file to process. The file name cannot contain wildcards.
pConfig	Pointer to the CONFIG structure that receives the configuration data. You must be sure to pad extra space in the CONFIG structure to store string values (512 bytes). If *pConfig* is null, Developer's Toolkit does not return any data, although it still stores the values. The function reads and processes the configuration file, but does not require you to allocate a CONFIG structure.
bufLen	Variable that contains the length of the CONFIG data buffer.
configFlags	Bit field that controls the operation of this function. Values include: ■ PVCS_CONFIG_APPEND Appends the configuration settings to the current configuration. ■ PVCS_CONFIG_MASTER_FILE Treats the specified file as the master configuration file. The function will not read any other master configuration file unless explicitly directed.

- PVCS_CONFIG_NO_MASTER
  Does not read the master configuration file embedded into the DLL using VCONFIG.
- PVCS_CONFIG_OVERWRITE
  Sets the CONFIG structure to the default settings before processing the configuration file.
- PVCS_CONFIG_READ_MASTER
  If you use VCONFIG to embed a master configuration file into the DLL, Developer's Toolkit reads it unconditionally before reading the specified file.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_ACCESS_DENIED
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_INVALID_CONFIG_PARM
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If you don't call this function, Developer's Toolkit ignores configuration files, and all directives retain their default values.
- If you specify a null *pointer* for the *configFile* parameter, the function finds the default configuration file using the normal Version Manager rules. If you specify a null *string*, the function does not read a configuration file and sets the configuration structure to the default settings.
- To process a second configuration file, set *configFlags* to PVCS_CONFIG_APPEND. This flag appends the second configuration file to the current configuration. If you don't use this flag, the function sets all directives to their defaults before reading the configuration file.

Example

```
#define CONFIG_PAD 1024
CONFIG *config;
int status;

/* Allocate buffer to hold configuration information */
config = (CONFIG *)malloc(sizeof(CONFIG) + CONFIG_PAD);

/* Process configuration file */
status = PvcsQueryConfiguration(NULL,
   config,
   sizeof(CONFIG) + CONFIG_PAD,
   PVCS_CONFIG_OVERWRITE);

if (!status) {
   printf("Quiet:                        %s\n",
            config->quiet ? "on" : "off");
   printf("WriteProtect:                 %s\n",
            config->wrtProtArchive ? "on" : "off");
   printf("CheckLock:                    %s\n",
            config->checkLock ? "on" : "off");
   printf("BranchWarn:                   %s\n",
            config->branchWarn ? "on" : "off");
   printf("IgnorePath:                   %s\n",
            config->ignorePath ? "on" : "off");
```

```
printf("ForceUnlock:                  %s\n",
        config->forceUnlock ? "on" : "off");
printf("AutoCreate:                   %s\n",
        config->autoCreate ? "on" : "off");
printf("firstMatch:                   %s\n",
        config->firstMatch ? "on" : "off");
printf("DeleteMessageFile:            %s\n",
        config->delMsgFile ? "on" : "off");
printf("Semaphore: Local              %s\n",
        config->semaphoreLocal ? "on" : "off");
printf("Semaphore: Network            %s\n",
        config->semaphoreNetwork ? "on" : "off");
printf("ArchiveWork:                  %s\n",
        config->useArchiveWork ? "on" : "off");
printf("Signon:                       %s\n",
        config->signon ? "on" : "off");
printf("NoCase VCSID                  %s\n",
        config->ignIDCase ? "on" : "off");
printf("Share Local                   %s\n",
        config->shareLocal ? "on" : "off");
printf("Share Network:                %s\n",
        config->shareNet ? "on" : "off");
printf("MemSwap:                      %s\n",
        config->memswap ? "on" : "off");
printf("Ctrlz:                        %s\n",
        config->ctrlz ? "on" : "off");
printf("NodeleteWork NoWriteProtect: %s\n",
        config->writable_workfile ? "on" : "off");
printf("AccessDB NoWriteProtect:     %s\n",
        config->writable_accessdb ? "on" : "off");
printf("SemaphoreRetry:               %s\n",
        config->semRetry ? "on" : "off");
printf("SemaphoreDelay:               %s\n",
        config->semDelay ? "on" : "off");
printf("DeleteWork:                   %s\n",
        config->delWork ? "on" : "off");
printf("MultiLock User:               %s\n",
        config->multiLockUser ? "on" : "off");
printf("MultiLock Revision:           %s\n",
        config->multiLockRev ? "on" : "off");
printf("ExclusiveLock:                %s\n",
        config->exclusiveLock ? "on" : "off");
printf("Vlogin:                       %s\n",
        config->vloginInt ? "on" : "off");
printf("Internal diagnostic level: %ld\n", config->diag);
printf("PathSeparator:        %c\n", config->pathSep);
printf("Translate:            %s\n", config->translate);
printf("ArchivSuffix:         %s\n", config->archsuf);
printf("MessageSuffix:        %s\n", config->msgsuf);
printf("SemSuffix:            %s\n", config->semsuf);
printf("DefaultVersion:       %s\n", config->defVers);
printf("BaseVersion:          %s\n", config->baseVers);
printf("branchVersion:        %s\n", config->branchVers);
printf("SemaphoreDir:         %s\n", config->semDir);
printf("WorkDir:              %s\n", config->workDir);
printf("ArchiveWork:          %s\n", config-
```

```
            archiveWorkDir);
    printf("VCSEdit:              %s\n", config->vcsEdit);
    printf("VCSID:                %s\n", config->vcsid);
    printf("Owner:                %s\n", config->owner);
    printf("Journal:              %s\n", config->journal);
    printf("Newline:              %s\n", config->newline);
    printf("AccessDB:             %s\n", config->accessdb);
    printf("ColumnMask:           %s\n", config->columnMask);
    printf("Renumber:             %s\n", config->renum);
    printf("VCSDIR:               %s\n", config->vcsdir);
    printf("CompressWorkImage:    %s\n", config-
            >compressImage);
    printf("CompressDelta:        %s\n", config-
            >compressDelta);
    printf("RecordLength:         %s\n", config->recordLength);
    printf("ExpandKeywords:       %s\n", config->expKeywords);
    printf("Login:                %s\n", config->loginSource);
```

Related Functions
PvcsQueryConfigurationError on page 143
PvcsQueryConfigurationItem on page 144

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Reading configuration files	*Configuration Files*
Default configuration settings	*Directives*

PvcsQueryConfigurationError

This function returns information about an error in configuration file processing.

Syntax

```
PvcsQueryConfigurationError(
   unsigned char  *pCfgFile,/* Output */
   unsigned short cfgFileLen,/* Input */
   unsigned long  *pLineNumber,/* Output */
   unsigned char  *pCfgLine,/* Output */
   unsigned short cfgLineLen)/* Input */
```

Parameters

pCfgFile	Pointer to a buffer that receives the name of the configuration file that contained the error. If there was no error, the buffer contains a null string.
cfgFileLen	Length of the buffer receiving the file name.
pLineNumber	Pointer to a variable that receives the line number of the line that contained the error. If there was no error, the variable is set to zero.

pCfgLine	Pointer to a buffer that receives a copy of the line that contained the error. If there was no error, the buffer contains a null string.
cfgLineLen	Length of the buffer receiving the line that contains the error. The maximum length is 256.

Return Values

The return value is zero if the function is successful. Otherwise it returns PVCS_E_BUFFER_OVERFLOW. See Chapter 4, "Return Values" for descriptions of return values.

Example

```
CONFIG *config;
char *filename;
char *line;
long lineno;
int status;

/* Process configuration file */
config = (CONFIG *)malloc(sizeof(CONFIG) + CONFIG_PAD);
status = PvcsQueryConfiguration(NULL,
   config,
   sizeof(CONFIG) + CONFIG_PAD,
   PVCS_CONFIG_OVERWRITE);

if (status) {

/* Allocate buffers to hold configuration file name and line */
filename = malloc(256);
line = malloc(256);

/* Obtain information about error in configuration processing */
PvcsQueryConfigurationError(
   filename,/* Buffer receiving name of config file */
   256,     /* Length of buffer */
   &lineno, /* Variable receiving last line number */
   line,    /* Buffer receiving copy of last line */
   256);    /* Length of buffer */
   if (lineno)
            printf("Configuration error on line %ld of file
            %s:\n%s\n", lineno, filename, line);
}
```

Related Functions

PvcsGetErrorMessage on page 92
PvcsQueryConfiguration on page 140

PvcsQueryConfigurationItem

This function returns the current value of a directive.

Syntax

```
PvcsQueryConfigurationItem(
   unsigned short itemType,/* Input */
   unsigned char *pCfgData,/* Output */
   unsigned short bufLen)/* Input */
```

Parameters

itemType	Variable that specifies which configuration directive to query. See *Directive Values,* below, for possible values.
pCfgData	Pointer to a buffer that receives the requested value.
bufLen	Variable that contains the length of the data buffer.

Return Values

The return value is zero if the function is successful. Other values can be:
PVCS_E_BUFFER_OVERFLOW
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Special Consideration

This function does not read the configuration file. Call **PvcsQueryConfiguration** to read the configuration file and initialize the Developer's Toolkit internal copy of the configuration values.

Example

```
char *vcsdir;

/* Initialize configuration information */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Allocate buffer to hold VCSDir value */
vcsdir = malloc(256);

/* Obtain VCSDir setting */
PvcsQueryConfigurationItem(
   PVCS_CFGITEM_VCSDIR,/* Configuration item identifier */
   vcsdir,  /* Buffer to receive config value */
   256);    /* Length of buffer */

/* Display VCSDIR setting */
if (strlen(vcsdir))
   printf("VCSDIR value is \"%s\".\n", vcsdir);
else
   printf("VCSDIR is not set.\n");
```

Related Function

PvcsQueryConfiguration on page 140

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Version Manager directives	*Directives*

Directive Values

The lists below show possible values for the *itemType* parameter and the directive to which each corresponds. It also shows the data type returned in the buffer, depending on the value of the *itemType* parameter. Parameters are grouped by the data type they return.

Boolean Values

If you specify one of the following *itemType* values, the returned buffer contains an unsigned short integer Boolean value.

Directive	Value for *itemType*
AccessDB NoWriteProtect	PVCS_CFGITEM_WRITABLE_ACCESSDB
AutoCreate	PVCS_CFGITEM_AUTOCREATE
BranchWarn	PVCS_CFGITEM_BRANCHWARN
CheckLock	PVCS_CFGITEM_CHECKLOCK
CtrlZ	PVCS_CFGITEM_CTRLZ
DeleteMessageFile	PVCS_CFGITEM_DELMSGFILE
DeleteWork	PVCS_CFGITEM_DELWORK
ExclusiveLock	PVCS_CFGITEM_EXCLUSIVELOCK
FirstMatch	PVCS_CFGITEM_FIRSTMATCH
ForceUnlock	PVCS_CFGITEM_FORCEUNLOCK
IgnorePath	PVCS_CFGITEM_IGNOREPATH
MemSwap	PVCS_CFGITEM_MEMSWAP
MultiLock Revision	PVCS_CFGITEM_MULTILOCKREV
MultiLock User	PVCS_CFGITEM_MULTILOCKUSER
NoArchiveWork	PVCS_CFGITEM_NOLOGWORK
NoCase VCSID	PVCS_CFGITEM_IGNIDCASE
NoDeleteWork NoWriteProtect	PVCS_CFGITEM_WRITABLE_WORKFILE
Quiet	PVCS_CFGITEM_QUIET
ReferenceDir TrunkOnly	PVCS_CFGITEM_REFDIR_TRUNKONLY
ReferenceDir WriteProtect	PVCS_CFGITEM_REFDIR_WR_PROTECT
Screen	PVCS_CFGITEM_SCREEN
Screen Hercules	PVCS_CFGITEM_HERC_SCREEN
Share Local	PVCS_CFGITEM_SHARELOCAL
Share Network	PVCS_CFGITEM_SHARENET
SignOn	PVCS_CFGITEM_SIGNON
WriteProtect	PVCS_CFGITEM_WRTPROTARCHIVE

Integer Values

If you specify one of the following *itemType* values, the returned buffer contains an unsigned short integer value.

Directive	Value for *itemType*
Diagnostic	PVCS_CFGITEM_DIAG
Semaphore Local	PVCS_CFGITEM_SEMAPHORELOCAL

Semaphore Network	PVCS_CFGITEM_SEMAPHORENETWORK
SemaphoreDelay	PVCS_CFGITEM_SEMDELAY
SemaphoreRetry	PVCS_CFGITEM_SEMRETRY
VLogIn	PVCS_CFGITEM_VLOGININT

If you specify PVCS_CFGITEM_SEMAPHORELOCAL or PVCS_CFGITEM_SEMAPHORENETWORK, the value is an unsigned short integer that stands for one of the following:

PVCS_SEM_NONE	No semaphore
PVCS_SEM_FILE	File
PVCS_SEM_NETWARE	NetWare

Character Values

If you specify one of the following *itemType* values, the returned buffer contains a null-terminated character string. The maximum length is 256 bytes.

Directive	Value for *itemType*
AccessDB	PVCS_CFGITEM_ACCESSDB
AccessList	PVCS_CFGITEM_ACCESSLIST
ArchiveSuffix	PVCS_CFGITEM_ARCHSUF
ArchiveSuffix	PVDS_CFGITEM_ARCHSUFLIST
ArchiveWork	PVCS_CFGITEM_ARCHIVEWORKDIR
BaseVersion	PVCS_CFGITEM_BASEVERS
BranchVersion	PVCS_CFGITEM_BRANCHVERS
DefaultVersion	PVCS_CFGITEM_DEFVERS
Delta Delete	PVCS_CFGITEM_DELTA_DELETE
Delta Insert	PVCS_CFGITEM_DELTA_INSERT
Delta Replace	PVCS_CFGITEM_DELTA_REPLACE
Delta Seq	PVCS_CFGITEM_DELTA_SEQUENCE
EventTrigger	PVDS_CFGITEM_EVENTTRIGGER
Journal	PVCS_CFGITEM_JOURNAL
MessageSuffix	PVCS_CFGITEM_MSGSUF
Owner	PVCS_CFGITEM_OWNER
ReferenceDir	PVCS_CFGITEM_REFDIR
SemaphoreDir	PVCS_CFGITEM_SEMDIR
SemSuffix	PVCS_CFGITEM_SEMSUF
VCSDir	PVCS_CFGITEM_VCSDIR
VCSEdit	PVCS_CFGITEM_VCSEDIT
WorkDir	PVCS_CFGITEM_WORKDIR

Extension Values

Extensions are null-terminated strings and contain a leading period (.). A string that contains a single period indicates a directive that applies to files with no extension. The table is terminated by a null character. If there are no extensions for the *itemType* value, the list contains a single null character. For example, the list might contain:
.c<null>.h<null>.asm<null><null>

If you specify one of the following *itemType* values, the returned buffer contains a list of extensions for the corresponding directive.

Directive	Value for *itemType*
CompressDelta	PVCS_CFGITEM_COMPRESSDELTA
CompressWorkImage	PVCS_CFGITEM_COMPRESSIMAGE
ExpandKeywords	PVCS_CFGITEM_EXPKEYWORDS
NoCompressDelta	PVCS_CFGITEM_NOCOMPRESSDELTA
NoCompressWorkImage	PVCS_CFGITEM_NOCOMPRESSIMAGE
NoExpandKeywords	PVCS_CFGITEM_NOEXPKEYWORDS
NoTranslate	PVCS_CFGITEM_NOTRANSLATE
Translate	PVCS_CFGITEM_TRANSLATE

Extension/String Values

Extensions are null-terminated strings that contain a leading period (.). The table is terminated by a null character. If there are no extensions for the *itemType* value, the list contains a single null character. For example, the list for PVCS_CFGITEM_COMMENTPREFIX might contain:
.c<null>*<null>.bat<null>REM<null>.asm<null>; <null><null>

The following values for *itemType* return a list of extension/string value pairs. This type of directive assigns a different string value to each extension.

Directive	Value for *itemType*
ArchiveSuffix	PVCS_CFGITEM_ARCHSUFLIST
CommentPrefix	PVCS_CFGITEM_COMMENTPREFIX
NewLine	PVCS_CFGITEM_NEWLINE
RecordLength	PVCS_CFGITEM_RECORDLENGTH

Special Values

If you specify one of the following *itemType* values, the returned buffer contains a special value as described.

Directive	Value for *itemType*
Alias	PVCS_CFGITEM_ALIAS Returns a list of aliases defined by the Alias directive. The format of the list is a series of strings of the form *name=value*. Each element of the list is followed by a null character, and the list is terminated by a null character. If the list is empty, the list contains a single null character. This list contains only aliases defined by the Alias directive. There is currently no way to get a list of aliases defined on the command line or by environment variables. However, you can obtain the value for specific names by calling **PvcsQueryAlias**.

ColumnMask	PVCS_CFGITEM_COLUMNMASK
	Returns a list of extension/value pairs. The value is a null-terminated string with the following format:
	`start_column-end_column, start_column-end_column`
	For example:
	`.cob<null>1-6, 73-80<null><null>`

Disallow	PVCS_CFGITEM_DISALLOW Returns a list of directives that have been disallowed using the Disallow directive in the master configuration file. Each directive is a null-terminated string, and the list is terminated by a null character. For example: `SEMAPHORE<null>LOGIN<null><null>`
EventTrigger	PVCS_CFGITEM_EVENTTRIGGER Returns a sequence of pairs where the first string is an event name, and the second is the event command line (if it is registered). Each group is a null-terminated string, and the list is terminated by a null pointer. No event triggers are registered in the following example. `AllEvents\0\QPrePut\0\0 ... PrePromote \0\0PreVersionLabel\0\0\0`
ExpandKeywordsTouch	PVCS_CFGITEM_EXPANDKEYWORDS_TOUCH Returns a Boolean value that indicates the setting of the ExpandKeywords Touch directive. Zero indicates that the NoTouch option is in effect, which means that the timestamp of the workfile will not be updated if there are keywords in the workfile.
LogIn	PVCS_CFGITEM_LOGINSOURCE Returns a comma-separated list of login sources, terminated by a null character. For example: `netware,vcsid,unknown<null>`
PathSeparator	PVCS_CFGITEM_PATHSEP Returns a single character.

Directive	Value for *itemType*
Promote	PVCS_CFGITEM_PROMOTE Returns the promotion pairs *from_group* and *to_group*. Each group is a null-terminated string, and the list is terminated by a null. For example: `Dev<null>QA<null>QA<null>Prod<null><null>`

Renumber	PVCS_CFGITEM_RENUM
	Returns a list of extension/value pairs. The value is a null-terminated string in the following format:
	`start_column-end_column from start_value by step`
	For example:
	`.cob<null>1-6 from 10 by 10<null><null>`

VCSDir	PVCS_CFGITEM_VCSDIR Returns a semicolon-separated list of archive directories.
VCSDir	PVCS_CFGITEM_VCSDIR_REFDIR Returns a semicolon-separated list of archive directories. Unlike the string returned by PVCS_CFGITEM_VCSDIR, this string contains the reference directory name, if any. For example: `J:\proj1\arc(j:\proj1\source);J:\proj2\archives`

PvcsQueryUserAccess

This function checks a user's privileges for a specific action. It does not verify the user's access privileges to an archive.

Syntax

```
PvcsQueryUserAccess(
   unsigned char *userName,/* Input */
   char *privilege, /* Input */
   int *pStatus,    /* Output */
   void *reserved)  /* Reserved */
```

Parameters

userName	Pointer to a string that contains the name of the user.
privilege	Pointer to a string that contains the base privilege.
pStatus	The variable that receives the privilege status. A non-zero value indicates that the user has the privilege.
reserved	Reserved for future use. Must be null.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_INVALID_PARAMETER
PVCS_E_UNKNOWN_USER
PVCS_E_UNKNOWN_PRIVILEGE
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
unsigned char *userName = "DAVEE";
char *privilege = "GETTIP";
int status;
void *reserved = (void *)0;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* See if user has permission to perform an operation */
status = PvcsQueryUserAccess(
   userName,/* Name of user */
   privilege,/* Name of privilege */
   &status, /* Receives privilege status */
   reserved);/* Reserved for future use */

if (!status && pStatus)
   printf("User \"%s\" has permission to perform \"%s\".\n",
            userName, privilege);
```

Related Functions

PvcsLogin on page 122
PvcsGetUserInfo on page 110
PvcsIsUserInDatabase on page 114

PvcsQueryVconfigItem

This function returns the current value of a VCONFIG setting.

Syntax

```
PvcsQueryVconfigItem(
   unsigned char *DLLName,/* Input */
   unsigned short itemCode,/* Input */
   unsigned char *pVcfgData,/* Output */
   unsigned short bufLen)/* Input */
```

Parameters

DLLName Pointer to a string that specifies the configured DLL file:

- **Windows**: VMWFVC.DLL

The file name cannot contain wildcards. If *DLL_Name* is a null pointer, then the default DLL file name is used. If it is a null string, then Developer's Toolkit returns an error.

itemCode The variable that specifies which item to query. Values include:

- PVCS_VCFGITEM_ACCESS_DB
  Specifies the access control database.
- PVCS_VCFGITEM_MASTER_CFG_FILE
  Specifies the master configuration file.
- PVCS_VCFGITEM_USER_ID_SOURCE
  Specifies the user identification source.
- PVCS_VCFGITEM_LANGUAGE
  Specifies the default language.

- PVCS_VCFGITEM_MSG_FILE
  Specifies the path to the message file.
- PVCS_VCFGITEM_LOCALE_FILE
  Specifies the path to the locale file.

pVcfgData — Pointer to the buffer that receives the configuration value. This buffer contains a null-terminated character string. If this value is not set, then the buffer contains a null string.

bufLen — Length of the data buffer.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_INVALID_PARAMETER
PVCS_E_BUFFER_OVERFLOW
PVCS_E_BAD_SERIAL_NUMBER
PVCS_E_CANT_OPEN_VCONFIG_FILE
PVCS_E_BAD_VCONFIG_FILE
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Consideration

Note that under UNIX, the *DLL_Name* parameter is *library*.

Example

```
/* Initialize configuration information */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);
/* Allocate buffer to hold master configuration file name */
master_cfg = malloc(256);
/* Obtain master configuration file name */
PvcsQueryVconfigItem(
   (char *)0,/* Use default name */
   PVCS_VCFGITEM_MASTER_CFG_FILE,/* VCONFIG item id */
   master_cfg,    /* VCONFIG value */
   256);    /* Length of buffer */

/* Display master configuration file name */
if (strlen((char *)master_cfg))
   printf("Master configuration file is \"%s\".\n", master_cfg);
else
   printf("Master configuration file not set.\n");
```

Related Function

PvcsVconfig on page 175

PvcsReadDB

This function displays records from an access control database. It is equivalent to the READDB command and requires the ViewAccessDB privilege.

Syntax

```
PvcsReadDB(
   unsigned char *databaseName,/* Input */
   unsigned char *listFile,/* Input */
   void *reserved,  /* Input */
   PVCS_FLAGS flags)/* Input */
```

Parameters

databaseName	Pointer to a string containing the name of the access control database. If this parameter is null, then it reads the file name that was embedded by the VCONFIG -A command or specified by the AccessDB directive.
listFile	Pointer to a string containing the name of the output file.
reserved	Reserved for future use. Must be null.
flags	Bit field that controls the operation of this function. Value includes: ■ PVCS_READDB_DISPLAY_PASSWORD Displays all passwords in the database.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_INVALID_PARAMETER
PVCS_E_FILE_NOT_FOUND
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;
char *databaseName = "access.db";
char *listFile = "access.txt";
void *reserved = (void *)0;
FILE *stream;
char line[128];

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Create access control database */
status = PvcsReadDB(
   databaseName,/* Name of access control database */
   listFile,/* Name of output text file */
   reserved, /* Reserved for future use */
   0);      /* Do not write passwords to text file */

if (!status) {
   printf("Contents of \"%s\":\n", listFile);
   stream = fopen(listFile, "r");
   while (fgets(line, sizeof(line), stream) != NULL)
   printf("%s", line);
   fclose(stream);
   }
```

Related Functions

PvcsAccessCloseDB on page 22
PvcsAccessOpenDB on page 40
PvcsMakeDB on page 123

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Viewing the access control database	*READDB command*

PvcsRedirectOutput

This function redirects standard output and standard error messages to a file.

Syntax

```
PvcsRedirectOutput(
   unsigned char *fileName,/* Input */
   PVCS_FLAGS flags)/* Input */
```

Parameters

fileName Pointer to a string containing the name of an output file. This file will be overwritten if it already exists.

flags Bit field that controls the operation of this function. Values include:

- PVCS_REDIR_STDOUT
  Redirects messages that go to the standard output device. This includes most processing messages.
- PVCS_REDIR_STDERR
  Redirects messages that go to the standard error device. This includes the sign-on message and all error messages.

Return Values

The return value is zero if the function is successful. Other values may be:

PVCS_E_BAD_REDIRECT
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Special Consideration

You can use the flags PVCS_REDIR_STDOUT and PVCS_REDIR_STDERR together to redirect all messages to the same file.

Example

```
int status;

/* Keep a log of error messages */
PvcsRedirectOutput("error.log", PVCS_REDIR_STDERR);

/* Turn off screen output to silence PvcsQueryConfiguration */
PvcsRedirectOutput("nul", PVCS_REDIR_STDOUT);

/* Process configuration file */
status = PvcsQueryConfiguration(NULL,
   NULL,
   0,
   PVCS_CONFIG_OVERWRITE);

/* If no error, turn screen output back on */
if (!status)
   PvcsRedirectOutput("con", PVCS_REDIR_STDOUT);
/* Program would continue on from here */
```

Related Functions

PvcsQueryConfigurationError on page 143
PvcsGetErrorMessage on page 92
PvcsInit on page 113
PvcsSetGlobalParameter on page 168
PvcsDiagnosticEnable on page 68

PvcsRegisterCallback

This function registers a callback function that Developer's Toolkit calls when an event occurs during Version Manager processing. The parameter list passed to your callback function varies depending on the type of callback.

Syntax

For Microsoft C users:

```
PvcsRegisterCallback(
   int callbackType,/* Input */
   int (PVCS_CALLBACK *function)())/* Input */
```

For IBM C Set users:

```
PvcsRegisterCallback(
   int callbackType,/* Input */
   int (* PVCS_CALLBACK function)())/* Input */
```

For Visual C++ users:

```
PvcsRegisterCallback(
   callbackType,     /* Input */
   (int (__cdec1 *)(void)) function)
```

Parameters

NOTE The syntax for each specific callback type is explained in detail later in this section.

callbackType Integer that specifies the callback function identified in the header file. Values include:

- PVCS_CALLBACK_CFG_ALIASREF
  Calls the callback function when **PvcsQueryConfiguration** encounters an alias referenced in a configuration file.
- PVCS_CALLBACK_CFG_CONDITION
  Calls the callback function when **PvcsQueryConfiguration** encounters a conditional construct. This callback function is invoked no more than once per configuration file.
- PVCS_CALLBACK_CFG_INCLUDE
  Calls the callback function when **PvcsQueryConfiguration** encounters an INCLUDE (or @) directive.
- PVCS_CALLBACK_CHGDESC
  Calls the callback function that Developer's Toolkit calls to retrieve a change description.
- PVCS_CALLBACK_CONFIG
  Calls the callback function for each line processed in the configuration file.
- PVCS_CALLBACK_CONFIRM
  Calls the callback function when a Yes/No response is required for a particular event.
- PVCS_CALLBACK_DELAY
  Calls the callback function that indicates when a file is in use, and that it has entered a semaphore delay/retry loop.

- PVCS_CALLBACK_FREEMEM
  Calls the callback function that indicates when a buffer, supplied by your application, has finished processing.
- PVCS_CALLBACK_NO_DIRECTORY
  Calls the callback function if a nonexistent directory is encountered as a directive parameter while processing the configuration file. This callback gives your application the opportunity to create the directory. If you create the directory, your function should return zero.
- PVCS_CALLBACK_WORKDESC
  Calls the callback function that Developer's Toolkit calls (**PvcsCreateArchive** or **PvcsPutRevision**) to retrieve a workfile description.
- PVCS_CALLBACK_YIELD
  Calls the callback function at frequent intervals during processing to yield control to the operating system.

function — Pointer to the function that Developer's Toolkit calls when the callback event occurs. The function type is PVCS_CALLBACK, which is defined in PVCS.H.

Return Value — The return value is zero if the function is successful. Otherwise it returns PVCS_E_INVALID_PARAMETER. See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- Your callback function should return zero if you want Developer's Toolkit to continue execution. If the function returns a non-zero value, the Developer's Toolkit stops processing and returns PVCS_E_USER_ABORTED.
- When calling this function from a DLL, disable stack checking on callback functions. To disable stack checking using the Microsoft C compiler, use either the /Gs compiler option, or the check_stack pragma.

PVCS_CALLBACK_CFG_ALIASREF

This callback type registers a function that Developer's Toolkit calls when **PvcsQueryConfiguration** encounters an alias in the configuration file. This callback is invoked at most once per configuration file.

Syntax

```
int PVCS_CALLBACK PvcsCallbackCfgAliasRef(
   char *configFile)/* Input */
```

Parameter

configFile — Pointer to a string that contains the name of the configuration file that is being processed. This is the name of the file that contains the alias reference.

Return Value — This function should return PVCS_OK or PVCS_CANCEL. Developer's Toolkit terminates the operation if you return PVCS_CANCEL.

PVCS_CALLBACK_CFG_CONDITION

This callback type registers a function that Developer's Toolkit calls when **PvcsQueryConfiguration** encounters a conditional construct. This callback is invoked at most once per configuration file.

Syntax

```
int PVCS_CALLBACK PvcsCallbackCfgConditional(
   unsigned char *configFile)/* Input */
```

Parameter

configFile	Pointer to a string that contains the name of the configuration file that is being processed. This is the name of the file that contains the conditional construct.

Return Value This function should return *PVCS_OK* or PVCS_CANCEL. Developer's Toolkit terminates the operation if you return PVCS_CANCEL.

PVCS_CALLBACK_CFG_INCLUDE

This callback type registers a function that Developer's Toolkit calls when **PvcsQueryConfiguration** encounters an INCLUDE (or @) directive.

Syntax

```
int PVCS_CALLBACK PvcsCallbackCfgInclude(
   char *configFile,/* Input */
   char *includeFile,/* Input */
   int includeLevel,/* Input */
   void *reserved)  /* Reserved */
```

Parameters

configFile	Pointer to a string that contains the name of the configuration file that is being processed. This is the name of the file that contains the INCLUDE directive.
includeFile	Pointer to a string that contains the name of the file that is being included.
includeLevel	Integer that contains the number of include nesting levels.
reserved	Reserved for future use; must be null.

Return Value This function should return PVCS_OK or PVCS_CANCEL. Developer's Toolkit terminates the operation if you return PVCS_CANCEL.

PVCS_CALLBACK_CHGDESC

This callback type registers a function that Developer's Toolkit (**PvcsPutRevision**) calls to retrieve a change description from your application.

Syntax

```
int PVCS_CALLBACK PvcsCallbackChgDesc(
   char *workfileName,/* Input */
   char *archiveName,/* Input */
   char **ppDescription,/* Input */
   void *reserved)  /* Reserved */
```

Parameters

workfileName	Pointer to a string that contains the name of the workfile.
archiveName	Pointer to a string that contains the name of the archive.
ppDescription	Pointer to a pointer to a buffer that contains the workfile description. Your program supplies the buffer and indirectly updates the pointer.
reserved	Reserved for future use; must be null.

Return Value This function should return PVCS_OK or PVCS_CANCEL. Developer's Toolkit terminates the operation if you return PVCS_CANCEL.

PVCS_CALLBACK_CONFIG

This callback type registers a function that Developer's Toolkit (**PvcsQueryConfiguration**) calls to retrieve configuration information. In most cases, Developer's Toolkit calls your callback function once for each line that it reads. However, no callbacks exist for conditional constructs (for example, %if) or the following directives:

Abort
Echo
End
EndMaster
Include

Some directives apply to two configuration items. For example, the MultiLock directive is equivalent to MultiLock User and MultiLock Revision. The following directives generate two callbacks:

Multilock	PVCS_CFGITEM_MULTILOCKUSER PVCS_CFGITEM_MULTILOCKREV
Semaphore	PVCS_CFGITEM_SEMAPHORELOCAL PVCS_CFGITEM_SEMAPHORENETWORK
Compress	PVCS_CFGITEM_COMPRESSIMAGE PVCS_CFGITEM_COMPRESSDELTA
Share	PVCS_CFGITEM_SHARELOCAL PVCS_CFGITEM_SHARENET

Syntax

```
int PVCS_CALLBACK ConfigCallbackFunction
   int configItemType,/* PVCS_CFGITEM_... */
   char *configLine,/* Copy of the line */
   char *configFileName,/* Name of config file */
   unsigned configLineNumber)/* Line number in file */
```

Parameters

configItemType	Variable that specifies the configuration item type.
configLine	Pointer to a buffer that receives a copy of a line in the configuration file.

configFileName	Pointer to the name of the configuration file.
configLineNumber	Pointer to a buffer that receives the line number in the configuration file.

Return Value: This function should return PVCS_YES, PVCS_NO, or PVCS_CANCEL. Developer's Toolkit terminates the operation if you return PVCS_CANCEL.

Related Function: PvcsQueryConfigurationItem on page 144

PVCS_CALLBACK_CONFIRM

This callback type registers a function that Developer's Toolkit calls when it requires a Yes/No response to a particular event.

Syntax

```
int PVCS_CALLBACK PvcsCallbackConfirm(
   char *promptString,/* Input */
   int category,    /* Input */
   char void *reserved)/* Reserved */
```

Parameters

promptString	Pointer to a string that is the message prompt provided by Developer's Toolkit. For example, this string might be: `"A writable \"test.c\" exists."`
category	Integer that identifies the category of the prompt (list below).
reserved	Reserved for future use; must be null.

Return Value: This function should return PVCS_YES, PVCS_NO, or PVCS_CANCEL. Developer's Toolkit terminates the operation if you return PVCS_CANCEL.

Prompt Categories

The *category* parameter identifies the type of prompt. The following table lists the categories and the *promptString* parameters associated with each:

Category	Prompt String
PVCS_CONFIRM_DELETE_BRANCH	Delete branch "branchRevisionNumber" from archive?
PVCS_CONFIRM_DELETE_REVISION	Delete revision "revision1" from archive?
PVCS_CONFIGRM_DELETE_REVISIONS	Delete revisions "revision1" to "revision2" from archive?
PVCS_CONFIRM_MRG_OVERWRITE	Output file "workfile" exists, overwrite?
PVCS_CONFIRM_OVERWRITE_ARCHIVE	Archive "archive" exists. Overwrite?
PVCS_CONFIRM_LOCK_BRANCHWARN	Locking "archive" rev revision would cause a branch when checked in, get anyway?
PVCS_CONFIRM_OVERWRITE_WORK	A writable "workfile" exists, get anyway?
PVCS_CONFIRM_REUSE_COMMENT	Use previous comment for file "archive"?
PVCS_CONFIRM_PUT_UNCHANGED	Workfile "workfile" unchanged, put anyway?

Category	Prompt String
PVCS_CONFIRM_REPLACE_VERSION	Version "version" is already defined in archive "archive". Overwrite?
PVCS_CONFIRM_PUT_OLDER	Workfile "workfile" is older than its parent, put anyway?
PVCS_CONFIRM_MULTILOCK	Rev "revision1" in archive "archive" already locked. Apply additional lock?"
PVCS_CONFIRM_REPLACE_GROUP	Group "promotionGroup" is already defined in archive "*archive*". Overwrite?
PVCS_CONFIRM_BREAKLOCK	Unlock anyway?
PVCS_CONFIRM_PROMOTE_XBRANCH	Promoting "archive" from "group1" to "group2" will cross a branch boundary, has a merge been run?

PVCS_CALLBACK_DELAY

This callback type registers a function that Developer's Toolkit calls when a file is in use, and it has entered a semaphore delay/retry loop. This callback can display the number of remaining retries or cancel the operation.

NOTE Developer's Toolkit invokes this callback when a semaphore exists. The configuration file specifies the number of retry attempts (**SemaphoreRetry**), and the amount of time to delay between attempts (**SemaphoreDelay**).

Syntax

```
PvcsCallbackDelay(
   char *fileName,   /* Input */
   int attemptsRemaining,/* Input */
   void *reserved)   /* Reserved */
```

Parameters

fileName	Pointer to a string that contains the name of the file in use.
attemptsRemaining	Integer that contains the number of remaining retries.
reserved	Reserved for future use; must be null.

Return Values This function should return PVCS_OK or PVCS_CANCEL. Developer's Toolkit terminates the operation if you return PVCS_CANCEL.

PVCS_CALLBACK_FREEMEM

This callback type registers a function that Developer's Toolkit calls when it finishes processing a buffer that was supplied by your application. For example, the PVCS_CALLBACK_CHGDESC callback returns a pointer to a buffer that contains the change description. Developer's Toolkit calls the PVCS_CALLBACK_FREEMEM function to notify your program that it can free memory associated with the buffer.

Syntax

```
PvcsCallbackFreeMem(
   char *pAllocateMemory,/* Input */
   void *reserved)   /* Input */
```

Special Considerations

Developer's Toolkit calls PVCS_CALLBACK_FREEMEM when it finishes processing the description buffer (*ppDescription*). It doesn't call the PVCS_CALLBACK_FREEMEM function if you didn't register one.

Parameters

pAllocateMemory	Pointer to memory allocated by your program.
reserved	Must be null; reserved for future use.

Return Value

This function should return PVCS_OK or PVCS_CANCEL. Developer's Toolkit terminates the operation if you return PVCS_CANCEL.

PVCS_CALLBACK_WORKDESC

This callback type registers a function that Developer's Toolkit (**PvcsCreateArchive** or **PvcsPutRevision**) calls to retrieve a workfile description.

Syntax

```
PvcsCallbackWorkDesc(
   char *workfileName, /* Input */
   char *archiveName,/* Input */
   char **ppDescription,/* Input */
   void *reserved)  /* Reserved */
```

Parameters

workfileName	Pointer to a string that contains the name of the workfile.
archiveName	Pointer to a string that contains the name of the archive.
ppDescription	Pointer to a pointer to a buffer that contains the workfile description. Your program supplies the buffer and indirectly updates the pointer.
reserved	Reserved for future use; must be null.

Return Value

This function should return PVCS_OK or PVCS_CANCEL. Developer's Toolkit terminates the operation if you return PVCS_CANCEL.

PVCS_CALLBACK_NO_DIRECTORY

Developer's Toolkit calls this callback function it if detects a nonexistent directory as a directive parameter while processing the configuration file. The following directives can trigger this callback:

ArchiveWorkPVCS_CFGITEM_ARCHIVEWORKDIR
WorkDir PVCS_CFGITEM_WORKDIR
SemaphoreDirPVCS_CFGITEM_SEMDIR
VCSDir PVCS_CFGITEM_VCSDIR

Syntax

```
int PVCS_CALLBACK NoDirectoryCallbackFunction(
    int configItemType,/* PVCS_CFGITEM_... */
    char *path)/* Directory path */
```

Parameters

configItemType	Variable that identifies the configuration item.
path	Pointer to a string that identifies the path to the directory.

Return Value This function should return PVCS_OK or PVCS_CANCEL. Developer's Toolkit terminates the operation if you return PVCS_CANCEL.

PVCS_CALLBACK_YIELD

This callback type calls the callback function at frequent intervals during processing to yield central control to the operating system.

Syntax
```
int PVCS_CALLBACK YieldCallbackFunction(void)
```

Example
```
/* Register a "yield" function */
PvcsRegisterCallback(
   PVCS_CALLBACK_YIELD,/* Callback func identifier */
   callbackYield);/* Function to call */

/* Register a "config" function */
PvcsRegisterCallback(
   PVCS_CALLBACK_CONFIG,/* Callback func identifier */
   callbackConfig);/* Function to call */

/* Register a "no directory" function */
PvcsRegisterCallback(
   PVCS_CALLBACK_NO_DIRECTORY, /* Callback func identifier */
   callbackNoDirectory);       /* Function to call */

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);
.
.
.
}

#pragma check_stack(off)
           /* When calling functions from a DLL, you */
            * must disable stack checking. Microsoft C */
            * provides this pragma to disable stack checking. */

/*
 * Callback function that is called at frequent intervals.
 */
static int PVCS_CALLBACK callbackYield(void)
{
   static int timesCalled = 0;
   int status = 0;
   printf("Yield callback: %d\n", ++timesCalled);
   return status;
}

/*
 * Callback func called during configuration file processing.
```

```
 */
static int PVCS_CALLBACK callbackConfig(int itemType,
   char *cfgLine, char *fileName, unsigned lineNum)
{
   int status = 0;
   printf("Config callback: %2d: %s (%u) \"%s\"\n",
            itemType, fileName, lineNum, cfgLine);
   return status;
}

/*
 * Callback function that is called during configuration file
 * processing if a directive specifies a nonexistent
      directory.
 */
static int PVCS_CALLBACK callbackNoDirectory(int itemType, char *path)
{
   int answer;
   int status = 0;
   printf("NoDirectory callback: directory \"%s\" does
            not exist.\n",path);
   printf("\tCreate? (y/n) ");
   answer = getche();
   printf("\n");
   if (answer == 'n')
            status = 1;/* Abort processing */
   else
            mkdir(path);/* Create dir and continue */
   return status;
}

    #pragma check_stack()   /* Enable stack checking */
```

Related Functions
PvcsQueryConfigurationItem on page 144
PvcsRegisterBuildCallback on page 190

PvcsRegisterEvent

This function registers an event trigger. Once registered, Developer's Toolkit executes the event trigger whenever its associated event occurs during run time.

Syntax

```
PvcsRegisterEvent(
   int eventID,     /* Input */
   int eventTriggerType,/* Input */
   unsigned char *triggerInfo)/* Input */
```

Parameters

eventID

Integer that identifies the event that triggers this handler. Values include:

- PVCS_EVENT_ALL_EVENTS
  Registers a trigger that is executed whenever any event occurs.

- PVCS_EVENT_UNCOND_PRE_PUT
  Registers a trigger that is executed before a new revision is checked in and when the workfile has not yet been read by Version Manager.
- PVCS_EVENT_PRE_PUT
  Registers a trigger that is executed before a new revision is checked in.
- PVCS_EVENT_POST_PUT
  Registers a trigger that is executed after a revision is checked in.
- PVCS_EVENT_PRE_GET
  Registers a trigger that is executed before a workfile is checked out.
- PVCS_EVENT_POST_GET
  Registers a trigger that is executed after a workfile is checked out.
- PVCS_EVENT_POST_JOURNAL
  Registers a trigger that is executed after an entry is written to the journal file (Journal directive in effect).
- PVCS_EVENT_PRE_PROMOTE
  Registers a trigger that is executed before a revision is promoted.
- PVCS_EVENT_POST_PROMOTE
  Registers a trigger that is executed after a revision is promoted.
- PVCS_EVENT_PRE_VERSION_LABEL
  Registers a trigger that is executed before a version label is applied.
- PVCS_EVENT_POST_VERSION_LABEL
  Registers a trigger that is executed after a version label is applied.
- PVCS_EVENT_PRE_LOCK
  Registers a trigger that is executed before a revision is locked.
- PVCS_EVENT_POST_LOCK
  Registers a trigger that is executed after a revision is locked.
- PVCS_EVENT_PRE_UNLOCK
  Registers a trigger that is executed before a revision is unlocked.
- PVCS_EVENT_POST_UNLOCK
  Registers a trigger that is executed after a revision is unlocked.
- PVCS_EVENT_PRE_CREATE_ARCHIVE
  Registers a trigger that is executed before a new archive is created.

- PVCS_EVENT_POST_CREATE_ARCHIVE
  Registers a trigger that is executed after a new archive is created.

eventTriggerType — Identifies the type of event trigger. It can be an executable or batch file command line.

PVCS_EVENT_TYPE_UNKNOWN
PVCS_EVENT_TYPE_CMDLINE

triggerInfo — Pointer to a string that contains the name of the executable.

Return Value

The return value is zero if the function is successful. Otherwise you can receive PVCS_E_INVALID_PARAMETER. See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- You can register only one trigger for each *eventID*. If you try to register a trigger for an event that already has a trigger registered, it replaces the first trigger.
- Remember that if you register an event using the PVCS_EVENT_ALL_EVENTS flag, it registers a trigger that is executed whenever *any* event occurs.

Example

```
char *theCmdLine = "build -f myevent.bld";

/* Register an event. */
PvcsRegisterEvent(PVCS_EVENT_POST_PUT,
   PVCS_EVENT_TYPE_CMDLINE,
   (PVCS_PUCHAR)theCmdLine);


/* At this point you might do any other processing.
 * When a check-in action occurs, theCmdLine will be executed,
 * causing the build script MYEVENT.BLD to be executed. */


/* At program completion (or earlier) you should unregister
 * all events that you registered */

PvcsUnRegisterEvent(PVCS_EVENT_POST_PUT);
```

Related Functions

PvcsUnRegisterEvent on page 173
PvcsQueryConfigurationItem on page 144

PvcsReportDifferences

This function compares two files or revisions and reports the differences between them. It is equivalent to the VDIFF command.

Syntax

```
PvcsReportDifferences(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *refFile,/* Input */
   unsigned char *refRev,/* Input */
   unsigned char *tgtFile,/* Input */
   unsigned char *tgtRev,/* Input */
   unsigned char *outFile,/* Input */
```

```
unsigned short tabs,/* Input */
unsigned short contextLines, /* Input */
unsigned short recordLength, /* Input */
unsigned char *columnMask,/* Input */
PVCS_FLAGS flags)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
refFile	Pointer to a string containing the name of the reference file. Specify a null pointer if the reference file is an archive whose archive handle is specified in the *hArchive* parameter.
refRev	Pointer to a string containing the revision number or version label of the reference file. If the version label begins with a number, precede it with a backslash (\). If the reference file is not an archive, or if the revision is the tip revision, specify a null pointer.
tgtFile	Pointer to a string containing the name of the target file. Specify a null pointer if the target file is an archive whose archive handle is specified in the *hArchive* parameter.
tgtRev	Pointer to a string containing the revision number or version label of the target file. If the version label begins with a number, precede it with a backslash. If the target file is not a revision, or if it is the tip revision, specify a null pointer.
outFile	Pointer to a string containing the name of a file to which this function writes the difference report. The specified file will be overwritten if it exists. If this parameter is null, the report is sent to standard output. This parameter is equivalent to the VDIFF -XO command.
tabs	Integer containing the number of spaces to display between tabs in the difference report. This parameter is equivalent to the VDIFF -E command.
contextLines	Integer containing the number of context lines to display in the report. If you specify 0, the entire report displays. This parameter is equivalent to the VDIFF -L command.
recordLength	Integer specifying the record length of the files to be compared. Specify 0 to indicate that the files use newline characters to indicate the ends of lines. This parameter is equivalent to the VDIFF -XRecordLength command.
columnMask	String specifying the columns to convert to spaces when comparing files. Specify a null pointer to indicate no column masking. This parameter is equivalent to the VDIFF -XColumnMask command.
flags	Bit field that controls the operation of this function. Values include: ■ PVCS_DIFF_ALLDIFF Displays all differences. This flag is equivalent to the VDIFF -A command. ■ PVCS_DIFF_APPEND Appends the report to an existing file.

- PVCS_DIFF_BRIEF
  Eliminates line numbers from the report. This flag is equivalent to the VDIFF -N command.
- PVCS_DIFF_NOMOVE
  Does not display moved text. The report displays moved text as deletions and insertions.
- PVCS_DIFF_IGN_WHITE
  Ignores leading and trailing white space during comparison. This flag is equivalent to the VDIFF -B command.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_ACCESS_DENIED
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_BAD_FILENAME
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
PVCS_E_NO_REVISION
PVCS_E_USER_ABORTED
```

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
char *outFile = "foo.dif";
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Report differences between tip of archive and workfile */
status = PvcsReportDifferences(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",/* Use archive as reference file */
   NULL,    /* Use tip of archive */
   "foo.c", /* Use workfile as target file */
   NULL,    /* Not an archive */
   outFile, /* Difference output file */
   0,       /* Use default tab spacing */
   10,      /* Display 10 lines of context */
   0,       /* Newlines are end of line */
   NULL,    /* No columnmask */
   PVCS_DIFF_IGN_WHITE); /* Ignore white space */

if (!status)
   printf("Created difference report in \"%s\".\n",outFile);
```

Related Functions

PvcsFindFirstArchive on page 79
PvcsTestDifferences on page 170

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Comparing files	*VDIFF command*
Specifying archive and workfile names	*File Specification*

PvcsSetGlobalParameter

This function sets global parameters that affect all Developer's Toolkit functions. If you are using Developer's Toolkit functions to develop a Windows application, you must call **PvcsSetGlobalParameter()** with the PVCS_GLOBAL_NOPROMPT flag to suppress prompts from the toolkit. You should also call the PVCS_GLOBAL_NOMESSAGES flag to suppress the display of console messages.

Syntax

```
PvcsSetGlobalParameter(
   int globalParameter)/* Input */
```

Parameters

globalParameter Integer that identifies the global parameter to set. Values include:

- PVCS_GLOBAL_PROMPT
  Enables all interactive prompts. This is the default.
- PVCS_GLOBAL_NOPROMPT
  Suppresses all interactive prompts. Developer's Toolkit functions return an error in circumstances that require a prompt.
- PVCS_GLOBAL_MESSAGES
  Enables console messages. This is the default.
- PVCS_GLOBAL_NOMESSAGES
  Suppresses the display of console messages.

Special Consideration

- If you are using Developer's Toolkit functions to develop a Windows application, you must call **PvcsSetGlobalParameter()** with the PVCS_GLOBAL_NOPROMPT flag to suppress prompts from the toolkit. You should also call the PVCS_GLOBAL_NOMESSAGES flag to suppress the display of console messages.

  Using **PvcsSetGlobalParameter()** the PVCS_GLOBAL_NOPROMPT flag causes Developer's Toolkit to return an error when it encounters a situation in which a prompt is required. To handle the error, you should also call **PvcsRegisterCallback()** with the PVCS_CALLBACK_CONFIRM type to register a function that is called every time a yes/no response is required by Developer's Toolkit. Developer's Toolkit only terminates the application if the callback function returns PVCS_CANCEL.

Example

```
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Abort program if a prompt is required */
PvcsSetGlobalParameter(PVCS_GLOBAL_NOPROMPT);
/* Program would continue on from here */
```

Related Functions

PvcsCreateArchive on page 65
PvcsGetRevision on page 100
PvcsMerge on page 124
PvcsPutRevision on page 134
PvcsReportDifferences on page 165

PvcsSetProjectSemaphore

This function sets or clears a semaphore for an archive directory, or all directories named by the VCSDir directive. If set, this semaphore disallows update access to all archives in the directory. It requires the SetProjectSemaphore privilege.

Syntax

```
PvcsSetVcsdirSemaphore(
   unsigned char *archDir,/* Input */
   unsigned char *semName,/* Input */
   PVCS_FLAGS flags /* Input */
```

Parameters

archDir Pointer to a string that contains the name of an archive directory or archive name. The pointer can be null, which implies all directories named by VCSDir.

semName Name of the project semaphore. A null pointer causes Developer's Toolkit to use the default name.

flags Bit field that controls how the function operates. Values include:

- PVCS_SEMAPHORE_SET
  Sets the semaphore.
- PVCS_SEMAPHORE_CLEAR
  Clears the samaphore.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_CANT_CREATE_SEMAPHORE
PVCS_E_CANT_DELETE_SEMAPHORE
PVCS_E_INVALID_PARAMETER

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If the *archDir* parameter points to an archive directory, Developer's Toolkit creates the semaphore for that directory. If it points to an archive name, the semaphore is created for the directory containing that archive. The archive name can be unqualified; in this case the archive directory is located using the VCSDir path. If the *archDir* parameter is a null pointer, semaphores are created for all directories named by VCSDir.
- All project semaphores are file semaphores. Project semaphores are unaffected by Semaphore directives.
- If a semaphore exists in a directory, and you try to create another semaphore in the same directory, then Developer's Toolkit invokes the PVCS_CALLBACKFUNC_SEMAPHORE function. The directory name, and PVCS_SEMAPHORE_ALREADY_SET error are passed to the callback function.
- If a semaphore does not exist in a directory, and you try to clear it, Developer's Toolkit invokes the PVCS_CALLBACKFUNC_SEMAPHORE function. The directory name and PVCS_SEMAPHORE_ALREADY_CLEAR error are passed to the callback function.
- Each time an archive is opened for an update operation, the directory in which the archive resides is checked for a project semaphore. If one is present, then the Developer's Toolkit call attempting an update returns a PVCS_E_PROJECT_BUSY error. No retries are attempted.

- Command-line consideration: If a command-line command (GET, PUT, VCS) attempts to update an archive residing in a directory containing a project semaphore, Developer's Toolkit returns an error.
- Previous versions of Version Manager (5.1.1 or earlier) do not recognize project semaphores.

Example

```
PVCS_PUCHAR szArchive = NULL;
PVCS_PUCHAR szSemaphore = NULL;

/ * Initialize configuration settings */
rc = PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/*
 * Set a project semaphore. This code just puts one in the archive
 * directory that the archive we were given is in using the default
 * project semaphore name. Other options are:
 *
 * szArchive:
 * NULL - set semaphore on all directories in VCSDir,
 * A directory - set semaphore in that directory,
 * An archive - set semaphore in directory containing the archive */

rc = PvcsSetProjectSemaphore(
   szArchive,
   NULL,
   PVCS_SEMAPHORE_SET);

/*
 * Clear the project semaphore.
 */

rc = PvcsSetProjectSemaphore(
   szArchive,
   NULL,
   PVCS_SEMAPHORE_CLEAR);
```

PvcsTestDifferences

This function compares two files or revisions and returns a status code that indicates whether they are different. This is equivalent to the VDIFF -T command.

Syntax

```
PvcsTestDifferences(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *refFile,/* Input */
   unsigned char *refRev,/* Input */
   unsigned char *tgtFile,/* Input */
   unsigned char *tgtRev,/* Input */
   unsigned short *diffStatus,/* Output */
   unsigned short recordLength,/* Input */
   unsigned char *columnMask,/* Input */
   PVCS_FLAGS flags)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
refFile	Pointer to a string containing the name of the reference file. Specify a null parameter if the reference file is an archive whose archive handle is specified in the *hArchive* parameter.
refRev	Pointer to a string containing the revision number or version label of the reference file. If the version label begins with a number, precede it with a backslash (\). If the reference file is not an archive, or if it is the tip revision, specify a null pointer.
tgtFile	Pointer to a string containing the name of the target file. Specify a null pointer if the target file is an archive whose archive handle is specified in the *hArchive* parameter.
tgtRev	Pointer to a string containing the revision number or version label of the target file. If the version label begins with a number, precede it with a backslash. If the target file is not an archive, or if it is the tip revision, specify a null pointer.
diffStatus	Pointer to an integer that indicates whether the target file differs from the reference file. Zero indicates that the files are identical; a non-zero value indicates that the target file has changed.
recordLength	Integer specifying the record length of the files to be compared. Specify 0 to indicate that the files use newline characters to indicate the ends of lines. This parameter is equivalent to the VDIFF -XRecordLength command.
columnMask	String specifying the columns to ignore when comparing. This parameter is equivalent to the VDIFF -XColumnMask command.
flags	Bit field that controls the operation of this function. Value includes: ■ PVCS_DIFF_IGN_WHITE Ignores leading and trailing white space during comparison. This flag is equivalent to the VDIFF -B command.

Return Values

The return value is zero if the function is successful. Other values may be:

PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_NO_REVISION
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
PVCS_E_ACCESS_VIOLATION

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int diffStatus;
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Test differences between tip of archive, and workfile */
status = PvcsTestDifferences(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",/* Use archive as ref file */
   NULL,    /* Use tip of archive */
   "foo.c", /* Use workfile as target */
```

```
   NULL,    /* Not an archive */
   &diffStatus,/* Receives difference status */
   0,       /* Newlines are end of line */
   NULL,                /* No columnmask */
   PVCS_DIFF_IGN_WHITE); /* Ignore leading and trailing
            whitespace */

if (!status) {
   if (!diffStatus)
            printf("Files are identical.\n");
   else
            printf("Files are different.\n");
}
```

Related Functions
PvcsFindFirstArchive on page 79
PvcsReportDifferences on page 165

Related Topics For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Comparing files	*VDIFF command*
Specifying archive and workfile names	*File Specification*

PvcsUnLockRevision

This function unlocks a revision in an archive. It is equivalent to the VCS -U command.

Syntax

```
PvcsUnLockRevision(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *revarg,/* Input */
   unsigned char *locker)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open.
revarg	Pointer to a string containing the revision number, version label, or promotion group to unlock. If the version label begins with a number, precede it with a backslash (\). If this parameter is null, the function removes all locks owned by *locker.* This parameter is equivalent to the -R option used with the VCS -U command.
locker	Pointer to a string containing the user ID of the person owning the lock. If this parameter is null, the function uses the current user ID.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
PVCS_E_LOCKED_REVISION
PVCS_E_NOT_LOCKED
PVCS_E_USER_ABORTED
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Consideration

If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Unlock a revision */
PvcsUnLockRevision(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",/* Archive name */
   "1.12",  /* Revision to unlock */
   "DAVEE");/* User ID of lock owner */
```

Related Functions

PvcsGetLockInfo on page 95
PvcsLockRevisionGroup on page 117
PvcsOpenArchive on page 129

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Unlocking revisions	*VCS command*
Locking revisions	*Revision Locking*
Specifying archive and workfile names	*File Specification*

PvcsUnRegisterEvent

This function removes the handle for a specified event.

Syntax

```
PvcsUnRegisterEvent(
   int eventID)     /* Input */
```

Parameters

eventID Integer that identifies the event that triggers this handler. Values include:

- PVCS_EVENT_ALL_EVENTS
  Unregisters a trigger that would otherwise execute whenever any event occurs.

- PVCS_EVENT_UNCOND_PRE_PUT
  Unregisters a trigger that is executed before a new revision is checked in and when the workfile has not yet been read by Version Manager.
- PVCS_EVENT_PRE_PUT
  Unregisters a trigger that executes before a new revision is checked in.
- PVCS_EVENT_POST_PUT
  Unregisters a trigger that executes after a revision is checked in.
- PVCS_EVENT_PRE_GET
  Unregisters a trigger that executes before a workfile is checked out.
- PVCS_EVENT_POST_GET
  Unregisters a trigger that executes after a workfile is checked out.
- PVCS_EVENT_POST_JOURNAL
  Unregisters a trigger that executes after an entry is written to the journal file (Journal directive in effect).
- PVCS_EVENT_PRE_PROMOTE
  Unregisters a trigger that executes before a revision is promoted.
- PVCS_EVENT_POST_PROMOTE
  Unregisters a trigger that is executed after a revision is promoted.
- PVCS_EVENT_PRE_VERSION_LABEL
  Unregisters a trigger that is executed before a version label is applied.
- PVCS_EVENT_POST_VERSION_LABEL
  Unregisters a trigger that is executed after a version label is applied.
- PVCS_EVENT_PRE_LOCK
  Unregisters a trigger that is executed before a revision is locked.
- PVCS_EVENT_POST_LOCK
  Unregisters a trigger that is executed after a revision is locked.
- PVCS_EVENT_PRE_UNLOCK
  Unregisters a trigger that is executed before a revision is unlocked.
- PVCS_EVENT_POST_UNLOCK
  Unregisters a trigger that is executed after a revision is unlocked.
- PVCS_EVENT_PRE_CREATE_ARCHIVE
  Unregisters a trigger that is executed before a new archive is created.
- PVCS_EVENT_POST_CREATE_ARCHIVE
  Unregisters a trigger that is executed after a new archive is created.

Return Values

The return value is zero if the function is successful. Otherwise you can receive PVCS_E_INVALID_PARAMETER. See Chapter 4, "Return Values" for descriptions of return values.

Example

```
PVCS_PUCHAR theCmdLine = "build -f myevent.bld";

/* Register an event. */
PvcsRegisterEvent(PVCS_EVENT_POST_PUT,
```

```
  PVCS_EVENT_TYPE_CMDLINE,
  theCmdLine);

/* At this point you might do any other processing. When
     * a check in action occurs, theCmdLine will be executed,
 * causing the build script MYEVENT.BLD to be executed. */

/* At program completion (or earlier) you should unregister
 * all events that you registered */
PvcsUnRegisterEvent(PVCS_EVENT_POST_PUT);
```

Related Function
PvcsRegisterEvent on page 163
PvcsQueryConfigurationItem on page 144

PvcsVconfig

This function embeds the master configuration file, access control database name, and the list of user ID sources in the Version Manager DLL files. This function is equivalent to the VCONFIG command.

Syntax

```
PvcsVconfig(
   unsigned char *DLLName,/* Input */
   unsigned char *accessDbName,/* Input */
   unsigned char *masterConfigName, /* Input */
   unsigned char *userIDSources, /* Input */
   unsigned char *language,/* Input */
   unsigned char *messagePath,/* Input */
   unsigned char *localePath,/* Input */
   void *reserved)  /* Reserved */
```

Parameters

DLLName	Pointer to a string that specifies the Developer's Toolkit DLL to configure (Windows or Protected Mode DLL). The file name cannot contain wildcards. If this is a null pointer, then Developer's Toolkit uses the default file name. If it is a null string, then it returns an error.
accessDbName	Pointer to a string that specifies the access control database. If it is a null pointer, then its setting is not affected. If it is a null string, then the currently embedded access control database is removed. This parameter is equivalent to the VCONFIG -A command.
masterConfigName	Pointer to a string that specifies the master configuration file. If it is a null pointer, then its setting is not affected. If it is a null string, then the currently embedded master configuration name is removed. This parameter is equivalent to the VCONFIG -C command.

userIDSources	Pointer to a string that specifies the user identification sources and the order in which Version Manager uses them. Sources are not case-sensitive, and include the following values: HOST LANMAN NETWARE UNKNOWN VCSID VLOGIN If *userIdSources* is a null pointer, then its setting is not affected. If it is a null string, then the currently embedded user identification sources are removed. This parameter is equivalent to the VCONFIG -C command.
language	Pointer to a string that specifies the default language. If *language* is a null pointer, then its setting is not affected. If *language* is a null string, then the currently embedded default language is removed. This parameter is equivalent to the VCONFIG -I command.
messagePath	Pointer to a string that names the path to the message file. If *messagePath* is a null pointer, then its setting is not affected. If it is a null string, then the currently embedded message file path is removed. This parameter is equivalent to the VCONFIG -P command.
localePath	Pointer to a string that specifies the path to the locale file. If it is a null pointer, then its setting is not affected. If it is a null string, then the currently embedded locale file path is removed. This parameter is equivalent to the VCONFIG -C command.
reserved	Reserved for future use.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_INVALID_PARAMETER
PVCS_E_BAD_USER_ID_SOURCE
PVCS_E_CANT_OPEN_VCONFIG_FILE
PVCS_E_CANT_READ_VCONFIG_FILE
PVCS_E_CANT_WRITE_VCONFIG_FILE
PVCS_E_BAD_SERIAL_NUMBER
PVCS_E_BAD_VCONFIG_FILE
PVCS_E_UNKNOWN_ID_SOURCE
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

For UNIX users:

- Under UNIX, the *DLLName* parameter is *library*.
- Under UNIX, LANMAN and NETWARE are not applicable values for the *userIdSources* parameter. If you use VLOGIN as a login source, you must call **PvcsLogin** to validate a login ID.

Example

```
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Implant configuration information into DLL */
status = PvcsVconfig(
   NULL,          /* Use default DLL file name */
```

```
   "c:\\pvcs\\access.db",/* Name of access control db */
   "c:\\pvcs\\master.cfg",/* Name of master cfg file */
   "netware, vlogin, host",/* List of user ID sources */
   NULL,          /* Default language unchanged */
   NULL,    /* Path unchanged */
   NULL,    /* Path unchanged */
   (void *)0);/* Reserved for future use */

if (!status)
   printf("Implanted configuration information.\n");
}
```

PvcsVerifyPromoTree

This function verifies the promotion model by making sure it has exactly one group at the highest level (or *production group*) and that every group (except the production group) promotes to exactly one parent.

Syntax

```
PvcsVerifyPromoTree(
   void)
```

Return Value

The return value is zero if the function is successful. The other possible value is PVCS_E_BAD_PROMO_TREE. See Chapter 4, "Return Values" for descriptions of return values.

Example

```
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Build the promotion model */
PvcsAddPromoteTreeNode("QA", "PRODUCTION");
PvcsAddPromoteTreeNode("DEV1", "QA");
PvcsAddPromoteTreeNode("DEV2", "QA");
PvcsAddPromoteTreeNode("DEV3", "QA");

/* Verify the model */
status = PvcsVerifyPromoTree();
if (!status)
   printf("Promotion model defined successfully.\n");
```

Related Functions

PvcsGetPromoParent on page 99
PvcsGroupToRevision on page 111
PvcsPromoteRevision on page 131
PvcsVerifyPromoTreeNodeExist on page 178

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Promotion models	*Promotion*

Defining a promotion group	*Promote directive*
Promoting revisions	*VPROMOTE command*

PvcsVerifyPromoTreeNodeExist

This function determines whether a promotion group exists in a promotion model and indicates whether it is a production group, development group, or neither.

Syntax

```
PvcsVerifyPromoTreeNodeExist(
   unsigned char *node_name,/* Input */
   unsigned short *value)/* Output */
```

Parameters

node_name Pointer to the name of the promotion group to verify.

value Pointer to a variable that receives one of the following values:

- PVCS_PROMO_DEV
  Bottom-level group, called a *development group.*
- PVCS_PROMO_MID
  Group exists and is neither a *development group* nor the *production group*.
- PVCS_PROMO_NO_EXIST
  Group does not exist in the model.
- PVCS_PROMO_ROOT
  Production group.

Return Value The return value is zero if the function is successful. The other possible value is PVCS_E_INVALID_PARAMETER. See Chapter 4, "Return Values" for descriptions of return values.

Special Consideration Use **PvcsGroupToRevision** to determine whether an archive contains a revision at the specified promotion group.

Example

```
int type;
int i;
char *production = "PRODUCTION";
char *qa = "QA";
char *development[3] = {"DEV1", "DEV2", "DEV3"};
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Build the promotion model */
for (i = 0; i < 3; i++)
   PvcsAddPromoteTreeNode(development[i], qa);
   PvcsAddPromoteTreeNode(qa, production);

/* Verify the existence of groups */
for (i = 0 ; i < 3; i++) {
```

```
   PvcsVerifyPromoTreeNodeExist(development[i], &type);
   displayGroupType(development[i], type);
}
PvcsVerifyPromoTreeNodeExist(qa, &type);
displayGroupType(qa, type);
PvcsVerifyPromoTreeNodeExist(production, &type);
displayGroupType(production, type);
.
.
.
void displayGroupType(char *group, int type)
{
   switch (type) {
            case PVCS_PROMO_DEV:
            printf("\"%s\" is a development group.\n",
            group); break;
   case PVCS_PROMO_MID:
            printf("\"%s\" is an intermediate group.\n",
            group); break;
   case PVCS_PROMO_ROOT:
            printf("\"%s\" is the production group.\n",
            group); break;
   case PVCS_PROMO_NO_EXIST:
            printf("\"%s\" does not exist in the
            hierarchy.\n", group); break;
   }
}
```

Related Functions

PvcsGetPromoParent on page 99
PvcsGroupToRevision on page 111
PvcsPromoteRevision on page 131
PvcsVerifyPromoTree on page 177

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide.*

For information about...	See...
Promotion models	*Promotion*
Defining a promotion group	*Promote directive*
Promoting revisions	*VPROMOTE command*

PvcsVersionToRevision

This function returns the revision numbers that are assigned to a specified version label.

Syntax

```
PvcsVersionToRevision(
   ARCHIVEHANDLE hArchive,/* Input */
   unsigned char *fileName,/* Input */
   unsigned char *versionLabel,/* Input */
   unsigned short *pVersCount,/* Input / Output */
   unsigned char *versionTable,/* Output */
   unsigned short bufLen)/* Input */
```

Parameters

hArchive	Handle returned by **PvcsOpenArchive**. If the archive is not open, specify ARCHIVEHANDLE_NOT_OPEN.
fileName	Pointer to the name of an archive or a workfile. This parameter is required only if the archive is not open.
versionLabel	Pointer to a string containing a version label. The version label must exist in the archive. If *versionLabel* is null, the function returns all revision/version pairs up to the limit specified in *pVersCount* or until the *versionTable* buffer is full.
pVersCount	Points to a variable that specifies the number of revision/version pairs to retrieve. The parameter returns the actual number retrieved.
versionTable	Pointer to a buffer that receives the revision/version pairs.
bufLen	Specifies the length of the *versionTable* buffer.

Return Values

The return value is zero if the function is successful. Other values may be:

```
PVCS_E_ACCESS_VIOLATION
PVCS_E_ARCHIVE_NOT_FOUND
PVCS_E_BUFFER_OVERFLOW
PVCS_E_FILE_BUSY
PVCS_E_INVALID_PARAMETER
PVCS_E_NO_REVISION
PVCS_E_NO_VERSION
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Considerations

- If *fileName* refers to a workfile, the program infers the archive name. If *fileName* contains wildcards, the program expands it according to the usual Version Manager rules.
- The format of the buffer is:

revisionString
null
versionString
null
. . .
null

Each revision string and version string is null-terminated, and the end of the buffer is marked by an additional null character. The revision number is returned as a printable string—for example, 1.15, 1.5.2.3, or 1.2.1.*. (The last example is a revision string for a floating version label.)

Example

```
unsigned short versCount = -1;
char versBuffer[512];
char *bufPtr;
```

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Obtain revisions associated with version labels */
PvcsVersionToRevision(ARCHIVEHANDLE_NOT_OPEN,
   "foo.c_v",/* Name of archive or workfile */
   NULL,    /* All revision/version pairs */
   &versCount,/* Returns number of pairs
            retrieved */
   versBuffer,/* Buffer receiving rev/ver
            pairs */
   sizeof(versBuffer));/* Length of the buffer */

/* Display all revisions with version labels */
bufPtr = versBuffer;
while (versCount--) {
   printf("Revision: %s", bufPtr);
   bufPtr += strlen(bufPtr) + 1;/* Skip over revision */
   printf("  Version: %s\n", bufPtr);
   bufPtr += strlen(bufPtr) + 1;/* Skip over version */
}
```

Related Functions

PvcsAssignVersion on page 54
PvcsGetRevisionInfo on page 103
PvcsOpenArchive on page 129

Related Topics

For more information, see the following topics in the *Serena PVCS Version Manager Command-Line Reference Guide*.

For information about...	See...
Version labels	*Version Labels*
Revision numbers	*Revision Numbering*

Chapter 3

Serena Configuration Builder Functions

PvcsBuild

This function builds a target according to the rules in a build script that was previously read with **PvcsReadBuildScript**.

Syntax

```
PvcsBuild(
   SCRIPTHANDLE theScript,/* Input */
   unsigned char *targetName,/* Input */
   unsigned short FAR *lastErrorlevel)/* Output */
```

Parameters

theScript	Handle that was initialized with **PvcsReadBuildScript**.
targetName	Pointer to a buffer containing the name of the target to be built. The function builds all sources for *targetName* first, if necessary, and then builds *targetName*. The function builds the default target if the value is null.
lastErrorlevel	Operating system exit code of the last command executed.

Return Values

The return value is one of the following if the function is successful:

```
PVCS_E_MKUPTODATE
PVCS_E_MKWORKED
```

If the operation is not successful, the return value is one of the following:

```
PVCS_E_MKFAILED
PVCS_E_MKABORTED
```

See Chapter 4, "Return Values" for descriptions of return values.

Special Consideration

You can call **PvcsBuild** multiple times using a script handle that was initialized just once.

Example

```
char *scriptName = "makefile";
char *target = "foo.exe";
SCRIPTHANDLE handle;
PVCS_FLAGS options = PVCS_RD_DEBUG_OUTPUT |
   PVCS_RD_INHERIT_ENVIRONMENT |
   PVCS_RD_DEFINE_ENV_MACROS;
int errorLevel;
int status;

/* Read build script */
status = PvcsReadBuildScript(
   scriptName,/* Name of build script */
   options, /* Options */
   "MEMORY_MODEL=LARGE",/* Define a macro */
   &handle);/* Receives the script handle */
if (!status) {

  /* Build a target */
  status = PvcsBuild(
   handle,  /* Script handle */
   target,  /* Target to build */
   &errorLevel);/* Receives exit code of last command */
```

```
  /* Display message and close handle */
  if (!status) {
   printf("Successfully built target \"%s\".\n", target);
   PvcsCloseBuildScript(handle);
  }
}
```

Related Functions
PvcsReadBuildScript on page 186
PvcsCloseBuildScript on page 185

Related Topics
For more information, see the following topics in the *Serena Configuration Builder Reference Guide*.

For information about...	See...
Build script contents	*Build Scripts*
Building targets	*Targets*

PvcsCloseBuildScript

This function closes a script handle that was opened by **PvcsReadBuildScript**.

Syntax

```
PvcsCloseBuildScript(
   SCRIPTHANDLE theScript)   /* Input */
```

Parameter

theScript	Handle that was initialized with **PvcsReadBuildScript**.

Return Value
The return value is zero if the function is successful.

Special Considerations
- Use this function to free memory allocated in other Configuration Builder functions.
- Do not call this function if any previous Serena Configuration Builder function terminated unsuccessfully.

Example

```
char *scriptName = "makefile";
SCRIPTHANDLE handle;
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Read build script */
status = PvcsReadBuildScript(scriptName,
   CS_RD_INHERIT_ENVIRONMENT, "", &handle);
if (!status) {

   /* Close build script */
   status = PvcsCloseBuildScript(handle);
   if (!status)
```

```
                printf("Successfully closed build script \"%s\".\n",
        scriptName);
}
```

Related Functions
PvcsBuild on page 184
PvcsReadBuildScript on page 186

Related Topics For more information, see the following topics in the *Serena Configuration Builder Reference Guide*.

For information about...	See...
Build script contents	*Build Scripts*

PvcsReadBuildScript

This function reads a build script and returns a script handle, which is passed to **PvcsBuild**. A script handle identifies a body of rules that describe how to build a project (or parts of a project). You cannot access or modify this body of rules directly; instead, you must use this function.

Syntax

```
PvcsReadBuildScript(
   unsigned char *scriptName,/* Input */
   PVCS_FLAGS buildOptions,/* Input */
   unsigned char *macroDefs,/* Input */
     SCRIPTHANDLE FAR *theScript)/* Output */
```

Parameters

scriptName Name of the build script. If this parameter is null, the function finds the build script using the usual Configuration Builder rules.

buildOptions Bit field that controls the operation of this function. Values include:

- PVCS_RD_INHERIT_ENVIRONMENT
  Passes environment to commands executed by **PvcsBuild**. This flag emulates the program's default behavior, which you can override from the command line using the -NoEnvInherit flag.
- PVCS_RD_ALLOW_DEFAULTRULES
  Uses built-in rules. This flag emulates the program's default behavior, which you can override from the command line using the -Reject flag.
- PVCS_RD_ALLOW_INI_FILE
  Looks for the TOOLS.INI file. This flag emulates the program's default behavior, which you can override from the command line using the -Init flag.

- PVCS_RD_DEBUG_OUTPUT
  Displays debugging information. This flag is equivalent to the -Debug flag.

- PVCS_RD_DEFINE_ENV_MACROS
  Uses environment variables as predefined macros. This flag emulates the program's default behavior, which you can override from the command line using the -NoEnvMacros flag.

- PVCS_RD_DISPLAY_SYMBOL_TABLE
  Displays a summary of macro definitions, archive declarations, rules, and dependencies. This flag is equivalent to the -Summary flag.

- PVCS_RD_ENV_MACRO_PRECEDENCE
  Gives environment variables precedence over macro definitions. This flag emulates the program's default behavior, which you can override from the command line using the -NoRedefine flag.

- PVCS_RD_IGNORE_ERRORS
  Ignores any non-zero exit codes from operations. This flag is equivalent to the -Ignore flag.

- PVCS_RD_KEEP_WORKING
  Keeps working despite errors. This flag is equivalent to the -KeepWorking flag.

- PVCS_RD_MS_NMAKE_MODE
  Emulates Microsoft NMAKE. This flag is equivalent to the -Nmake flag.

- PVCS_RD_NOEXECUTION
  Displays operation lines but does not execute them. This flag is equivalent to the -NoExecute flag.

- PVCS_RD_SILENT_OPERATION
  Does not display commands and error messages. This flag is equivalent to the -Quiet flag.

- PVCS_RD_TOUCH_ONLY
  Sets the time and date of any files that need to be built to the current time and date. This flag is equivalent to the -Touch flag.

- PVCS_RD_UNCONDITIONAL_BUILD
  Forces a rebuild without regard to the timestamp of the target and all sources. This flag is equivalent to the -All flag.

macroDefs	A null-terminated string in the following form: `MacroName=Value...` Separate multiple `MacroName=Value` pairs with spaces.
theScript	Script handle for use in subsequent operations.

Return Values

The return value is zero if the function is successful. Other values may be:
PVCS_E_MKABORTED
PVCS_E_FILE_NOT_FOUND

See Chapter 4, "Return Values" for descriptions of return values.

Example

```
/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Read build script */
status = PvcsReadBuildScript(
  scriptName,/* Name of build script */
  options,  /* Options */
  "MEMORY_MODEL=LARGE",/* Define a macro */
  &handle); /* Receives the script handle */

if (!status) {

  /* Build a target */
  status = PvcsBuild(
    handle, /* Script handle */
    target, /* Target to build */
    &errorLevel);/* Receives exit code */
```

Related Function

PvcsCloseBuildScript on page 185

Related Topics

For more information, see the following topics in the *Serena Configuration Builder Reference Guide*.

For information about...	See...
Command-line flags	*Flags*
Build script contents	*Build Scripts*

PvcsRedirectBuildOutput

This function redirects standard output or standard error to a file.

Syntax

```
PvcsRedirectBuildOutput(
   SCRIPTHANDLE theScript,/* Input */
   unsigned char *newStdOut,/* Input */
```

```
  unsigned char *newStdErr)/* Input */
```

Parameters

theScript	Handle initialized with **PvcsReadBuildScript**. You can call this function with different build scripts, and specify different files for redirection. This parameter can be null, so you can call this function before calling **PvcsReadBuildScript** to redirect read-time output. If you set *theScript* to null, you must call **PvcsReadBuildScript** immediately after calling **PvcsRedirectBuildOutput**. The redirection is effective only for the SCRIPTHANDLE initialized by that call to **PvcsReadBuildScript**.
newStdOut	Pointer to a string that specifies a file name for standard output. If the string is a hyphen (-), the function redirects standard output to the file specified by *newStdErr*. If the string is null, the function does not redirect standard output. This enables you to redirect standard error without redirecting standard output.
newStdErr	Pointer to a string that specifies a file name for standard error. If the string is a hyphen (-), the function redirects standard error to the file specified by *newStdOut*. If the string is null, the function does not redirect standard error. This enables you to redirect standard output without redirecting standard error.

Return Value This function always returns zero.

Example

```
char *scriptName = "makefile";
char *target = "foo.exe";
SCRIPTHANDLE handle;
PVCS_FLAGS options = PVCS_RD_DEBUG_OUTPUT |
  PVCS_RD_INHERIT_ENVIRONMENT |
  PVCS_RD_DEFINE_ENV_MACROS;
char *outFile = "build.log";
int errorLevel;
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Read build script */
status = PvcsReadBuildScript(scriptName, options,
  "MEMORY_MODEL=LARGE", &handle);
if (!status) {

  /* Place all output in a log file */
  PvcsRedirectBuildOutput(
    handle, /* Script handle */
    outFile,/* File to contain std output messages */
    outFile);/* File to contain std error messages */

  /* Build a target */
  status = PvcsBuild(handle, target, &errorLevel);
```

```
  /* Close handle */
  if (!status)
    PvcsCloseBuildScript(handle);
  printf("Build output log is in \"%s\".\n", outFile);
}
```

Related Functions
PvcsReadBuildScript on page 186
PvcsBuild on page 184

Related Topics For more information, see the following topics in the *Serena Configuration Builder Reference Guide*.

For information about...	See...
Redirecting error messages	-DirectOutput flag -DirectAll flag -DirectErrors flag

PvcsRegisterBuildCallback

This function registers a callback function that Developer's Toolkit calls when an event occurs. The parameter list passed to your callback function varies depending on the type of the callback.

Syntax

```
PvcsRegisterBuildCallback(
   SCRIPTHANDLE theScript,/* Input */
   int callbackType, /* Input */
   PVCSCB_YIELDPROC pFunction)/* Input */
```

Parameters

theScript	Handle initialized with **PvcsReadBuildScript**. You can call this function with different build scripts, and specify different callbacks. This parameter can be null, so you can call this function before calling **PvcsReadBuildScript** to register callback functions. If you set *theScript* to null, call **PvcsReadBuildScript** immediately after calling **PvcsRegisterBuildCallback**. The callback function is effective only for the SCRIPTHANDLE initialized by that call to **PvcsReadBuildScript**.
callbackType	Integer that specifies the callback function identified in the header file. Values include: ■ PVCS_CALLBACK_YIELD Developer's Toolkit calls the specified function at frequent intervals during processing. Your function should have the following form: `int PVCS_CALLBACK YieldCallbackFunction(void)` ■ PVCSCB_CALLBACK_VIEW Developer's Toolkit calls the specified function at any point where the command line executable would write to standard output or standard error, and passes the message string to the callback function.
pFunction	Pointer to the function that Developer's Toolkit calls when the callback event occurs. If *callbackType* is PVCS_CALLBACK_YIELD, *pFunction* must point to a function of type PVCSCB_CALLBACK. If *callbackType* is PVCSCB_CALLBACK_VIEW, *pFunction* must point to a function of type PVCSCB_VIEWPROC, which requires a typecast. For example, the standard C `vprintf()` function is of this type.

Return Value

This function always returns zero.

Special Considerations

- By default, the Windows version of Serena Configuration Builder has a yield function that is a PeekMessage() loop. Some applications may need a more complex yield function. You can call **PvcsRegisterBuildCallback** with *callbackType* set to PVCS_CALLBACK_YIELD to register a yield function that Configuration Builder calls in place of the default yield function.

  If you register a yield callback that blocks the flow of messages, then the Windows desktop environment will be stopped until the entire build finishes.

  If the yield function returns a non-zero value, Configuration Builder terminates with the following message:

  `Received Interrupt: cleaning up.`

  This enables the application to cancel the build process—for example, with a Cancel button.

- If you call **PvcsRegisterBuildCallback** with *callbackType* set to PVCSCB_CALLBACK_VIEW, then Configuration Builder sends every message to the

display function before it displays the message. After calling the display function, Configuration Builder writes the normal output to the output window or redirected file.

- When calling this function from a DLL, disable stack checking on callback functions. To disable stack checking using the Microsoft C compiler, use either the /Gs compiler option, or the check_stack pragma.

Example

```
char *scriptName = "makefile";
char *target = "foo.exe";
SCRIPTHANDLE handle;
PVCS_FLAGS options = PVCS_RD_DEBUG_OUTPUT |
  PVCS_RD_INHERIT_ENVIRONMENT |
  PVCS_RD_DEFINE_ENV_MACROS;
int errorLevel;
int status;

/* Initialize configuration settings */
PvcsQueryConfiguration(NULL, NULL, 0, PVCS_CONFIG_OVERWRITE);

/* Read build script */
status = PvcsReadBuildScript(
   scriptName, options, "MEMORY_MODEL=LARGE", &handle);
if (!status) {

   /* Register a "yield" function */
   PvcsRegisterBuildCallback(
            handle,/* Build script handle */
            PVCS_CALLBACK_YIELD,/* Callback func id */
            callbackYield);/* Function to call */

   /* Build a target */
            PvcsBuild(handle, target, &errorLevel);

   /* Close handle */
            PvcsCloseBuildScript(handle);
}
.
.
.

#pragma check_stack(off)
   /*  When calling functions from a DLL, */
   /*  you must disable stack checking. Microsoft */
   /*  C provides this pragma to disable stack */
   /*  checking. */
/*
 * Callback function that is called at frequent intervals.
 */
static int PVCS_CALLBACK callbackYield(void)
{
   int status = 0;
   static int timesCalled = 0;
   printf("Yield callback: %d\n", ++timesCalled);
   return status;
}
```

```
#pragma check_stack()/* Enable stack checking */
```

Related Functions
PvcsReadBuildScript on page 186
PvcsBuild on page 184
PvcsRegisterCallback on page 155

PvcsSaveWinmainParams

This function saves the parameters that Windows passes to an application when it starts the application.

Syntax

```
PvcsSaveWinmainParams(
   unsigned hInst,  /* Input */
   unsigned hPrevInstance,/* Input */
   unsigned char *lpCmdLine,/* Input */
   int nCmdShow)    /* Input */
```

Parameters

hInst	The instance handle that Windows uses to identify the application.
hPrevInstance	The previous instance handle. This is the instance handle of the most recent active instance of the same program.
lpCmdLine	Command-line parameters.
nCmdShow	A number that specifies how to display the window.

Return Values

This function returns zero if it is successful. If the **PvcsSaveWinmainParams** parameters could not be stored, the Developer's Toolkit will return a non-zero value. If you receive this error, check to see that you have called **PvcsUnsaveWinmainParams** at the end of each Configuration Builder application.

Special Considerations

- Windows applications must call this function before calling other Configuration Builder functions, passing it the arguments that Windows passed to the application's WinMain function. Since you can run multiple instances of the DLL, this function records the instance handle, the previous instance handle, and command-line parameters for each program instance.
- This function is only available (and necessary) in the Windows version of the Developer's Toolkit.

Example

```
WinMain(
   HANDLE hInstance,
   HANDLE hPrevInstance,
   LPSTR lpCmdLine,
   int nCmdShow
   )
{
PvcsSaveWinmainParams(hInstance, hPrevInstance, lpCmdLine,
   nCmdShow);

/*... do other Configuration Builder calls here ... */

PvcsUnsaveWinmainParams();
```

```
return 0;
}
```

Related Function PvcsUnsaveWinmainParams on page 194

PvcsUnsaveWinmainParams

This function frees resources reserved by a previous call to **PvcsSaveWinmainParams**.

Syntax

```
PvcsUnsaveWinmainParams(
   void)
```

Parameters There are no parameters to this function.

Return Value This function always returns zero.

Special Considerations

- You must call this function as a cleanup function when your Windows program exits.
- This function is only available (and necessary) in the Windows version of the Developer's Toolkit.

Example

```
WinMain(
   HANDLE hInstance,
   HANDLE hPrevInstance,
   LPSTR lpCmdLine,
   int nCmdShow
)
{
PvcsSaveWinmainParams(hInstance, hPrevInstance, lpCmdLine,
   nCmdShow);

/*... do other Configuration Builder calls here ... */

PvcsUnsaveWinmainParams();

return 0;
}
```

Related Function PvcsSaveWinmainParams on page 193

Chapter 4

Return Values

Return Values

Developer's Toolkit error codes

The following table lists the values returned by Serena PVCS Version Manager Developer's Toolkit functions and the reasons why each occurs. The values for these codes are defined in PVCS.H. If you don't see a value listed here, check PVCS.H.

Code	Return Value	Description
1	PVCS_E_INTERNAL	An internal error occurred because of an unexpected condition.
2	PVCS_E_ARCHIVE_NOT_FOUND	Developer's Toolkit cannot find the specified archive.
3	PVCS_E_ALREADY_EXISTS	A file cannot be created because it already exists, and the `NO_OVERWRITE` flag was specified.
4	PVCS_E_INVALID_CONFIG_PARM	A line in the configuration file is invalid.
5	PVCS_E_ACCESS_DENIED	File access is denied at the file system level. The most likely cause is insufficient network privileges.
6	PVCS_E_FILE_BUSY	File access is denied because the file is in use by another process.
7	PVCS_E_INVALID_PARAMETER	A parameter to a function is invalid.
8	PVCS_E_BUFFER_OVERFLOW	The buffer receiving information is not large enough.
9	PVCS_E_BAD_FILENAME	The file name is invalid because the drive or path does not exist, or the file name contains invalid characters.
10	PVCS_E_BAD_ARCHIVE_HANDLE	The archive handle does not specify an open archive.
11	PVCS_E_NO_VERSION	The specified version label does not exist.
12	PVCS_E_VERSION_EXISTS	Developer's Toolkit could not add a version label because the label already exists.

Code	Return Value	Description
13	PVCS_E_ACCESS_VIOLATION	Access to the archive is denied by Version Manager because the current user has insufficient access privileges.
14	PVCS_E_LOCKED_REVISION	You tried to operate on a revision that is currently locked.
15	PVCS_E_USER_ABORTED	The user canceled the request in response to a prompt.
16	PVCS_E_NO_MEMORY	A memory allocation request failed.
17	PVCS_E_BAD_DATE	The date specification is invalid.
18	PVCS_E_NO_REVISION	The specified revision does not exist.
19	PVCS_E_BAD_REVISION	The revision specification contains incorrect syntax.
20	PVCS_E_ARCHIVE_EMPTY	Developer's Toolkit cannot process the request because the archive contains no revisions.
21	PVCS_E_EXCESS_CFG_NESTING	The levels of configuration file nesting exceed the maximum of 20.
22	PVCS_E_SAME_NAME	The workfile and archive names are identical.
23	PVCS_E_CANT_DELETE	An error occurred when the program tried to delete the file.
24	PVCS_E_OVERWRITE	A file already exists, and permission to overwrite it was not given.
25	PVCS_E_NO_HANDLES	The operating system is out of file handles.
26	PVCS_E_NET_NOT_ALLOWED	Developer's Toolkit detected a license violation and is denying access to network files.
27	PVCS_E_BAD_ARCHIVE	The file is not an archive.
28	PVCS_E_NO_ATTRIBUTE	No attribute exists.
29	PVCS_E_CANT_LOCK	Developer's Toolkit cannot lock a revision. Either lock checking is disabled for the archive, or the archive has the ExclusiveLock attribute and there is already a lock on it.
30	PVCS_E_NOT_LOCKED	Developer's Toolkit expected to find a locked revision, but no locks exist on the specified revisions.
31	PVCS_E_NO_MORE_FILES	There are no more files that match the wildcard file specification.

Code	Return Value	Description
32	PVCS_E_WORK_NOT_FOUND	The program cannot find the specified workfile.
33	PVCS_E_PROMPT	Version Manager requires user input, but prompts are suppressed.
34	PVCS_E_INVALID_PROMO	Either you tried to promote the production group, there are no revisions at the group from which you tried to promote, or you attempted some other promotion that violates the promotion model.
35	PVCS_E_BAD_PROMO_TREE	**PvcsQueryConfiguration** returns this error code when a configuration file contains Promote directives with recursive promotions or multiple top levels.
36	PVCS_E_PROMO_NO_PARENT	You tried to promote from the promotion group that is at the top of the promotion model.
37	PVCS_E_PROMO_NO_NODE	The specified promotion group does not exist.
38	PVCS_E_PROMO_ERROR	Obsolete.
39	PVCS_E_NO_ALIAS	The **PvcsQueryAlias** parameter cannot find the alias.
40	PVCS_E_GROUP_EXISTS	The promotion group already exists.
41	PVCS_E_NO_GROUP	The specified promotion group does not exist in the archive.
42	PVCS_E_NO_NETWARE_DLL	**PvcsInit** returns this code under Windows if the NetWare drivers are loaded, but NWNETAPI.DLL is not in the PATH. This prevents Version Manager from using any non-file I/O services, such as getting user IDs or creating semaphores.
43	PVCS_E_NO_JOURNAL_FILES	**PvcsListJournal** returns this error when you do not specify a journal file, and it cannot find a default file.
44	PVCS_E_BAD_JOURNAL_FILE	**PvcsListJournal** returns this error code when it encounters a format error in a journal file, which may happen if someone edits the file.
45	PVCS_E_NOT_DEV_GROUP	Developer's Toolkit cannot lock a revision because the specified group is not a development group.

Code	Return Value	Description
46	PVCS_E_GROUP_LOCKED	The program could not place a lock on a revision because the specified development group is already associated with another lock.
47	PVCS_E_BRANCHWARN	**PvcsGetRevision** and **PvcsLockRevisionGroup** return this code when a revision that you are locking would cause a branch to be created when it is checked in, and the BranchWarn directive is in effect.
48	PVCS_E_CANT_OPEN_ACCESSDB	Developer's Toolkit cannot open the access control database.
49	PVCS_E_PROMOTE_XBRANCH	**PvcsPromoteRevision** returns this error when you specify the PVCS_PROMOTE_NO_XBRANCH flag and a promotion would cross a branch.
50	PVCS_E_NO_USER_ID	Developer's Toolkit cannot find the user identification.
51	PVCS_E_OLD_WORKFILE	**PvcsPutRevision** cannot check in a revision because the workfile is older than the previous revision. To check it in, use the PVCS_PUT_FORCE_PUT flag.
52	PVCS_E_UNCHANGED_WORKFILE	The workfile is identical to the previous revision in the archive. Use the PVCS_PUT_FORCE_PUT flag to check in the file anyway.
53	PVCS_E_EXPECTING_FILENAME	**PvcsReportDifferences** and **PvcsMerge** return this code when you do not specify the required number of file names or revisions. **PvcsReportDifferences** must be able to identify two files by some combination of file names and revision parameters. **PvcsMerge** must be able to identify a base file and two branch files by some combination of file names and revision parameters.
55	PVCS_E_FILE_NOT_FOUND	A specified file cannot be found.
56	PVCS_E_BAD_REDIRECT	A redirection specification contains a syntax error, invalid file name, or invalid path name.
57	PVCS_E_CFG_INCLUDE	A configuration file contains a syntax error in the Include directive.

Code	Return Value	Description
58	PVCS_E_CANT_SHARE	The file system does not support file sharing.
59	PVCS_E_CANT_MAP	Developer's Toolkit cannot map to a network drive.
60	PVCS_E_CFG_COLUMNMASK	A configuration file contains a syntax error in the ColumnMask directive.
61	PVCS_E_SEMAPHORE_NOT_ALLOWED	The Semaphore directive specifies a type that is not allowed by the operating system.
62	PVCS_E_CFG_BAD_CONDITIONAL	A configuration file conditional construct contains a syntax or logic error.
63	PVCS_E_CFG_RENUMBER	A configuration file contains a syntax error in the Renumber directive.
64	PVCS_E_LINE_EDITOR	Developer's Toolkit cannot open the console for the line editor.
65	PVCS_E_CANT_CREATE_SEMAPHORE	Developer's Toolkit issues this error for any of the following reasons: You tried to create a semaphore, but it already exists. The NoSemaphore directive is enabled. You tried to use a nonexistent directory as the *archDir* parameter. You tried to use a nonexistent archive as the *archDir* parameter.
66	PVCS_E_CANT_DELETE_SEMAPHORE	You tried to delete a semaphore that does not exist.
67	PVCS_E_BAD_PRIVILEGE	A privilege statement in the access control database contains a syntax error.
68	PVCS_E_UNKNOWN_PRIVILEGE	Developer's Toolkit did not recognize a privilege in the *privilegeList* parameter as base, composite, or custom.
69	PVCS_E_BAD_ACCESS_GROUP	A group definition statement in the access control database contains a syntax error.
70	PVCS_E_BAD_ACCESS_USER	A user definition statement in the access control database contains a syntax error.
71	PVCS_E_WORK_MSG_SAME	The workfile and message file are the same.

Code	Return Value	Description
72	PVCS_E_NO_MSGFILE	Developer's Toolkit cannot find a message file.
73	PVCS_E_EDITFILE	The program cannot open a temporary file specified by the VCSEdit directive.
74	PVCS_E_CFG_VCSEDIT	A configuration file contains a syntax error in the VCSEdit directive.
75	PVCS_E_VCSEDIT_CANCEL	The editor returned an error code as specified by the VCSEdit directive.
76	PVCS_E_NOT_DIR	The specified directory does not exist.
77	PVCS_E_DISK_FULL	A write error occurred. The disk is probably full.
78	PVCS_E_BRANCH_REVISION	Version Manager cannot delete a revision because it contains a branch.
79	PVCS_E_PROCESS_ABORTED	The process was interrupted. For example, the user pressed Ctrl+C.
80	PVCS_E_NO_ACCESS_DB	You did not specify an access control database.
81	PVCS_E_BAD_ACCESS_DB	Corrupted access control database.
82	PVCS_E_USER_EXPIRED	Valid dates for this user have expired, and access to the access control database is denied.
83	PVCS_E_UNKNOWN_USER	Developer's Toolkit did not find the user in the database specified by the *userName* parameter.
84	PVCS_E_INVALID_PASSWORD	The user ID you specified does not exist in the access control database.
85	PVCS_E_DIAG_FILE	Developer's Toolkit cannot open the diagnostic file, tried to open it when it is already opened, or tried to close it and it was already closed.
86	PVCS_E_UNKNOWN_GROUP	Developer's Toolkit did not find the group in the database specified by the *groupName* parameter.
87	PVCS_E_USER_EXISTS	The user you wanted to add using the *userName* parameter already exists in the database.

Code	Return Value	Description
88	PVCS_E_PRIVILEGE_EXISTS	The name specified by the *newName* parameter for this privilege already exists in the database.
89	PVCS_E_BAD_DATABASE_HANDLE	Trying to update access control database that is open as read-only.
90	PVCS_E_DLL_INUSE	Trying to load the Toolkit DLL more than once (DLL is not reentrant).
91	PVCS_E_SCRIPT_NOT_FOUND	Developer's Toolkit could not open the file specified by the *ScriptName* parameter
92	PVCS_E_CANT_LOAD_DLL	Developer's Toolkit cannot load the Configuration Builder DLL.
94	PVCS_E_EVENT_REGISTERED	Developer's Toolkit tried to register an event more than once.
95	PVCS_E_NOT_IN_LIST	User ID was not found in AccessList.
96	PVCS_E_BAD_SERIAL_NUMBER	Developer's Toolkit detects a serial number mismatch.
97	PVCS_E_BAD_VCONFIG_FILE	The file specified by the *fileToConfigure* parameter cannot be configured.
98	PVCS_E_CANT_OPEN_VCONFIG_FILE	Developer's Toolkit cannot open the file specified by the *fileToConfigure* parameter to modify it.
99	PVCS_E_CANT_READ_VCONFIG_FILE	Developer's Toolkit either cannot find or read the file specified by the *fileToConfigure* parameter.
100	PVCS_E_CANT_WRITE_VCONFIG_FILE	Developer's Toolkit cannot write to the file specified by the *fileToConfigure* parameter.
101	PVCS_E_UNKNOWN_ID_SOURCE	Developer's Toolkit does not recognize the user identification source.
102	PVCS_E_PROJECT BUSY	Project semaphore exists in directory where archive update is to occur.
103	PVCS_E_TRIGGER_ABORTED	Developer's Toolkit returns this error if an action being performed aborts because an event trigger returns a non-zero exit code. For example, a **PvcsPutRevision()** might return this error if a *PrePut* event trigger returns a non-zero exit code.

Code	Return Value	Description
104	PVCS_E_TRIGGER_ISLVDOS_MISSING	Trigger execution failed because the ISLVDOS.386 VxD is not installed.
105	PVCS_E_TRIGGER_WEXECDOS_MISSING	Trigger execution failed because WEXECDOS.PIF or WEXECDOS.EXE does not exist in the executable directory.
106	PVCS_E_TRIGGER_FAILED	The trigger failed to execute for some other reason.
107	PVCS_E_CANT_OPEN_CONFIG_FILE	Can't open a configuration file.
108	PVCS_E_ACCESS_GROUP_EXISTS	Group already exists in access control database.
109	PVCS_E_DIRECTIVE_DISALLOWED	Configuration directive has been dissallowed.
110	PVCS_E_NOT_SUPPORTED	The action or item requested is not supported.
111	PVCS_E_NOT_DEFINED	The action or item requested is not defined.
112	PVCS_E_CHECKSUM_ERROR	Licensing error.
113	PVCS_E_LICENSE_ERROR	Licensing error.
254	PVCS_E_FUNCTION_NOT_FOUND	A wrapper function failed to find an address for function.
255	PVCS_E_DTK_NOT_LOADED	A wrapper function failed to load the toolkit.

Chapter 5

Data Structures

Data Structures

Data structures used by the API

This chapter lists the data structures used by Serena PVCS Version Manager Developer's Toolkit functions. These structures are defined in the files PVCSVM.H and PVCSCB.H.

ARCHIVEINFO

Archive information

```
typedef struct _ARCHIVEINFO {
  USHORT revcnt;          /* Number of revisions */
  USHORT lockers;         /* Number of locks */
  USHORT attributes;      /* Attribute bits */
  USHORT reclen;          /* Workfile fixed record length */
  PVCSDATE create_time;   /* Workfile creation time */
  USHORT renum_start_col; /* Renumber start column */
  USHORT renum_end_col;   /* Renumber end column */
  LONG   renum_start_val; /* Renumber starting value */
  LONG   renum_step_val;  /* Renumber increment */
  USHORT archive;         /* Offs to name of archive */
  USHORT workfile;        /* Offs to name of workfile */
  USHORT owner;           /* Offs to archive owner */
  USHORT access;          /* Offs to archive access list */
  USHORT cmt_str;         /* Offs to comment prefix str */
  USHORT newline;         /* Offs to newline string */
  USHORT diffmask;        /* Offs to column mask range */
  UCHAR  info[1];         /* Variable length information */
} ARCHIVEINFO;
```

ARCHIVEINFO Attribute Definitions

Attribute definitions

```
#define PVCS_ATTR_CHK_LOCK   0x0001 /* check lock on
           check-in */
#define PVCS_ATTR_WRT_PROT   0x0002 /* archive write
           protection */
#define PVCS_ATTR_EXCL_LOCK  0x0004 /* only one lock at
           a time */
#define PVCS_ATTR_EXP_KEYS    0x0008 /* expand keywords */
#define PVCS_ATTR_TRANSLATE   0x0010 /* do eol translation */
#define PVCS_ATTR_CMPRS_DELTA 0x0020 /* compress deltas */
#define PVCS_ATTR_CMPRS_TEXT 0x0040 /* compress full text
           revs */
```

CONFIG

Directive values

```
typedef struct _CONFIG {
```

```
    BOOL quiet;              /* Quiet */
    BOOL wrtProtArchive;     /* WriteProtect */
    BOOL checkLock;          /* CheckLock */
    BOOL branchWarn;         /* BranchWarn */
    BOOL ignorePath;         /* IgnorePath */
    BOOL forceUnlock;        /* ForceUnlock */
    BOOL autoCreate;         /* AutoCreate */
    BOOL firstMatch;         /* FirstMatch */
    BOOL delMsgFile;         /* DeleteMessageFile */
    BOOL useArchiveWork;     /* ArchiveWork */
    BOOL signon;             /* SignOn */
    BOOL ignIDCase;          /* NoCase VCSID */
    BOOL shareLocal;         /* Share Local */
    BOOL shareNet;           /* Share Network */
    BOOL memswap;            /* MemSwap */
    BOOL ctrlz;              /* Ctrlz */
    BOOL writable_workfile;  /* NoDeleteWork NoWriteProtect*/
    BOOL writable_accessdb;  /* AccessDB NoWriteProtect*/
    USHORT semaphoreLocal;   /* Semaphore Local */
    USHORT semaphoreNetwork; /* Semaphore Network */
    USHORT semRetry;         /* SemaphoreRetry */
    USHORT semDelay;         /* SemaphoreDelay */
    USHORT delWork;          /* DeleteWork */
    USHORT multiLockUser;    /* MultiLock User */
    USHORT multiLockRev;     /* MultiLock Revision */
    USHORT exclusiveLock;    /* ExclusiveLock */
    USHORT vloginInt;        /* VLogIn */
    LONG diag;               /* Internal diagnostic level */
    char pathSep;            /* PathSeparator */
    UCHAR *translate;        /* Translate */
    UCHAR *archsuf;          /* ArchivSuffix */
    UCHAR *msgsuf;           /* MessageSuffix */
    UCHAR *semsuf;           /* SemSuffix */
    UCHAR *defVers;          /* DefaultVersion */
    UCHAR *baseVers;         /* BaseVersion */
    UCHAR *branchVers;       /* BranchVersion */
    UCHAR *semDir;           /* SemaphoreDir */
    UCHAR *workDir;          /* WorkDir */
    UCHAR *archiveWorkDir;   /* ArchiveWork */
    UCHAR *vcsEdit;          /* VCSEdit */
    UCHAR *vcsid;            /* VCSID */
    UCHAR *owner;            /* Owner */
    UCHAR *journal;          /* Journal */
    UCHAR *newline;          /* NewLine */
    UCHAR *accessdb;         /* AccessDB */
    UCHAR *columnMask;       /* ColumnMask */
    UCHAR *renum;            /* Renumber */
    UCHAR *vcsdir;           /* VCSDir */
    UCHAR *compressImage;    /* CompressWorkImage */
    UCHAR *compressDelta;    /* CompressDelta */
    UCHAR *recordLength;     /* RecordLength */
    UCHAR *expKeywords;      /* ExpandKeywords */
    UCHAR *loginSource;      /* LogIn */
} CONFIG;
```

LOCK

Lock information

This structure is used by **PvcsGetLockInfo**. The buffer that you pass to **PvcsGetLockInfo** must be large enough to contain this structure, plus room for the *revision*, *newRevision*, and *lockerdata*. In other words, **PvcsGetLockInfo** uses the storage that you provide to return the actual lock data.

PvcsGetLockInfo can return information about more than one lock. If this happens, your buffer will contain an array of LOCK structures. The LOCK structures contain pointers which point to the actual lock data.

Allocate 64 bytes per lock record.

```
typedef struct _LOCK {
  UCHAR  *revision;     /* The locked revision */
  UCHAR  *newRevision;  /* Revision to create at check in */
  UCHAR  *locker;       /* User ID of the locker */
} LOCK;
```

Example

The following is a sample piece of code to walk the LOCK list:

```
LOCK *pLockInfo = malloc(256);
PvcsGetLockInfo(
    ARCHIVEHANDLE_NOT_OPEN,/* Archive handle */
    archive, /* Archive name
    NULL,  /* Revision (any revision) */
    NULL,  /* VCSID (any) */
     &lockCount,/* Count (unlimited) */
    pLockInfo, /* Buffer */
    sizeof(pLockInfo));/* Buffer length */
    while (lockCount-- > 0) {
      printf("Revision: %s  NewRevision: %s  locker: %s\n",
            pLockInfo->revision, pLockInfo->newRevision,
            pLockInfo->locker);

// Advance pointer to next lock structure
pLockInfo += 1;
)
```

PVCSDATE

Time/date stamp

```
typedef struct _PVCSDATE {
  unsigned year    : 7;  /*  Number of years since 1980 */
  unsigned month   : 4;  /*  Month (1-12) */
  unsigned day     : 5;  /*  Day of month (1-31) */
  unsigned hours   : 5;  /*  Hour of day (24 hr. clock) */
  unsigned minutes : 6;  /*  Minutes (0-59) */
  unsigned twosecs : 5;  /*  2-sec. increments (0-29) */
} PVCSDATE;
```

REVINFO

Revision information

```
typedef struct _REVINFO {
  USHORT  branch_count; /* Number of branches off this rev */
  USHORT  lock_count;   /* Number of locks on this rev */
  USHORT  level;        /* Revision tree level, 0 == trunk */
```

```
  PVCSDATE date;        /* Check in date of revision */
  PVCSDATE mdate;       /* Modification date of revision */
  USHORT  ord;          /* Ordinal number in archive */
  UCHAR   revstr[64];   /* Revision number (ASCII string)*/
} REVINFO;
```

Index

I

J

L

M

N

O

P

Q

R

S

T

U

V

W