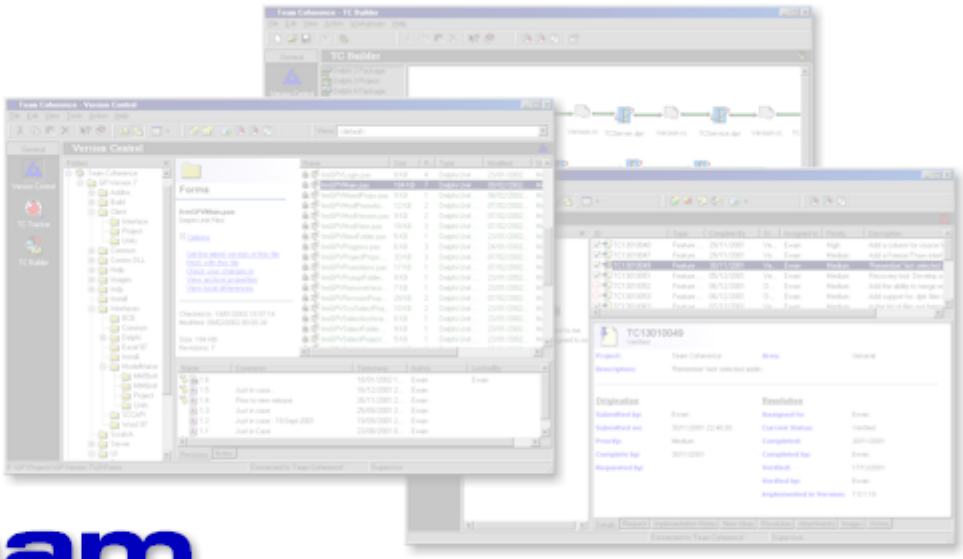
MCN Software Solutions



Team Coherence

www.teamcoherence.com

Version Manager API Reference

MCN Software Solutions

TC API Help

Printed: May 2015 in Scotland

Publisher

MCN Software
Suttie Way
Bridge of Allan
Stirlingshire
FK9 4NQ
Scotland

E-Mail: info@mcnsoftware.com
Website: http://www.mcnsoftware.com

Table of Contents

Foreword 0

Part I Introduction 5

1 How to buy Team Coherence 7

2 What is Team Coherence? 8

Part II API 10

1 Constants (TCVcsConst.pas) 10

2 Main Interface (GPVMain.dll) 12

TCVcsAbout 12
TCVcsBrowseFolder 12
TCVcsCheckInFile 13
TCVcsCheckInFiles 13
TCVcsCheckInFilesEx 14
TCVcsCheckOutFile 15
TCVcsCheckOutFiles 15
TCVcsCheckOutFilesEx 16
TCVcsCreateProject 16
TCVcsCurrentConnection 17
TCVcsCurrentUserName 17
TCVcsErrorString 17
TCVcsFileMainExtension 18
TCVcsGetCheckInInfo 18
TCVcsGetCheckOutInfo 19
TCVcsGetFileID 19
TCVcsGetFileList 20
TCVcsGetFileListForFolders 20
TCVcsGetFileStatus 21
TCVcsGetObjectType 22
TCVcsGetObjectVersions 22
TCVcsGetViewBaseID 23
TCVcsGetViews 23
TCVcsHaveLock 24
TCVcsInitialize 24
TCVcsIsFileArchived 25
TCVcsLogin 25
TCVcsLockFile 25
TCVcsProjectIDByName 26
TCVcsRefreshCache 26
TCVcsRelease 27
TCVcsRemoveFiles 27
TCVcsRenameFile 27
TCVcsSelectConnection 28
TCVcsSelectProject 28
TCVcsSelectView 29
TCVcsShowArchiveManager 29
TCVcsShowConnectionInfo 30
TCVcsShowHistory 30
TCVcsShowProperties 31
TCVcsShowWorkfileDifferences 32
TCVcsUndoCheckOutFile 32
TCVcsUndoCheckOutFiles 33

3 Direct Interface (GPVCCore.dll) ... 34
TCDAddVersionLabelEx ... 34
TCDVcsAddConnection ... 35
TCDVcsAddFolder ... 36
TCDVcsAddProject ... 36
TCDVcsAddVersionLabel ... 37
TCDVcsAttachVersionLabel ... 37
TCDVcsCheckInFile ... 38
TCDVcsCheckOutFile ... 39
TCDVcsCheckOutFileEx ... 39
TCDVcsConnect ... 40
TCDVcsCurrentConnection ... 41
TCDVcsDeleteConnection ... 41
TCDVcsDeleteFile ... 42
TCDVcsDeleteFolder ... 42
TCDVcsDeleteProject ... 43
TCDVcsDeleteVersionLabel ... 43
TCDVcsDetachVersionLabel ... 44
TCDVcsDisconnect ... 44
TCDVcsEnumConnections ... 45
TCDVcsEnumFiles ... 45
TCDVcsEnumFolders ... 47
TCDVcsEnumLabels ... 48
TCDVcsEnumLockedFiles ... 49
TCDVcsEnumProjects ... 50
TCDVcsEnumRevisions ... 51
TCDVcsEnumViews ... 52
TCDVcsErrorString ... 53
TCDVcsFileStatus ... 54
TCDVcsGetFileCheckoutPath ... 54
TCDVcsGetFileGroupExt ... 55
TCDVcsGetFileID ... 56
TCDVcsGetFileWorkingPath ... 56
TCDVcsGetObjectNotes ... 57
TCDVcsLock ... 57
TCDVcsModifyVersionLabel ... 58
TCDVcsMoveFile ... 58
TCDVcsNextPromotionLabel ... 59
TCDVcsPromote ... 59
TCDVcsRemoveObject ... 60
TCDVcsRenameFolder ... 60
TCDVcsRenameProject ... 61
TCDVcsSetObjectNotes ... 61
TCDVcsSetView ... 62
TCDVcsShareFile ... 62
TCDVcsUncheckOutFile ... 63
TCDVcsUnshareFile ... 63
TCDVcsUpdateConnection ... 64
4 Other ... 66
VcsExport ... 66
VcsImport ... 66
5 Type Definitions (TCVcsTypes.pas) ... 68
6 Utilities (TCVcsUtils.pas) ... 69
InitializeCheckInInfo ... 69
InitializeCheckOutInfo ... 69
ReleaseCheckInInfo ... 70
ReleaseCheckOutInfo ... 70

Part III Addins and Triggers 72

1 Client-side ... 73

Check In ... 74
Check Out / Get ... 76
Lock / Unlock ... 78
Promote ... 79
VcsBeginAction ... 81
VcsConfigureAddin ... 81
VcsEndAction ... 82
VcsInitEventAddin ... 82
VcsReleaseAddin ... 83
VcsShowAbout ... 83
VcsToLocalFormat ... 84

2 Server-side ... 86

TDisableServer ... 88
TEnableServer ... 89
TEnumerateFiles ... 89
TFlushBuffers ... 90
TGetFileInfo ... 90
TGetUserInfo ... 91
Thaw ... 92
TReinitialize ... 92
TValidateRepository ... 93
uUtils.pas ... 94
VcsAfterAssignVersion ... 94
VcsAfterCheckIn ... 95
VcsAfterCheckOut ... 96
VcsAfterCreateFolder ... 97
VcsAfterCreateProject ... 98
VcsAfterDeleteObject ... 98
VcsAfterFreeze ... 99
VcsAfterGet ... 100
VcsAfterLock ... 101
VcsAfterLogin ... 102
VcsAfterLogout ... 103
VcsAfterPromote ... 103
VcsAfterRemoveVersion ... 104
VcsAfterUnlock ... 105
VcsBeginAction ... 106
VcsConfigureAddin ... 107
VcsEndAction ... 107
VcsInitAddin ... 108
VcsReleaseAddin ... 109

Index 110

Introduction

Part I

1 Introduction

The Team Coherence Version Control API is a set of functions exported from the core Version Control DLL's that allow you to programatically access much of the functionality of Team Coherence. It is aimed at developers who need to integrate Team Coherence with their own applications or with applications that are not currently supported.

In addition, this help file will describe how to create extensions to Team Coherence that are triggered when certain actions occur. Triggers are aimed at developers who want to change the way Team Coherence works, or to add additional functionality related to version control actions. Sample trigger addins are available for download from our website and include full source code. Within client-side triggers, you have access to most of the Direct Interface API calls.

This help file documents the supported functions and describes their use in more detail. Although this document is aimed mainly at Delphi developers, all API functions are available from any other application that can access exported functions from Windows DLL's.

In addition to the API and use of Triggers, Team Coherence can also be controlled using the Command-line tool: TC.exe. For more information on the Team Coherence command line, see the TCCmd.chm help file. The command-line tool was created using the Team Coherence API and illustrates what can be done using the API.

What is documented

Throughout this document we will describe the API as it is handled from within a Delphi application. The Delphi interface to the API is wrapped by a set of files that simply convert the low-level functions into a more pascal-friendly format:

Filename	Description
TCVcsConst.pas	Defines the constants, including error codes, used in the API
TCVcsTypes.pas	Defines the types used by the API
TCVcsUtils.pas	Useful utility functions that help with conversion and initializing.
TCIntf.pas	The main API wrapper file for **GPVMain.dll**. This DLL contains the User Interface for Team Coherence and provides a higher-level interface. It also re-exports the functionality provided in the Direct interface.
TCDirectIntf.pas	The API wrapper file for **GPVCCore.dll**. This DLL contains low-level functions to allow access to the Version Control functionality without using the User Interface.

Note that the functions available in **GPVCCore.dll** are also available in the higher level **GPVMain.dll**. The core API defined in GPVCCore.dll has a separate wrapper file simply so that it can be used without the overhead of loading the main DLL that contains the User Interface for Team Coherence.

In addition to controlling Version Manager from your own applications and tools, Version Manager also has support for Addins and Triggers. These allow you to extend the functionality of Version Manager. There are basically two types of addin:

Addin type	Description
Client Side	Client side addins and triggers allow you to modify actions before they are processed and allow you to carry out other actions before and after certain Version Control actions. Client side addins, as the name suggests, are installed on a per-client basis.
Server Side	Server side addins are installed on the server and, as well as being triggered by certain events, can be used to control the server in certain ways. Server-side addins do not require anything to be installed on the client applications.

Calling Convention

All functions are exported from the DLL's using the **stdcall** calling convention.

1.1 How to buy Team Coherence

You can order Team Coherence online directly from our home page.

Home page

www.teamcoherence.com

Email support

support@teamcoherence.com

Post

MCN Software Ltd
6 Suttie Way
Bridge of Allan
FK9 4NQ
Scotland

Fax

+44 (0)1786 834908

1.2 What is Team Coherence?

Team Coherence is the simplest to use Software Configuration Management (SCM) solution available today. Rather than turning SCM into a black art that requires weeks of training, enforces unacceptable restrictions on developers, and generally interferes with the development and testing processes, we have designed Team Coherence to be as easy to use as possible.

We have spent many years, and worked with many development teams, to make Team Coherence a real-world solution to what can be a complex process. Regardless of the size of your organization, Team Coherence 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.

Team Coherence has been designed with the future in mind. It is easily extendable through addins and has a modular architecture to allow for future enhancement.

Part II

2 API

2.1 Constants (TCVcsConst.pas)

This file contains the constant definitions for the API. Some of the important ones are:

Browse Folder Dialog box

```
btFolders   = 0;
btFiles     = 1;
```

Label Types

```
lt_VersionLabel      = 1;
lt_PromotionLabel    = 2;
```

View Types

```
vt_VersionLabel      = 1;
vt_PromotionLabel    = 2;
vt_Date              = 4;
```

Action ID's for multiple files

```
act_Checkin   = 1;
act_Checkout  = 2;
act_Get       = 3;
act_Promote   = 4;
act_Lock      = 5;
act_Unlock    = 6;
```

File Status

```
VCS_STATUS_INVALID        = -1;     // Status could not be obtained, don't rely on it
VCS_STATUS_NOTCONTROLLED  = $0000;  // File is not under source control
VCS_STATUS_CONTROLLED     = $0001;  // File is under source code control
VCS_STATUS_CHECKEDOUT     = $0002;  // Checked out to current user at local path
VCS_STATUS_OUTOTHER       = $0004;  // File is checked out to another user
VCS_STATUS_OUTMULTIPLE    = $0010;  // File is checked out to multiple people
VCS_STATUS_OUTOFDATE      = $0020;  // The file is not the most recent
VCS_STATUS_DELETED        = $0040;  // File has been deleted from the project
VCS_STATUS_FROZEN         = $0080;  // No more revisions allowed
VCS_STATUS_SHARED         = $0200;  // File is shared between projects
VCS_STATUS_MODIFIED       = $0800;  // Local file has been modified
VCS_STATUS_OUTBYUSER = $1000;  // File is checked out by current user someplace
```

Check In Flags

```
ci_CheckInIfUnchanged = 1;
ci_LeaveLocked = 2;
ci_MoveVersion = 4;
ci_ForceBranch = 8;
ci_MakeWorkfileReadonly = 16;
ci_DeleteWorkFile = 32;
```

History Report Columns

```
hrNone               = 0;
hrRevisionName       = 1;
hrComments           = 2;
hrTimestamp          = 4;
hrAuthor             = 8;
hrLockedBy           = 16;
hrVersionLabels      = 32;
hrPromotionLevels    = 64;
```

Error Codes

```
Err_UnknownError = -1;
```

```
Err_OK = 0;
Err_VersionAlreadyAssigned = 2;
Err_FileNotLockedBy = 3;
Err_FileNotChanged = 4;
Err_ErrorCreatingRevision = 5;
Err_CannotFindRevision = 6;
Err_ObjectHasLock = 10;
Err_InsufficientAccess = 14;
Err_UserAlreadyContained = 23;
Err_GroupAlreadyContained = 24;
Err_CannotDeleteSpecialGroup = 25;
Err_CannotDeleteSuperUser = 26;
Err_GroupContainsGroup = 27;
Err_RecordModifiedByOther = 28;
Err_InvalidPassword = 29;
Err_InvalidIndex = 30;
Err_LockAlreadyExists = 31;
Err_CannotFindLock = 32;
Err_FileNotLocked = 33;
Err_RevisionNotLocked = 34;
Err_RevisionNotLockedBy = 35;
Err_CannotLocateLabel = 38;
Err_InvalidName = 39;
Err_ObjectNotFound = 40;
Err_FileLockedByUser = 41;
Err_FileLocked = 42;
Err_ErrorCreatingFile = 43;
Err_PathNotSpecified = 44;
Err_NotCheckedOutTo = 45;
Err_FileIsWritable = 46;
Err_VersionNotFound = 47;
Err_NotLoggedIn = 48;
Err_ConnectionNotDefined = 49;
Err_LocalFileNotFound = 50;
Err_CannotFindVersion = 51;
Err_GroupNotFound = 52;
Err_CouldNotCreateGroup = 53;
Err_ObjectNameAlreadyInUse = 54;
Err_CannotFindView = 55;
Err_FileHasBeenRemoved = 56;
Err_RenameNotSupported = 57;
Err_RevisionBranched = 58;
Err_Offline = 59;
Err_FileNotShared = 60;
Err_ServerError = 61;
Err_FilesTheSame = 62;
Err_CannotRenameSharedFile = 63;
Err_FileGroupsNotSupported = 64;
Err_CannotCheckOutInPromotionView = 65;
Err_CannotCheckInInPromotionView = 66;
Err_CannotDeleteUserWithLocks = 67;
Err_HasOfflineLock = 68;
Err_CannotDelRevisionWithBranch = 69;
Err_RevisionIsLocked = 70;
Err_ServerInsufficientSpace = 71;
Err_ClientInsufficientSpace = 72;
Err_NoRevisions = 73;
Err_PersonalLicense = 74;
Err_NoLicenses = 75;
Err_ServerBusy = 76;
Err_FileFrozen = 77;
Err_InvalidConnection = 78;
Err_AlreadyConnected = 79;
Err_NotConnected = 80;
Err_CouldNotConnectToServer = 81;
Err_VersionDoesNotExist = 82;
```

2.2 Main Interface (GPVMain.dll)

This DLL exports functions that allow access to and control of the higher-level functionality of Team Coherence, including the User Interface. This section will describe the functions exported from this DLL in terms of how they are accessed from Delphi.

In the description of the function we include the **Delphi** definition of the function which is used to wrap the function that is exported from the DLL. We also include the definition of the function that is actually **Exported** from the DLL.

Where parameters are described, they are the parameters of the Delphi wrapper function and not that of the exported function. In most cases the parameters are the same (but with different types) for the exported function.

Notes

This file also exports faster versions of the Non-UI functions defined in Direct Interface. If using the **TCIntf.pas** wrapper for GPVMain.dll, you can mix and match the UI and Non-UI functions.

Linux Users

Currently, only the Direct API is supported under Linux.

2.2.1 TCVcsAbout

Displays the Team Coherence about box.

Delphi (TCIntf.pas)

```
procedure TCVcsAbout;
```

Parameters

None

Return Value

None

Exported

```
procedure TCVcsAbout; stdcall;
```

2.2.2 TCVcsBrowseFolder

Displays a dialog to allow selection of an object from the currently active repository.

Delphi (TCIntf.pas)

```
function TCVcsBrowseFolder( ACaption, Description, FileMask: String; DlgType: Word; var
AnID: Cardinal; var ObjectName: String; CanSelectProject: Boolean ): Boolean;
```

Parameters

Name	Description
ACaption	The caption for the dialog box
Description	The text to use for the description line in the dialog.
FileMask	If the dialog is for selecting archives, then this specifies the mask used to filter the displayed files. Defaults to *
DlgType	Type of dialog box to display: btFolders for selecting Folders and Projects. btFiles for selecting files.
AnID (Out)	Receives the ID of the selected Object
ObjectName (Out)	Receives the Name of the selected Object
CanSelectProject	If the dialog is of type btFolders, setting this True will allow selection of the root projects.

Return Value

True if successful.

Exported

```
function TCBrowseFolder( pCaption, pDescription, pFileMask: PChar; DlgType: Word; var
AnID: Cardinal; const pObjectName: PChar; CanSelectProject: Boolean ): Boolean; stdcall;
```

2.2.3 TCVcsCheckInFile

Checks in a file after displaying the Checkin dialog box.

Delphi (TCIntf.pas)

```
function TCVcsCheckInFile( AFile: String; AddNew: Boolean ): Integer;
```

Parameters

Name	Description
AFile	Full path to the local file
AddNew	If True, will add the file if it is not already archived

Return Value

Err_OK if successful or an error code indicating the error.

2.2.4 TCVcsCheckInFiles

Displays the Checkin dialog box before checking in the specified files.

Delphi (TCIntf.pas)

```
function TCVcsCheckInFiles( FileList: TStrings; AddNew, NewProject: Boolean ): Integer;
```

Parameters

Name	Description
FileList	List of files to check in
AddNew	If True, any files that do not already exist will be added to the repository.
NewProject	If True, and any of the files are new, will prompt for selection of a project

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

After the files are checked in, the Status dialog box will show the success or failure of each action. You then have the ability to retry the failed actions.

Exported

```
function TCVcsCheckInFiles( FileList: PChar; AddNew, NewProject: Boolean ): Integer;
stdcall;
```

2.2.5 TCVcsCheckInFilesEx

Checks in the specified files without displaying the CheckIn dialog box.

Delphi (TCIntf.pas)

```
function TCVcsCheckInFilesEx( ProjectID: Cardinal; FileList: TStrings; CheckInInfo:
PCheckInInfo; AddNew: Boolean ): Integer;
```

Parameters

Name	Description
ProjectID	ID of the project to check the files into
FileList	List of files to check in
CheckInInfo	A pointer to a structure that contains checkin information.
AddNew	If True, any files that do not already exist will be added to the repository.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The pointer PCheckInInfo points to a structure that contains checkin information, including comments. There is a utility function, InitializeCheckInInfo, that will initialize the memory required for this pointer and a function, ReleaseCheckInInfo , that releases the memory.

After the files are checked in, the Status dialog box will show the success or failure of each action. You then have the ability to retry the failed actions.

Exported

```
function TCVcsCheckInFilesEx( ProjectID: Cardinal; FileList: PChar; CheckInInfo:
```

```
PCheckInInfo; AddNew: Boolean ): Integer; stdcall;
```

2.2.6 TCVcsCheckOutFile

Checks out or Gets a file after displaying the CheckOut dailog box.

Delphi (TCIntf.pas)

```
function TCVcsCheckOutFile( AFile: String; Lock: Boolean ): Integer;
```

Parameters

Name	Description
AFile	Full path to the local file
Lock	If True, the local file will be made writable and the archive will be locked. If False, a Get is performed and the local file is made read-only.

Return Value

Err_OK if successful or an error code indicating the error.

2.2.7 TCVcsCheckOutFiles

Displays the CheckOut dialog box before checking out or getting the specified files.

Delphi (TCIntf.pas)

```
function TCVcsCheckOutFiles( FileList: TStrings; Lock: Boolean ): Integer;
```

Parameters

Name	Description
FileList	List of files to check in
Lock	If True, the local files will be made writable and the archives will be locked. If False, a Get is performed and the local files are made read-only.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

After the files are checked out, the Status dialog box will show the success or failure of each action. You then have the ability to retry the failed actions.

Exported

```
function TCVcsCheckOutFiles( FileList: PChar; Lock: Boolean ): Integer; stdcall;
```

2.2.8 TCVcsCheckOutFilesEx

Displays the CheckOut dialog box before checking out or getting the specified files.

Delphi (TCIntf.pas)

```
function TCVcsCheckOutFilesEx( FileList: TStrings; CheckOutInfo: PCheckOutInfo;
ShowStatus: Boolean = True; ShowProgress: Boolean = True ): Integer;
```

Parameters

Name	Description
FileList	List of files to check in
CheckOutInfo	A pointer to a structure that contains checkout information.
ShowStatus	If True, the Status dialog box will be displayed after all files have been checked out.
ShowProgress	If True, a progress window is displayed.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The pointer PCheckOutInfo points to a structure that contains checkout information, including comments and whether to lock the archive or not. There is a utility function, InitializeCheckInInfo, that will initialize the memory required for this pointer and a function, ReleaseCheckInInfo , that releases the memory.

After the files are checked out, and if ShowStatus is True, the Status dialog box will show the success or failure of each action. You then have the ability to retry the failed actions.

Exported

```
function TCVcsCheckOutFilesEx( FileList: PChar; CheckOutInfo: PCheckOutInfo; ShowStatus,
ShowProgress: Boolean ): Integer; stdcall;
```

2.2.9 TCVcsCreateProject

Create a new Project in the current repository.

Delphi (TCIntf.pas)

```
function TCVcsCreateProject( var ProjID: Cardinal; Name: String ): Integer;
```

Parameters

Name	Description
ProjID (Out)	If the function is successful, returns the ID of the newly created project.
Name	The name of the project to be created.

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCVcsCreateProject( var ProjID: Cardinal; pName: PChar ): Integer; stdcall;
```

2.2.10 TCVcsCurrentConnection

Returns the string representing the current repository connection

Delphi (TCIntf.pas)

```
function TCVcsCurrentConnection: String;
```

Parameters

None

Return Value

A string representing the current connection.

Exported

```
function TCVcsCurrentConnection( const pConnection: PChar ): Integer; stdcall;
```

2.2.11 TCVcsCurrentUserName

Returns the Name of the currently logged in user

Delphi (TCIntf.pas)

```
function TCVcsCurrentUserName: String;
```

Parameters

None

Return Value

The name of the user connected to the repository.

Exported

```
function TCVcsCurrentUserName( const pUserName: PChar ): Integer; stdcall;
```

2.2.12 TCVcsErrorString

Create a new Project in the current repository.

Delphi (TCIntf.pas)

```
function TCVcsErrorString( ID: Integer ): String;
```

Parameters

Name	Description
ID	Error Code

Return Value

The string representing the error.

Exported

```
procedure TCVcsErrorString( ID: Integer; const pMsg: PChar ); stdcall;
```

2.2.13 TCVcsFileMainExtension

Returns the main file extension for a file that is a member of a group.

Delphi (TCIntf.pas)

```
function TCVcsFileMainExtension( AFile: String ): String;
```

Parameters

Name	Description
AFile	Full local path of the file.

Return Value

The extension of the main file for this type of file.

Remarks

Since files are archived under their main extension, it is often important when building a list of local files that you ignore files that are a member of a Group, but do not have the main extension. This is because, when file groups are enabled on a repository, the associated files are archived together with the main file as a group and do not appear in the repository as separate entities.

For more information on File Groups, refer to the main Team Coherence help file.

Exported

```
function TCVcsFileMainExtension( pFile: PChar; const pExt: PChar ): Integer; stdcall;
```

2.2.14 TCVcsGetCheckInInfo

Displays the CheckIn dialog

Delphi (TCIntf.pas)

```
function TCVcsGetCheckInInfo( const Info: PCheckInInfo ): Boolean;
```

Parameters

Name	Description
Info (In/Out)	A pointer to a structure that contains checkin information.

Return Value

True if the user pressed the OK button, False otherwise.

Remarks

The pointer to the data structure should be initialized by calling InitializeCheckInInfo and, once finished with, destroyed using ReleaseCheckInInfo

Exported

```
function TCVcsGetCheckInInfo( const Info: PCheckInInfo ): Boolean; stdcall;
```

2.2.15 TCVcsGetCheckOutInfo

Displays the CheckOut dialog

Delphi (TCIntf.pas)

```
function TCVcsGetCheckOutInfo( const Info: PCheckOutInfo ): Boolean;
```

Parameters

Name	Description
Info (In/Out)	A pointer to a structure that contains checkout information.

Return Value

True if the user pressed the OK button, False otherwise.

Remarks

The pointer to the data structure should be initialized by calling InitializeCheckOutInfo and, once finished with, destroyed using ReleaseCheckOutInfo

Exported

```
function TCVcsGetCheckOutInfo( const Info: PCheckOutInfo ): Boolean; stdcall;
```

2.2.16 TCVcsGetFileID

Returns the ID of the specified file

Delphi (TCIntf.pas)

```
function TCVcsGetFileID( FileName: String ): Cardinal;
```

Parameters

Name	Description
FileName	Full path to the local file

Return Value

The ID of the file, or zero if it is not archived.

Exported

```
function TCVcsGetFileID( pFileName: PChar ): Cardinal; stdcall;
```

2.2.17 TCVcsGetFileList

Internal Only. Returns the list of files contained under the object identified by RootID

Delphi (TCIntf.pas)

```
function TCVcsGetFileList( RootID: Cardinal; const Files: TStrings ): Boolean;
```

Parameters

Name	Description
RootID	ID of the object to search
Files (In/Out)	List to return the located files.

Return Value

True is successful

Remarks

Reserved for internal use only. Returns a list of filenames contained by RootID, including files contained in any subfolders.

Exported

```
function TCVcsGetFileList( RootID: Cardinal; const pFiles: PChar; var Size: Integer ):
Boolean; stdcall;
```

2.2.18 TCVcsGetFileListForFolders

Internal Only. Returns the list of files contained by the passed folders

Delphi (TCIntf.pas)

```
function TCVcsGetFileListForFolders( RootID: Cardinal; const Folders, Files: TStrings;
Recursive: Boolean ): Boolean;
```

Parameters

Name	Description
RootID	ID of the object to search
Folders (In)	List of Folders to search
Files (In/Out)	List to return the located files.
Recursive	True to recursively check subfolders

Return Value

True is successful

Remarks

Reserved for internal use only. Returns a list of filenames contained by ProjectID, but restricted to those contained by the passed Folders.

Exported

```
function TCVcsGetFileListForFolders( RootID: Cardinal; const pFolders, pFiles: PChar; var
Size: Integer; Recursive: Boolean ): Boolean; stdcall;
```

2.2.19 TCVcsGetFileStatus

Returns the current status of the passed File

Delphi (TCIntf.pas)

```
function TCVcsGetFileStatus( FileID: Cardinal; var Locked, LockedByMe, MultipleLocks,
Modified, OutOfDate: Boolean ): Boolean;
```

Parameters

Name	Description
FileID	ID of the File
Locked (Out)	True if the file is locked by someone
LockedByMe (Out)	True if the file is Locked by the current user
MultipleLocks (Out)	True if more than one user has this file locked
Modified	True if the local copy of the file has been modified since the last checkout.
OutOfDate	True if there is a later revision of the file in the repository.

Return Value

True is successful

Exported

```
function TCVcsGetFileStatus( FileID: Cardinal; var Locked, LockedByMe, MultipleLocks,
Modified, OutOfDate: Boolean ): Boolean; stdcall;
```

2.2.20 TCVcsGetObjectType

Internal Only. Returns the Object type and ID of the passed File/Folder

Delphi (TCIntf.pas)

```
function TCVcsGetObjectType( var ObjID: Cardinal; ProjID: Cardinal; AName: String ):
Integer;
```

Parameters

Name	Description
ObjID (Out)	ID of the File or Folder
ProjID	ID of the Project to search
AName	Full path to the file or folder

Return Value

Returns the object type which can be one of: `0, na_File, na_Folder`

Remarks

Reserved for internal use only

Exported

```
function TCVcsGetObjectType( var ObjID: Cardinal; ProjID: Cardinal; pName: PChar ):
Integer; stdcall;
```

2.2.21 TCVcsGetObjectVersions

Returns the list of Version Labels contained by the passed ID.

Delphi (TCIntf.pas)

```
function TCVcsGetObjectVersions( const ID: Integer; const VersionList: TStrings ):
Integer;
```

Parameters

Name	Description
ID	ID of the object
VersionList (In/Out)	Returns the list of contained Version Labels

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The Objects property of the passed VersionList contains the ID's of the associated Version Labels

Exported

```
function TCVcsGetObjectVersions( const ID: Integer; const Stream: TStream ): Integer;
stdcall;
```

2.2.22 TCVcsGetViewBaseID

Returns the ID of the Version Label or Promotion Label that forms the basis of the current View.

Delphi (TCIntf.pas)

```
function TCVcsGetViewBaseID: Cardinal;
```

Parameters

None

Return Value

The base ID of the current View.

Remarks

Gets the object ID that represents the Version or Promotion level that the currently selected View is based upon, or zero if there is no View selected. For more information on Views, see the main Team Coherence help file.

Exported

```
function TCVcsGetViewBaseID: Cardinal; stdcall;
```

2.2.23 TCVcsGetViews

Returns the list of currently defined Views

Delphi (TCIntf.pas)

```
procedure TCVcsGetViews( const Views: TStrings; var ActiveIndex: Integer );
```

Parameters

Name	Description
Views (In/Out)	Returns the list of Views
ActiveIndex (Out)	Index of the currently active View

Return Value

None

Remarks

For more information on Views, see the main Team Coherence help file.

Exported

```
procedure TCVcsGetViews( const pViews: PChar; var ActiveIndex: Integer ); stdcall;
```

2.2.24 TCVcsHaveLock

Check if you have a lock on the file

Delphi (TCIntf.pas)

```
function TCVcsHaveLock( FileID: Cardinal ): Boolean;
```

Parameters

Name	Description
FileID	ID of the File

Return Value

True if you have a current lock on this file.

Exported

```
function TCVcsHaveLock( FileID: Cardinal ): Boolean; stdcall;
```

2.2.25 TCVcsInitialize

Initializes version control.

Delphi (TCIntf.pas)

```
function TCVcsInitialize( Handle: Cardinal ): Integer;
```

Parameters

Name	Description
Handle	ID of the File

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This function initializes the internal structures and loads the internal cache. This function should be called before any other functions are called.

May prompt for a Connection, Username and Password.

Exported

```
function TCVcsInitialize( Handle: Cardinal; LoadCache: Boolean; ProgressProc:
TSplashProgress ): Integer; stdcall;
```

2.2.26 TCVcsIsFileArchived

Check if the passed file is already archived

Delphi (TCIntf.pas)

```
function TCVcsIsFileArchived( FileName: String; var ID: Cardinal ): Boolean;
```

Parameters

Name	Description
FileName	Full path to the local file
ID (Out)	If the file is archived, returns the ID of the file.

Return Value

True if the file is already archived.

Exported

```
function TCVcsIsFileArchived( pFileName: PChar; var ID: Cardinal ): Boolean; stdcall;
```

2.2.27 TCVcsLogin

Displays the Login dialog to validate a Username/password

Delphi (TCIntf.pas)

```
function TCVcsLogin: Boolean;
```

Parameters

None

Return Value

True if the user successfully logged in.

Exported

```
function TCVcsLogin: Boolean; stdcall;
```

2.2.28 TCVcsLockFile

Locks or Unlocks the file passed in AFile without performing a Get or CheckOut.

Delphi (TCIntf.pas)

```
function TCVcsLockFile( AFile: String; Lock: Boolean ): Integer;
```

Parameters

Name	Description
AFile	The full local path to the file to be locked or unlocked
Lock	Lock the file or unlock it.

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCVcsLockFile( pFile: PChar; Lock: Boolean ): Integer; stdcall;
```

2.2.29 TCVcsProjectIDByName

Returns the ID of the Project

Delphi (TCIntf.pas)

```
function TCVcsProjectIDByName( AName: String ): Cardinal;
```

Parameters

Name	Description
AName	Name of the Project

Return Value

The ID of the passed project, or zero if it could not be found.

Exported

```
function TCVcsProjectIDByName( const pName: PChar ): Cardinal; stdcall;
```

2.2.30 TCVcsRefreshCache

Reloads the cache data. Useful if recent changes have been made by other users.

Delphi (TCIntf.pas)

```
procedure TCVcsRefreshCache;
```

Parameters

None

Return Value

None

Exported

```
procedure TCVcsRefreshCache; stdcall;
```

2.2.31 TCVcsRelease

Once done with the DLL, call this to release internal memory structures.

Delphi (TCIntf.pas)

```
procedure TCVcsRelease;
```

Parameters

None

Return Value

None

Exported

```
procedure TCVcsRelease; stdcall;
```

2.2.32 TCVcsRemoveFiles

Removes the passed files from the Project

Delphi (TCIntf.pas)

```
function TCVcsRemoveFiles( ProjectID: Cardinal; FileList: TStrings ): Integer;
```

Parameters

Name	Description
ProjectID	ID of the project
FileList	List of files to be removed

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This function does not physically delete anything, but adds the object to the Recycle bin so that it is possible to recover it at a later date.

Exported

```
function TCVcsRemoveFiles( ProjectID: Cardinal; FileList: PChar ): Integer; stdcall;
```

2.2.33 TCVcsRenameFile

Renames the passed file

Delphi (TCIntf.pas)

```
function TCVcsRenameFile( FileID: Cardinal; NewName: String ): Integer;
```

Parameters

Name	Description
FileID	ID of the file to be renamed
NewName	New name to assign to the file

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Renames the file, specified by FileID to the NewName. This does not affect the local file.

Exported

```
function TCVcsRenameFile( FileID: Cardinal; pNewName: PChar ): Integer; stdcall;
```

2.2.34 TCVcsSelectConnection

Selects the passed connection, or displays the Connection dialog box to allow the user to select a different connection.

Delphi (TCIntf.pas)

```
procedure TCVcsSelectConnection( Connection: String );
```

Parameters

Name	Description
Connection	Name of the connection to switch to

Return Value

None

Remarks

If the parameter **Connection** is blank, the connection dialog will be displayed to allow the user to select a connection or create a new one. If a valid Connection name is passed, TC will connect to that repository.

Exported

```
procedure TCVcsSelectConnection( pConnection: PChar ); stdcall;
```

2.2.35 TCVcsSelectProject

Internal Only. Displays a VSS style dialog box to allow the user to select an existing project.

Delphi (TCIntf.pas)

```
function TCVcsSelectProject( var ProjName: String; var ProjID: Cardinal; CanCreate:
Boolean ): Boolean;
```

Parameters

Name	Description
ProjName (Out)	Returns the name of the selected project
ProjID (Out)	Returns the ID of the selected project

Return Value

True if the user selected a project.

Remarks

Reserved for internal use only.

Exported

```
function TCVcsSelectProject( const pProjName: PChar; var ProjID: Cardinal; CanCreate:
Boolean ): Boolean; stdcall;
```

2.2.36 TCVcsSelectView

Makes a different View current.

Delphi (TCIntf.pas)

```
function TCVcsSelectView( ViewName: String ): Integer;
```

Parameters

Name	Description
ViewName	The Name of the View to select

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The default View, which is always available, has a name of <default>. This view includes all revisions of all files. For more information on Views, see the main Team Coherence help file.

Exported

```
function TCVcsSelectView( pViewName: PChar ): Integer; stdcall;
```

2.2.37 TCVcsShowArchiveManager

Executes the Team Coherence Version Manager application.

Delphi (TCIntf.pas)

```
procedure TCVcsShowArchiveManager( AFile: String );
```

Parameters

Name	Description
AFile	If this file is archived, the folder that contains it will be current.

Return Value

None

Remarks

It is important to note that this executes a separate application that maintains its own internal cache. Changes made in this application may not immediately be reflected in another. To make sure you are working with the latest version of the cache, use TCVcsRefreshCache

2.2.38 TCVcsShowConnectionInfo

Displays a dialog with current connection information.

Delphi (TCIntf.pas)

```
procedure TCVcsShowConnectionInfo;
```

Parameters

None

Return Value

None

Exported

```
procedure TCVcsShowConnectionInfo; stdcall;
```

2.2.39 TCVcsShowHistory

Displays the History report for the specified files, or outputs it to a file.

Delphi (TCIntf.pas)

```
procedure TCVcsShowHistory( FileList: TStrings; OutFile: String = '';
                            Columns: Integer = hrNone;
                            Version: Cardinal = 0;
                            Promotion: Cardinal = 0;
                           FromVer: Cardinal = 0; ToVer: Cardinal = 0;
                            FromDate: TDateTime = 0; ToDate: TDateTime = 0;
                           IncFromVersion: Boolean = False;
                            FileDetails: Boolean = False;
                            ShowNotes: Boolean = false );
```

Parameters

Name	Description
FileList	List of files to display the report for
OutFile	If this is not an empty string, the report will be generated and stored in this file. If you pass an empty string, the Report Criteria dialog will be displayed and the report can either be viewed or stored.
Columns	If OutFile is defined, specifies the columns to display. This can be a combination of the hrXXX constants defined in TCVcsConst.pas
Version	ID of a Version Label
Promotion	ID of a Promotion Label
FromVer, ToVer	Generates a report containing the history between the two labels.
FromDate, ToDate	Generates a report containing the history between the two dates
IncFromVersion	If generating a report for a Version range, includes the revision containing the initial Version Label
FileDetails	Includes extended file information in the file header
ShowNotes	If checked, any notes attached to the file will also be included.

Return Value

None

Remarks

If outputting directly to a file, all the rest of the parameters should be zero except for the parameters (or parameter pairs) that define your criteria.

Exported

```
procedure TCVcsShowHistory( pFileList, pOutFile: PChar;
                            Columns: Integer;
                            Version,
                            Promotion,
                            FromVer, ToVer: Cardinal;
                            FromDate, ToDate: Integer;
                            IncFromVersion,
                            FileDetails, ShowNotes: Boolean ); stdcall;
```

2.2.40 TCVcsShowProperties

Display the Properties dialog for the specified file.

Delphi (TCIntf.pas)

```
procedure TCVcsShowProperties( FileName: String );
```

Parameters

Name	Description
FileName	The full path of the file

Return Value

None

Exported

```
procedure TCVcsShowProperties( pFileName: PChar ); stdcall;
```

2.2.41 TCVcsShowWorkfileDifferences

Display the difference viewer to compare the specified revision with the workfile.

Delphi (TCIntf.pas)

```
procedure TCVcsShowWorkfileDifferences( FileName, Revision: String; ShowDiffs: Boolean;
var Status: Integer );
```

Parameters

Name	Description
FileName	The full path of the work file
Revision	The name of the Revision to compare (e.g. 1.3)
ShowDiffs	If True, displays the difference viewer. If False, returns the status of the files in **Status**.
Status (Out)	If the files are different Status will be 0, otherwise returns `Err_FilesTheSame`

Return Value

None

Exported

```
procedure TCVcsShowWorkfileDifferences( pFileName, pRevision: PChar; ShowDiffs: Boolean;
var Status: Integer ); stdcall;
```

2.2.42 TCVcsUndoCheckOutFile

Undo the last checkout on this file

Delphi (TCIntf.pas)

```
function TCVcsUndoCheckOutFile( AFile: String ): Integer;
```

Parameters

Name	Description
AFile	The full path to the local file

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

When you undo a checkout, all changes to the local file are lost. Be careful when using this function.

2.2.43 TCVcsUndoCheckOutFiles

Undo the last checkout on the specified files

Delphi (TCIntf.pas)

```
function TCVcsUndoCheckOutFiles( FileList: TStrings ): Integer;
```

Parameters

Name	Description
FileList	List of files

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

When you undo a checkout, all changes to the local file are lost. Be careful when using this function.

Exported

```
function TCVcsUndoCheckOutFiles( FileList: PChar ): Integer; stdcall;
```

2.3 Direct Interface (GPVCCore.dll)

This DLL exports functions that allow direct, non-UI, access to the main Version Control functionality of Team Coherence. This section will describe the functions exported from this DLL in terms of how they are accessed from Delphi.

In the description of the function we include the **Delphi** definition of the function which is used to wrap the function that is exported from the DLL. We also include the definition of the function that is actually **Exported** from the DLL.

Where parameters are described, they are the parameters of the Delphi wrapper function and not that of the exported function. In most cases the parameters are the same (but with different types) for the exported function.

To distinguish these functions from those that interface with the User Interface of Team Coherence, these functions are exported with a **TCDVcs** prefix.

Notes

Important: All of these functions can be accessed from client-side addins, with the exception of TCDVcsConnect and TCDVcsDisconnect. When accessing the API from within a client-side addin, you **must** use the functions exported from the **GPVCCore** DLL (those wrapped in the **TCDirectIntf.pas** file).

Most of the functions defined in this section use ID's to identify objects in the repository. Each object in the repository has a unique numeric ID. To get the ID's of objects in the repository use the EnumXXX functions to enumerate through the hierarchy.

The versions of these functions exported from the **GPVCCore** DLL (those wrapped in the **TCDirectIntf.pas** file) do not utilize a cache so, if using these methods in your own applications, it is advisable to maintain some sort of internal cache to link file objects with their ID's in order to improve performance when multiple lookups are done on filenames.

If using the version of these functions exported from the **GPVMain** DLL (those wrapped in the **TCIntf.pas** file), they have access to the internal cache of the User Interface and are therefore much quicker. The downside of using these versions is the increased memory usage.

Linux Users

Currently, only the Direct API is supported under Linux.

2.3.1 TCDAddVersionLabelEx

Create a new Version Label in the current repository.

Delphi (TCDirectIntf.pas)

```
function VcsAddVersionLabelEx( var LabelID: Cardinal; Name, Comments: String; ProjectID:
Cardinal ): Integer;
```

Parameters

Name	Description
LabelID (Out)	If the function is successful, returns the ID of the newly created label.
Name	The name of the label to be created.
Comments	Comments to associate with the new label.
ProjectID	Associates the newly created label with the specified Project

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDVcsAddVersionLabelEx( var LabelID: Cardinal; pName, pComments: PChar;
ProjectID: Cardinal ): Integer; stdcall;
```

2.3.2 TCDVcsAddConnection

Adds a new connection to the connection list.

Delphi (TCDirectIntf.pas)

```
function VcsAddConnection( Name, Description, Host: String; Port: Integer; Params:
String; var Msg: String ): Boolean;
```

Parameters

Name	Description
Name	The name to use for the new connection.
Description	A description of the connection.
Host	The Host that the repository is located on. Can be a hostname or an IP address.
Port	The port number that the repository server is listening on.
Params	Any other parameters describing the connection. This parameter is in the form of Name/Value pairs separated by a semi-colon.
Msg (out)	Returns any error message that might have occurred if the result is False

Return Value

True if successful. If an error occurs, the string describing the error is returned in Msg.

Remarks

Params is a semi-colon delimited list of additional parameters defining the connection. This should be passed in the form `'Name1=Value1;Name2=Value2'`... where the following names are valid:

Key - If the server you are connecting to requires you to enter a key for encryption, use this parameter

DUNConnection - If Dial-up Networking is required, the name of the connection

DUNUsername - The username required to validate the Dial-up Networking connection
DUNPassword - The password required to validate the Dial-up Networking connection

SocksVersion - If connecting through a SOCKS compliant server, the version that is supported. Can be V4, V4A or V5
SocksHost - The hostname of the SOCKS server
SocksPort - The port number of the SOCKS server
SocksUsername - The username required to validate the connection on the SOCKS server
SocksPassword - The password required to validate the connection on the SOCKS server

UseWindowsUser - If the value is set to **1**, login to this connection will use the current Windows username. If set to **0**, Username and password is required.

Exported

```
function TCDVcsAddConnection( pName, pDescription, pHost: PChar; Port: Integer; pParams:
PChar; const pMsg: PChar ): Boolean; stdcall;
```

2.3.3 TCDVcsAddFolder

Create a new Folder in the current repository.

Delphi (TCDirectIntf.pas)

```
function VcsAddFolder( var FolderID: Cardinal; ParentID: Cardinal; Name, Path: String ):
Integer;
```

Parameters

Name	Description
FolderID (Out)	If the function is successful, returns the ID of the newly created folder.
ParentID	The ID of the Project or Folder that will contain the new Folder.
Name	The Name for the new folder.
Path	Specifies the default working path for the new folder. If this is an empty string, the default path will be based on the new name and the path defined for the parent.

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDVcsAddFolder( var FolderID: Cardinal; ParentID: Cardinal; pName, pPath: PChar
): Integer; stdcall;
```

2.3.4 TCDVcsAddProject

Create a new Project in the current repository.

Delphi (TCDirectIntf.pas)

```
function VcsAddProject( var ProjID: Cardinal; Name: String ): Integer;
```

Parameters

Name	Description
ProjID (Out)	If the function is successful, returns the ID of the newly created project.
Name	The name of the project to be created.

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDVcsAddProject( var ProjID: Cardinal; pName: PChar ): Integer; stdcall;
```

2.3.5 TCDVcsAddVersionLabel

Create a new Version Label in the current repository.

Delphi (TCDirectIntf.pas)

```
function VcsAddVersionLabel( var LabelID: Cardinal; Name, Comments: String ): Integer;
```

Parameters

Name	Description
LabelID (Out)	If the function is successful, returns the ID of the newly created label.
Name	The name of the label to be created.
Comments	Comments to associate with the new label.

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDVcsAddVersionLabel( var LabelID: Cardinal; pName, pComments: PChar ):
Integer; stdcall;
```

2.3.6 TCDVcsAttachVersionLabel

Attaches a Version Label to a file or revision.

Delphi (TCDirectIntf.pas)

```
function VcsAttachVersionLabel( FileID, RevisionID, VersionID: Cardinal; CanMove: Boolean
): Integer;
```

Parameters

Name	Description
FileID	ID of the file
RevisionID	The revision ID to attach the label to. If zero, the label will be attached to the tip revision for this View.
VersionID	The ID for the Version Label to attach
CanMove	If True, and the Version Label has already been assigned to a revision of this file, it will be deleted before the label is reassigned to the new revision.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Depending on the current View, the 'tip revision' may not be the latest revision of the file. For more information on Views, refer to the main Team Coherence help file.

Exported

```
function TCDVcsAttachVersionLabel( FileID, RevisionID, VersionID: Cardinal; CanMove:
Boolean ): Integer; stdcall;
```

2.3.7 TCDVcsCheckInFile

Add or checkin a file.

Delphi (TCDirectIntf.pas)

```
function VcsCheckInFile( ProjID, FolderID: Cardinal; var FileID, RevisionID: Cardinal;
FileName: String; CheckInInfo: PCheckInInfo ): Integer;
```

Parameters

Name	Description
ProjID	If adding a new file, this should be the ID of the project you are adding the file to. Otherwise it should be zero.
FolderID	If adding a new file, this should be the ID of the folder you are adding the file to. Otherwise it should be zero.
FileID (In/Out)	If this is an existing file, this should contain the FileID of the file. If it is a new file, this will return the ID of the newly added file.
FileName	The full path to the local copy of the file
CheckInInfo	A pointer to a structure that contains checkin information.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The pointer PCheckInInfo points to a structure that contains checkin information, including comments. There is a utility function, InitializeCheckInInfo, that will initialize the memory required for this pointer and a function, ReleaseCheckInInfo , that releases the memory.

Exported

```
function TCDVcsCheckInFile( ProjID, FolderID: Cardinal; var FileID, RevisionID: Cardinal;
pFileName: PChar; CheckInInfo: PCheckInInfo ): Integer; stdcall;
```

2.3.8 TCDVcsCheckOutFile

Get or Check out a file.

Delphi (TCDirectIntf.pas)

```
function VcsCheckOutFile( FileID: Cardinal; var RevisionID: Cardinal; CheckOutInfo:
PCheckOutInfo ): Integer;
```

Parameters

Name	Description
FileID	The ID of the file to get/checkout.
RevisionID (Out)	Returns the ID of the revision that was retrieved.
CheckOutInfo	A pointer to a structure that contains checkout information.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The pointer PCheckOutInfo points to a structure that contains checkout information, including comments and whether to lock the archive or not. There is a utility function, InitializeCheckInInfo, that will initialize the memory required for this pointer and a function, ReleaseCheckInInfo , that releases the memory.

Exported

```
function TCDVcsCheckOutFile( FileID: Cardinal; var RevisionID: Cardinal; CheckOutInfo:
PCheckOutInfo ): Integer; stdcall;
```

2.3.9 TCDVcsCheckOutFileEx

Get or Check out a file. This is an extension of the basic TCDVcsCheckOutFile function and is intended to allow a Get or CheckOut of a file to be carried out to a different FileName as well as to another path.

Delphi (TCDirectIntf.pas)

```
function VcsCheckOutFileEx( FileID: Cardinal; var RevisionID: Cardinal; CheckOutInfo:
PCheckOutInfo; FileName: String ): Integer;
```

Parameters

Name	Description
FileID	The ID of the file to get/checkout.
RevisionID (Out)	Returns the ID of the revision that was retrieved.
CheckOutInfo	A pointer to a structure that contains checkout information.
FileName	Use this field to specify a different FileName (not including the path) for the Get/Checkout action

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The pointer PCheckOutInfo points to a structure that contains checkout information, including comments and whether to lock the archive or not. There is a utility function, InitializeCheckInInfo, that will initialize the memory required for this pointer and a function, ReleaseCheckInInfo , that releases the memory.

Exported

```
function TCDVcsCheckOutFileEx( FileID: Cardinal; var RevisionID: Cardinal; CheckOutInfo:
PCheckOutInfo; pFileName: PChar ): Integer; stdcall;
```

2.3.10 TCDVcsConnect

Logon a user to a specific repository.

Delphi (TCDirectIntf.pas)

```
function VcsConnect( const Connection, Name, Password: String ): Integer;
```

Parameters

Name	Description
Connection	The name of the connection to use. This must have been defined previously using the Connection Manager in the IDE. If this is left blank, the last-used connection will be used.
Name	The username to login using
Password	The password associated with the user

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This function should be called before any other function.

Exported

```
function TCDVcsConnect( pConnection, pName, pPassword: PChar ): Integer; stdcall;
```

2.3.11 TCDVcsCurrentConnection

Returns information about the current connection

Delphi (TCDirectIntf.pas)

```
procedure VcsCurrentConnection( var Connection, Username: String; var Connected: Boolean
);
```

Parameters

Name	Description
Connection	Will contain the name of the current connection when the call returns
Username	Will contain the current Username when the call returns
Connected	Indicates the status of the connection.

Return Value

None

Exported

```
procedure TCDVcsCurrentConnection( const pConnection, pUsername: PChar; var Connected:
Boolean ); stdcall;
```

2.3.12 TCDVcsDeleteConnection

Deletes an existing repository connection from the connection list.

Delphi (TCDirectIntf.pas)

```
function VcsDeleteConnection( Name: String; var Msg: String ): Boolean;
```

Parameters

Name	Description
Name	The name of the connection to delete.
Msg (out)	Returns any error message that might have occurred if the result is False

Return Value

True if successful. If an error occurs, the string describing the error is returned in Msg.

Exported

```
function TCDVcsDeleteConnection( pName: PChar; const pMsg: PChar ): Boolean; stdcall;
```

2.3.13 TCDVcsDeleteFile

Permanently deletes an existing File.

Delphi (TCDirectIntf.pas)

```
function VcsDeleteFile( FileID: Cardinal ): Integer;
```

Parameters

Name	Description
FileID	The ID of the file to delete.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This function will permanently delete the file. If you want to remove the folder, without permanently deleting it, use TCDVcsRemoveObject instead.

If this file is part of a share, The link between it and the shares based on it will be broken.

Exported

```
function TCDVcsDeleteFile( FileID: Cardinal ): Integer; stdcall;
```

2.3.14 TCDVcsDeleteFolder

Permanently deletes an existing Folder.

Delphi (TCDirectIntf.pas)

```
function VcsDeleteFolder( FolderID: Cardinal ): Integer;
```

Parameters

Name	Description
FolderID	The ID of the folder to delete.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This function will permanently delete the folder, including all folders and files it contains. If you want to remove the folder, without permanently deleting it, use TCDVcsRemoveObject instead.

Exported

```
function TCDVcsDeleteFolder( FolderID: Cardinal ): Integer; stdcall;
```

2.3.15 TCDVcsDeleteProject

Permanently deletes an existing Project.

Delphi (TCDirectIntf.pas)

```
function VcsDeleteProject( ProjID: Cardinal ): Integer;
```

Parameters

Name	Description
ProjID	The ID of the project to delete.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This function will permanently delete the project, including all folders and files it contains. If you want to remove the project, without permanently deleting it, use TCDVcsRemoveObject instead.

Exported

```
function TCDVcsDeleteProject( ProjID: Cardinal ): Integer; stdcall;
```

2.3.16 TCDVcsDeleteVersionLabel

Permanently deletes an existing Version Label.

Delphi (TCDirectIntf.pas)

```
function VcsDeleteVersionLabel( LabelID: Cardinal ): Integer;
```

Parameters

Name	Description
LabelID	The ID of the label to delete.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This function will permanently delete the Version Label, and will remove the label from all objects it has been assigned to. If a View is based on this label, it will also be deleted.

Exported

```
function TCDVcsDeleteVersionLabel( LabelID: Cardinal ): Integer; stdcall;
```

2.3.17 TCDVcsDetachVersionLabel

Detaches a Version Label from a file or revision.

Delphi (TCDirectIntf.pas)

```
function VcsDetachVersionLabel( FileID, RevisionID, VersionID: Cardinal ): Integer;
```

Parameters

Name	Description
FileID	ID of the file
RevisionID	The revision ID to detach the label from. If zero, the label will be detached from whatever revision it was attached to.
VersionID	The ID for the Version Label to detach

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDVcsDetachVersionLabel( FileID, RevisionID, VersionID: Cardinal ): Integer;
stdcall;
```

2.3.18 TCDVcsDisconnect

Logs out the current user and disconnects from the repository.

Delphi (TCDirectIntf.pas)

```
function VcsDisconnect: Integer;
```

Parameters

None

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This should be the last call made before quitting your application..

Exported

```
function TCDVcsDisconnect: Integer; stdcall;
```

2.3.19 TCDVcsEnumConnections

Enumerates through the list of known repository connections and calls the passed function.

Delphi (TCDirectIntf.pas)

```
type
  TEnumConnections = function ( Data: Pointer; Name, Description, Host: String; Port:
Integer; Current: Boolean ): Boolean;

function VcsEnumConnections( EnumProc: TEnumConnections; Data: Pointer ): Integer;
```

Parameters

Name	Description
EnumProc	The procedure (prototype TEnumConnections) that will be called for each connection located.
Data	Pointer to additional information that can be passed to the EnumProc in the Data parameter.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

TEnumConnections is the prototype of the procedure that is called by VcsEnumConnections. This function should return True to continue enumerating the connections, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each connection:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
Name	Connection Name
Description	Description for the connection
Host	The host that the repository resides on. Can either be an IP address or a domain
Port	The Port number being used by the server.
Current	True if the connection described is currently active.

Exported

```
type
  TIntEnumConnections = function ( Context, Data: Pointer; pName, pDescription, pHost:
PChar; Port: Integer; Current: Boolean ): Boolean; stdcall;

function TCDVcsEnumConnections( Context, Data: Pointer; EnumProc: TIntEnumConnections ):
Integer; stdcall;
```

2.3.20 TCDVcsEnumFiles

Enumerates through the list of Files contained by the folder identified by RootID and calls the passed function.

Delphi (TCDirectIntf.pas)

```
type
  TEnumFiles = function ( Data: Pointer; Name, LocalPath, LockedBy: String; ID, ParentID,
AncestorID: Cardinal; Modified, Timestamp, CompressedSize, RevisionCount, ShareCount,
Status: Integer; IsVirtual, Frozen: Boolean ): Boolean;

function VcsEnumFiles( RootID: Cardinal; EnumProc: TEnumFiles; Data: Pointer; Recursive:
Boolean ): Integer;
```

Parameters

Name	Description
RootID	The ID of the Project or Folder that the enumeration starts from.
EnumProc	The procedure (prototype TEnumFiles) that will be called for each File found.
Data	Pointer to additional information that can be passed to the EnumProc in the Data parameter.
Recursive	If this is True, the enumeration will include all files contained in subfolders. If False, only the files contained directly by RootID will be enumerated.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

TEnumFiles is the prototype of the procedure that is called by VcsEnumFiles. This function should return True to continue enumerating the files, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each File:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
Name	Name of the file
LocalPath	The full path to the local copy of the file
LockedBy	Comma-separated list of users holding a lock on this file.
ID	The unique ID used to identify this file
ParentID	The unique ID of the parent Folder
AncestorID	If this is a shared file, indicates the ID of the shared file.
Modified	The date this file was last modified (FileDate)
Timestamp	The timestamp of the tip revision. E.g. last checkin
CompressedSize	The size, in bytes, of the tip revision
RevisionCount	The number of revisions contained by this file
ShareCount	If this is the root of a share, indicates the number of files dependent on this one.
Status	The current status of the local file
IsVirtual	True if this is a virtual file
Frozen	True if development has been halted on this file

Note that the enumeration only includes objects that are in the current View, and also excludes objects that have been removed and are in the recycle bin.

Exported

```
type
  TIntEnumFiles = function ( Context, Data: Pointer; pName, pLocalPath, pLockedBy: PChar;
ID, ParentID, AncestorID: Cardinal; Modified, Timestamp, CompressedSize, RevisionCount,
ShareCount, Status: Integer; IsVirtual, Frozen: Boolean ): Boolean; stdcall;

function TCDVcsEnumFiles( RootID: Cardinal; Context, Data: Pointer; EnumProc:
TIntEnumFiles; Recursive: Boolean ): Integer; stdcall;
```

2.3.21 TCDVcsEnumFolders

Enumerates through the list of Folders that are children of the passed ID and calls the passed function.

Delphi (TCDirectIntf.pas)

```
type
  TEnumFolders = function ( Data: Pointer; Name, TCPath, LocalFolder: String; ID,
ParentID: Cardinal; FolderCount, FileCount: Integer ): Boolean;

function VcsEnumFolders( RootID: Cardinal; EnumProc: TEnumFolders; Data: Pointer;
Recursive: Boolean ): Integer;
```

Parameters

Name	Description
RootID	The ID of the Project or Folder that the enumeration starts from.
EnumProc	The procedure (prototype TEnumFolders) that will be called for each Folder found.
Data	Pointer to additional information that can be passed to the EnumProc in the Data parameter.
Recursive	If this is True, the enumeration will include all subfolders. If False, only the immediate child folders of the object pointed to by RootID will be enumerated.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

TEnumFolders is the prototype of the procedure that is called by VcsEnumFolders. This function should return True to continue enumerating the folders, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each Folder:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
Name	Name of the Folder
TCPath	The full name of the folder as used by Team Coherence. e.g. //Project/Source/Units
LocalFolder	The local directory used as the default working directory for the current user.
ID	The unique ID used to identify this Folder
ParentID	The unique ID of the parent Folder or Project
FolderCount	The number of subfolders this Folder contains
FileCount	The number of Files directly contained by this Folder.

Note that the enumeration only includes objects that are in the current View, and also excludes objects that have been removed and are in the recycle bin.

Exported

```
type
  TIntEnumFolders = function ( Context, Data: Pointer; pName, pTCPath, pLocalFolder:
PChar; ID, ParentID: Cardinal; FolderCount, FileCount: Integer ): Boolean; stdcall;

function TCDVcsEnumFolders( RootID: Cardinal; Context, Data: Pointer; EnumProc:
TIntEnumFolders; Recursive: Boolean ): Integer; stdcall;
```

2.3.22 TCDVcsEnumLabels

Enumerates through the list of Version Labels or Promotion Labels in the repository. Can also be used to enumerate the labels attached to a specific revision of a file.

Delphi (TCDirectIntf.pas)

```
type
  TEnumLabels = function ( Data: Pointer; LabelType: Integer; Name, Comments: String; ID:
Cardinal; Timestamp: Integer ): Boolean;

function VcsEnumLabels( RootID, RevID: Cardinal; LabelType: Integer; EnumProc:
TEnumLabels; Data: Pointer ): Integer;
```

Parameters

Name	Description
RootID	Either an ID pointing to a Project, Folder, or a File, or Zero for all labels in the repository.
RevID	If RootID is the ID of a File this can be used to enumerate the labels attached to a specific revision.
LabelType	Type of Label to enumerate. Can be either lt_VersionLabel or lt_PromotionLabel
EnumProc	The procedure (prototype TEnumLabels) that will be called for each revision found.
Data	Pointer to additional information that can be passed to the EnumProc in the Data parameter.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

TEnumLabels is the prototype of the procedure that is called by VcsEnumLabels. This function should return True to continue enumerating the labels, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each Label:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
LabelType	Either lt_VersionLabel or lt_PromotionLabel
Name	Name of the Label
Comments	Comments assigned to the label
ID	Unique ID used to identify the label
Timestamp	The time the label was last modified

Note that the enumeration only includes objects that are in the current View, and also excludes objects that have been removed and are in the recycle bin.

Exported

```
type
  TIntEnumLabels = function ( Context, Data: Pointer; LabelType: Integer; pName,
pComments: PChar; ID: Cardinal; Timestamp: Integer ): Boolean; stdcall;

function TCDVcsEnumLabels( RootID, RevID: Cardinal; LabelType: Integer; Context, Data:
Pointer; EnumProc: TIntEnumLabels ): Integer; stdcall;
```

2.3.23 TCDVcsEnumLockedFiles

Enumerates through the list of Files locked by the user identified by UserID.

Delphi (TCDirectIntf.pas)

```
type
  TEnumLockedFiles = function ( Data: Pointer; Name, LockPath: String; ID, ParentID,
LockedRevID: Cardinal; Modified, RevisionCount, Status: Integer ): Boolean;

function VcsEnumLockedFiles( UserID: Cardinal; EnumProc: TEnumLockedFiles; Data: Pointer
): Integer;
```

Parameters

Name	Description
UserID	The ID of the User to retrieve the locked file list for. Currently the only valid value is 0 which is the currently logged in user.
EnumProc	The procedure (prototype TEnumLockedFiles) that will be called for each File found.
Data	Pointer to additional information that can be passed to the EnumProc in the Data parameter.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

TEnumLockedFiles is the prototype of the procedure that is called by VcsEnumLockedFiles. This function should return True to continue enumerating the files, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each File:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
Name	Name of the file
LockPath	The full path to the local location of the locked file. I.e. where the file was checked out to.
ID	The unique ID used to identify this file
ParentID	The unique ID of the parent Folder
LockedRevID	The ID of the revision that is locked
Modified	The date this file was last modified (FileDate)
RevisionCount	The number of revisions contained by this file
Status	The current status of the local file

Exported

```
type
  TIntEnumLockedFiles = function ( Context, Data: Pointer; pName, pLockPath: PChar; ID,
ParentID, LockedRevID: Cardinal; Modified, RevisionCount, Status: Integer ): Boolean;
stdcall;

function TCDVcsEnumLockedFiles( UserID: Cardinal; Context, Data: Pointer; EnumProc:
TIntEnumLockedFiles ): Integer; stdcall;
```

2.3.24 TCDVcsEnumProjects

Enumerates through the list of Projects in the current repository and calls the passed function.

Delphi (TCDirectIntf.pas)

```
type
  TEnumProjects = function ( Data: Pointer; Name: String; ID: Cardinal ): Boolean;

function VcsEnumProjects( EnumProc: TEnumProjects; Data: Pointer ): Integer;
```

Parameters

Name	Description
EnumProc	The procedure (prototype TEnumProjects) that will be called for each Project in the repository.
Data	Pointer to additional information that can be passed to the EnumProc in the Data parameter.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

TEnumProjects is the prototype of the procedure that is called by VcsEnumProjects. This function should return True to continue enumerating the projects, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each Project:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
Name	Name of the Project
ID	Unique ID used to identify the Project within the repository.

Note that the enumeration only includes objects that are in the current View, and also excludes objects that have been removed and are in the recycle bin.

Exported

```
type
  TIntEnumProjects = function ( Context, Data: Pointer; pName: PChar; ID: Cardinal ):
Boolean; stdcall;

function TCDVcsEnumProjects( Context, Data: Pointer; EnumProc: TIntEnumProjects ):
Integer; stdcall;
```

2.3.25 TCDVcsEnumRevisions

Enumerates through the list of Revisions contained by the file identified by FileID and calls the passed function.

Delphi (TCDirectIntf.pas)

```
type
  TEnumRevisions = function ( Data: Pointer; Name, Author, Comments, LockedBy: String;
ID, ParentID: Cardinal; Modified, Timestamp, CompressedSize, OriginalSize, CRC, VerCount,
PromoCount: Integer ): Boolean;

function VcsEnumRevisions( FileID: Cardinal; EnumProc: TEnumRevisions; Data: Pointer ):
Integer;
```

Parameters

Name	Description
FileID	The ID of the file that contains the revisions.
EnumProc	The procedure (prototype TEnumRevisions) that will be called for each revision found.
Data	Pointer to additional information that can be passed to the EnumProc in the Data parameter.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

TEnumRevisions is the prototype of the procedure that is called by VcsEnumRevisions. This function should return True to continue enumerating the revisions, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each Revision:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
Name	Name of the revision
Author	The author of this revision
Comments	The comments assigned when this revision was created.
LockedBy	Comma-separated list of users holding a lock on this revision.
ID	The unique ID of the Revision
ParentID	The ID of the file containing this revision.
Modified	The date this revision was last modified (FileDate)
Timestamp	The timestamp of the revision.
CompressedSize	The size, in bytes, of the data contained by the revision
OriginalSize	The original size of the files contained by this revision
CRC	The CRC value used to determine whether changes have been made to a file before it is checked in.
VerCount	The number of Version Labels attached to this revision
PromoCount	The number of Promotion Labels attached to this revision

Note that the enumeration only includes objects that are in the current View, and also excludes objects that have been removed and are in the recycle bin.

Exported

```
type
  TIntEnumRevisions = function ( Context, Data: Pointer; pName, pAuthor, pComments,
pLockedBy: PChar; ID, ParentID: Cardinal; Modified, Timestamp, CompressedSize,
OriginalSize, CRC, VerCount, PromoCount: Integer ): Boolean; stdcall;

function TCDVcsEnumRevisions( FileID: Cardinal; Context, Data: Pointer; EnumProc:
TIntEnumRevisions ): Integer; stdcall;
```

2.3.26 TCDVcsEnumViews

Enumerates through the defined Views in the repository.

Delphi (TCDirectIntf.pas)

```
type
  TEnumViews = function ( Data: Pointer; Name, Description: String; ID: Cardinal; Shared,
Current: Boolean ): Boolean;

function VcsEnumViews( EnumProc: TEnumViews; Data: Pointer ): Integer;
```

Parameters

Name	Description
EnumProc	The procedure (prototype TEnumViews) that will be called for each view found.
Data	Pointer to additional information that can be passed to the EnumProc in the Data parameter.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

TEnumViews is the prototype of the procedure that is called by VcsEnumViews. This function should return True to continue enumerating the labels, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each Label:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
Name	Name of the View
Description	Description of the View
ID	Unique ID used to identify the view
Shared	If True, indicates a View that is shared amongst all users.
Current	If True, indicates that this is the currently selected View

The view returned with the ID of Zero is the default view. This view includes all revisions in all files.

Exported

```
type
  TIntEnumViews = function ( Context, Data: Pointer; pName, pDescription: PChar; ID:
Cardinal; Shared, Current: Boolean ): Boolean; stdcall;

function TCDVcsEnumViews( Context, Data: Pointer; EnumProc: TIntEnumViews ): Integer;
stdcall;
```

2.3.27 TCDVcsErrorString

Most functions in the TC API return integer error codes. This function returns a string describing the error code.

Delphi (TCDirectIntf.pas)

```
function VcsErrorString( ErrorCode: Integer ): String;
```

Parameters

Name	Description
ErrorCode	The error code returned by an API function call.

Return Value

The string representation of the passed Error Code.

Exported

```
function TCDVcsErrorString( ErrorCode: Integer; const ErrorString: PChar; var Size:
Integer ): Integer; stdcall;
```

2.3.28 TCDVcsFileStatus

Returns information about the status of a local file.

Delphi (TCDirectIntf.pas)

```
function VcsFileStatus( var ParentID, FileID: Cardinal; var FilePath, LockedBy: String;
var Modified, Timestamp, CompressedSize, RevisionCount, ShareCount, Status: Integer; var
IsVirtual, Frozen: Boolean ): Integer;
```

Parameters

Name	Description
ParentID (In/Out)	Returns the unique ID of the parent Folder
FileID (In/Out)	If this is passed into this function, the FilePath returns the local path of the file. If zero, and the FilePath is specified, it will return the ID of the file.
FilePath (In/Out)	If this is passed into the function, and FileID is zero, FileID returns the ID of the specified file.
LockedBy (In/Out)	Who has this file locked.
Modified (Out)	The date this file was last modified (FileDate)
Timestamp (Out)	The timestamp of the tip revision. E.g. last checkin
CompressedSize (Out)	The size, in bytes, of the tip revision
RevisionCount (Out)	The number of revisions contained by this file
ShareCount (Out)	If this is the root of a share, indicates the number of files dependent on this one.
Status (Out)	The current status of the local file
IsVirtual (Out)	True if this is a virtual file
Frozen (Out)	True if development has been halted on this file

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

See also: TCDVcsEnumFiles.

Exported

```
function TCDVcsFileStatus( var ParentID, FileID: Cardinal; const pFilePath, pLockedBy:
PChar; var Modified, Timestamp, CompressedSize, RevisionCount, ShareCount, Status:
Integer; var IsVirtual, Frozen: Boolean ): Integer; stdcall;
```

2.3.29 TCDVcsGetFileCheckoutPath

If a file is locked by the current user, returns the path of the local file.

Delphi (TCDirectIntf.pas)

```
function VcsGetFileCheckoutPath( FileID: Cardinal; var LockedRevision: Cardinal; var
FullPath: String; var BranchedFrom, BranchTip: Cardinal ): Integer;
```

Parameters

Name	Description
FileID	ID of the file.
LockedRevision (Out)	Returns the revision of the file that is locked.
FilePath (Out)	Returns the full path of the local file.
BranchedFrom (Out)	Returns the ID of the root revision for this branch, or 0 if the locked revision is in the main development trunk
BranchTip (Out)	Returns the ID of the tip revision on this branch or, if not a branch, the main tip revision. If BranchTip <> LockedRevision, a branch will occur when the file is checked in.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Since a file can be checked out to any directory, this may or may not return the same path as VcsGetFileWorkingPath.

Exported

```
function TCDVcsGetFileCheckoutPath( FileID: Cardinal; var LockedRevision: Cardinal; const
pFilePath: PChar; var BranchedFrom, BranchTip: Cardinal ): Integer; stdcall;
```

2.3.30 TCDVcsGetFileGroupExt

Returns the main file extension for a file that is a member of a group.

Delphi (TCDirectIntf.pas)

```
function VcsGetFileGroupExt( FileName: String; var Ext: String ): Integer;
```

Parameters

Name	Description
FileName	Full local path of the file.
Ext (Out)	The file extension of the main file in the group.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Since files are archived under their main extension, it is often important when building a list of local files that you ignore files that are a member of a Group, but do not have the main extension. This is because, when file groups are enabled on a repository, the associated files are archived together with the main file as a group and do not appear in the repository as separate entities.

For more information on File Groups, refer to the main Team Coherence help file.

Exported

```
function TCDVcsGetFileGroupExt( pFileName: PChar; const pExt: PChar ): Integer; stdcall;
```

2.3.31 TCDVcsGetFileID

Given a local path, returns the ID of the file in the repository.

Delphi (TCDirectIntf.pas)

```
function VcsGetFileID( var FileID: Cardinal; FilePath: String ): Integer;
```

Parameters

Name	Description
FileID (Out)	If successful will contain the ID of the File object used to control the local file.
FilePath	Full path of the local file.

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDVcsGetFileID( var FileID: Cardinal; pFilePath: PChar ): Integer; stdcall;
```

2.3.32 TCDVcsGetFileWorkingPath

Returns the working path for the specified file.

Delphi (TCDirectIntf.pas)

```
function VcsGetFileWorkingPath( FileID: Cardinal; var FilePath: String ): Integer;
```

Parameters

Name	Description
FileID	ID of the file.
FilePath (Out)	Returns the full path of the local file.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Since the working path of a file is based on the View and User, the same FileID can return different paths in different contexts. For more information on Views, refer to the main Team Coherence help file.

Exported

```
function TCDVcsGetFileWorkingPath( FileID: Cardinal; const pFilePath: PChar ): Integer;
stdcall;
```

2.3.33 TCDVcsGetObjectNotes

Returns the notes associated with the object specified by ObjectID

Delphi (TCDirectIntf.pas)

```
function VcsGetObjectNotes( ObjectID: Cardinal; var Notes: String ): Integer;
```

Parameters

Name	Description
ObjectID	The ID of the object in question. Note that this function currently only works for Project, Folder, and File objects.
Notes (out)	Returns the Notes associated with the object

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Notes are stored in either HTML or plain text format.

Exported

```
function TCDVcsGetObjectNotes( ObjectID: Cardinal; const pNotes: PChar; var Length:
Integer ): Integer; StdCall
```

2.3.34 TCDVcsLock

Locks or unlocks a file.

Delphi (TCDirectIntf.pas)

```
function VcsLock( FileID, RevisionID: Cardinal; FullPath, LockComments: String; Lock,
UpdateStatusOnLock: Boolean ): Integer;
```

Parameters

Name	Description
FileID	The ID of the file to lock
RevisionID	The ID of the Revision to lock, or 0 for the tip revision.
FullPath	The full path to the file. I.e. the path you want to lock the file against.
LockComments	Comments you want to be associated with the lock you are placing.
Lock	True to lock the file, false to unlock.
UpdateStatusOnLock	If true, the readonly status of the local file will be updated to indicate the lock status.

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDVcsLock( FileID, RevisionID: Cardinal; pFullPath, pLockComments: PChar; Lock,
UpdateStatusOnLock: Boolean ): Integer; stdcall;
```

2.3.35 TCDVcsModifyVersionLabel

Modifies the name or comments for a Version Label

Delphi (TCDirectIntf.pas)

```
function VcsModifyVersionLabel( LabelID: Cardinal; Name, Comments: String ): Integer;
```

Parameters

Name	Description
LabelID	The ID of the label to be modified
Name	The new name for the label. If blank, the name will not be changed.
Comments	The new comments for this label. If blank, the comments will not be changed.

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDVcsModifyVersionLabel( LabelID: Cardinal; pName, pComments: PChar ): Integer;
stdcall;
```

2.3.36 TCDVcsMoveFile

Physically moves a file and its history to a new folder..

Delphi (TCDirectIntf.pas)

```
function VcsMoveFile( SourceFileID, TargetFolderID: Cardinal ): Integer;
```

Parameters

Name	Description
SourceFileID	The file to move
TargetFolderID	The ID of the folder to move the file to.

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDVcsMoveFile( SourceFileID, TargetFolderID: Cardinal ): Integer; stdcall;
```

2.3.37 TCDVcsNextPromotionLabel

Locates and returns the next higher Promotion Level based on the currently selected View.

Delphi (TCDirectIntf.pas)

```
function VcsNextPromotionLabel( var PromotionID: Cardinal; var PromotionLabel: String ):
Integer;
```

Parameters

Name	Description
PromotionID (Out)	Returns the ID of the next higher Promotion Level
PromotionLabel (Out)	Returns the Name associated with the next higher Promotion Level

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The next higher Promotion Level is defined by the currently selected View. If the current View is based on a Version Label, the next higher level will be the **first** Promotion Level. If the current View is based on a Promotion Level, the next higher level (if any) will be returned.

If the **<default>** View is current, or the current View is based on neither a Version Label or Promotion Level, first Promotion Level will be returned.

Exported

```
function TCDVcsNextPromotionLabel( var PromotionID: Cardinal; const pPromotionLabel:
PChar ): Integer; stdcall;
```

2.3.38 TCDVcsPromote

Promotes the specified file to the next Promotion Level. This command will promote the tip revision (as defined by the current View) to the next higher Promotion Level.

Delphi (TCDirectIntf.pas)

```
function VcsPromote( FileID, PromotionID: Cardinal ): Integer;
```

Parameters

Name	Description
FileID	The ID of the file to Promote
PromotionID	The ID of the promotion level

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Both the Tip Revision and the next higher Promotion Level are defined by the currently selected View. If the current View is based on a Version Label, the revision that has the Version Label attached will be promoted to the **first** Promotion Level. If the current View is based on a Promotion Level, the revision that is currently at that level will be promoted to the **next higher** Promotion Level.

If the **<default>** View is current, or the current View is based on neither a Version Label or Promotion Level, the tip revision of each file will be promoted.

Exported

```
function TCDVcsPromote( FileID, PromotionID: Cardinal ): Integer; stdcall;
```

2.3.39 TCDVcsRemoveObject

Removes an Object from the repository and adds it to the Recycle bin.

Delphi (TCDirectIntf.pas)

```
function VcsRemoveObject( ObjectID: Cardinal ): Integer;
```

Parameters

Name	Description
ObjectID	The unique ID of the object to be removed.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This function does not physically delete anything, but adds the object to the Recycle bin so that it is possible to recover it at a later date. The ObjectID can refer to a Project, Folder, or File.

Exported

```
function TCDVcsRemoveObject( ObjectID: Cardinal ): Integer; stdcall;
```

2.3.40 TCDVcsRenameFolder

Renames an existing Folder.

Delphi (TCDirectIntf.pas)

```
function VcsRenameFolder( FolderID: Cardinal; Name, Path: String; ReflectInSubFolders:
Boolean ): Integer;
```

Parameters

Name	Description
FolderID	The ID of the folder to be modified
Name	The new name for the folder. If blank, the name will not be changed.
Path	The new working path for this folder. If blank, the path will not be changed.
ReflectInSubFolders	If this is set to True, the changes will be reflected in all subfolders of the folder. If False, the changes only affect the folder identified by FolderID

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDVcsRenameFolder( FolderID: Cardinal; pName, pPath: PChar;
ReflectInSubFolders: Boolean ): Integer; stdcall;
```

2.3.41 TCDVcsRenameProject

Renames an existing Project.

Delphi (TCDirectIntf.pas)

```
function VcsRenameProject( ProjID: Cardinal; Name: String ): Integer;
```

Parameters

Name	Description
ProjID	The ID of the project to rename.
Name	The new name for the project.

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDVcsRenameProject( ProjID: Cardinal; pName: PChar ): Integer; stdcall;
```

2.3.42 TCDVcsSetObjectNotes

Returns the notes associated with the object specified by ObjectID

Delphi (TCDirectIntf.pas)

```
function VcsSetObjectNotes( ObjectID: Cardinal; Notes: String ): Integer;
```

Parameters

Name	Description
ObjectID	The ID of the object in question. Note that this function currently only works for Project, Folder, and File objects.
Notes	Returns the Notes to be associated with the object

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Notes are stored in either HTML or plain text format. Setting notes for an object will replace the notes already assigned to that object.

Exported

```
function TCDVcsSetObjectNotes( ObjectID: Cardinal; pNotes: PChar ): Integer; StdCall
```

2.3.43 TCDVcsSetView

Makes a different View current.

Delphi (TCDirectIntf.pas)

```
function VcsSetView( ViewID: Cardinal ): Integer;
```

Parameters

Name	Description
ViewID	The ID of the View to select

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The default View, which is always available, has an ID of zero. This view includes all revisions of all files. For more information on Views, see the main Team Coherence help file.

Exported

```
function TCDVcsSetView( ViewID: Cardinal ): Integer; stdcall;
```

2.3.44 TCDVcsShareFile

Creates a new, shared, file based on another file.

Delphi (TCDirectIntf.pas)

```
function VcsShareFile( var NewFileID: Cardinal; SourceFileID, TargetFolderID: Cardinal ):
Integer;
```

Parameters

Name	Description
NewFileID (Out)	Returns the ID of the newly created file.
SourceFileID	The file that the new file should be based on
TargetFolderID	The ID of the folder to create the new file in

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Shared files allow the same source and history to be shared amongst multiple projects. The new file is a placeholder for the source file and doesn't maintain its own history.

Exported

```
function TCDVcsShareFile( var NewFileID: Cardinal; SourceFileID, TargetFolderID: Cardinal
): Integer; stdcall;
```

2.3.45 TCDVcsUncheckOutFile

Undo a checkout on a file.

Delphi (TCDirectIntf.pas)

```
function VcsUncheckOutFile( FileID: Cardinal ): Integer;
```

Parameters

Name	Description
FileID	The ID of the file to undo the checkout on.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

When you undo a checkout, all changes to the local file are lost. Be careful when using this function.

Exported

```
function TCDVcsUncheckOutFile( FileID: Cardinal ): Integer; stdcall;
```

2.3.46 TCDVcsUnshareFile

Breaks the link between a shared file and the real file.

Delphi (TCDirectIntf.pas)

```
function VcsUnshareFile( FileID: Cardinal ): Integer;
```

Parameters

Name	Description
FileID	The ID of the file to Unshare

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

When you Unshare a file, its link with the real file is broken and it becomes independent with its own change history. When the link is initially broken, the history of the main file is copied to the new file.

Exported

```
function TCDVcsUnshareFile( FileID: Cardinal ): Integer; stdcall;
```

2.3.47 TCDVcsUpdateConnection

Updates an existing repository connection in the connection list.

Delphi (TCDirectIntf.pas)

```
function VcsUpdateConnection( Name: String; Params: String; var Msg: String ): Boolean;
```

Parameters

Name	Description
Name	The name of the connection to update.
Params	Any other parameters describing the connection. This parameter is in the form of Name/Value pairs separated by a semi-colon.
Msg (out)	Returns any error message that might have occurred if the result is False

Return Value

True if successful. If an error occurs, the string describing the error is returned in Msg.

Remarks

Params is a semi-colon delimited list of additional parameters defining the connection. This should be passed in the form `'Name1=Value1;Name2=Value2'`... where the following names are valid:

Description - A description for the connection
Host - The host that the repository is running on. Hostname or IP address
Port - The port number that the repository server is listening on.
Key - If the server you are connecting to requires you to enter a key for encryption, use this parameter

DUNConnection - If Dial-up Networking is required, the name of the connection
DUNUsername - The username required to validate the Dial-up Networking connection
DUNPassword - The password required to validate the Dial-up Networking connection

SocksVersion - If connecting through a SOCKS compliant server, the version that is supported. Can be V4, V4A or V5
SocksHost - The hostname of the SOCKS server
SocksPort - The port number of the SOCKS server
SocksUsername - The username required to validate the connection on the SOCKS server
SocksPassword - The password required to validate the connection on the SOCKS server

UseWindowsUser - If the value is set to **1**, login to this connection will use the current Windows username. If set to **0**, Username and password is required.

Exported

```
function TCDVcsUpdateConnection( pName: PChar; pParams: PChar; const pMsg: PChar ):
Boolean; stdcall;
```

2.4 Other

Functions listed here are combined functions that, because of their complexity and reliance on other exported functions, are listed separately. These functions are generally only available in the Delphi wrapper unit, but examination of the source should make it fairly easy to implement them in other languages.

2.4.1 VcsExport

Allows the exporting of a Project and/or Users and Groups from the current repository.

Available in both TCDirectIntf and TCIntf.

Delphi (TCIntf.pas, TCDirectIntf)

```
function VcsExport( What: Integer; ProjectID: Cardinal; FileName: String; Progress:
TImportExportProgress ): Integer;
```

Parameters

Name	Description
What	What describes what to export. Can be a combination of IE_HasUsers and IE_HasArchives.
ProjectID	If a Project is being included in the export, specifies the ID of the Project. If included, **What** must include HasArchives.
FileName	Specifies the file to hold the exported data.
Progress	The pointer to a Feedback function that allows feedback to be reported to the calling application. Can be **nil**.

Return Value

Err_OK if successful or an error code indicating the error.

Exported

Since this is a wrapper function, and utilizes server-side threads, it is implemented by calling several functions. For an example of how to user these functions, check the VcsExport function call in TCIntf.pas:

```
function TCDVcsBeginExport( What: Integer; ProjectID: Cardinal ): Integer; stdcall;
function TCDVcsGetExportProgress: Integer; stdcall;
function TCDVcsGetExportResult: Integer; stdcall;
function TCDVcsGetExportSize: Cardinal; stdcall;
function TCDVcsGetExportData( Start: Cardinal; Size: Integer; const Stream: TStream ):
Cardinal; stdcall;
function TCDVcsEndExport: Integer; stdcall;
```

2.4.2 VcsImport

Allows the Importing of a previously exported data file into a repository..

Available in both TCDirectIntf and TCIntf.

Delphi (TCIntf.pas, TCDirectIntf)

```
function VcsImport( FileName: String; Progress: TImportExportProgress ): Integer;
```

Parameters

Name	Description
FileName	Specifies the file to Import. The file contains information about what is to be imported.
Progress	The pointer to a Feedback function that allows feedback to be reported to the calling application. Can be **nil**.

Return Value

Err_OK if successful or an error code indicating the error.

Exported

Since this is a wrapper function, and utilizes server-side threads, it is implemented by calling several functions. For an example of how to user these functions, check the VcsImport function call in TCIntf.pas:

```
function TCDVcsBeginImport: Integer; stdcall;
function TCDVcsSetImportData( const Data: TStream ): Integer; stdcall;
function TCDVcsDoImport: Integer; stdcall;
function TCDVcsGetImportProgress: Integer; stdcall;
function TCDVcsGetImportResult: Integer; stdcall;
function TCDVcsEndImport: Integer; stdcall;
```

2.5 Type Definitions (TCVcsTypes.pas)

There are two main type definitions for the API. These are the records used to hold Checkin and Checkout information:

CheckOutInfo

```
type
  TCheckOutInfo = record
    Comments: PChar;
    Extra: PChar;
    Revision: PChar;
    LocalPath: PChar;
    VersionID: Cardinal;
    AssignVersionID: Cardinal;
    Overwrite: Boolean;
    Lock: Boolean;
  end;
  PCheckOutInfo = ^TCheckOutInfo;
```

CheckInInfo

```
type
  TCheckInInfo = record
    Comments: PChar;
    Extra: PChar;
    VersionID: Cardinal;
    Flags: Integer;
  end;
  PCheckInInfo = ^TCheckInInfo;
```

In both the above cases, instances of these records can be created using the InitializeCheckOutInfo and InitializeCheckInInfo and released with the corresponding Release functions.

These function simply initialize the records with typical sizes for the various fields. There is no restriction on the size of the Comments and Extra fields so you can assign your own sizes to these fields.

2.6 Utilities (TCVcsUtils.pas)

This file contains utility functions that may be useful in the context of the Team Coherence API. They are provided simply to make things a bit easier.

Four functions may be used frequently, though, in the context of checking files in and out:

InitializeCheckOutInfo
ReleaseCheckOutInfo

InitializeCheckInInfo
ReleaseCheckInInfo

2.6.1 InitializeCheckInInfo

Allocates memory for, and initializes, the TCheckInInfo structure.

Delphi (TCDVcsUtils.pas)

```
function InitializeCheckInInfo: PCheckOutInfo;
```

Parameters

None

Return Value

Pointer to an instance of the TCheckInInfo structure.

Remarks

Memory is not automatically released. To release memory allocated using this function, use the corresponding ReleaseCheckInInfo function.

2.6.2 InitializeCheckOutInfo

Allocates memory for, and initializes the TCheckOutInfo structure.

Delphi (TCDVcsUtils.pas)

```
function InitializeCheckOutInfo: PCheckOutInfo;
```

Parameters

None

Return Value

Pointer to an instance of the TCheckOutInfo structure.

Remarks

Memory is not automatically released. To release memory allocated using this function, use the corresponding ReleaseCheckOutInfo function.

2.6.3 ReleaseCheckInInfo

Releases the memory allocated with a previous call to InitializeCheckInInfo.

Delphi (TCDVcsUtils.pas)

```
procedure ReleaseCheckInInfo( Info: PCheckInInfo );
```

Parameters

None

Return Value

None

2.6.4 ReleaseCheckOutInfo

Releases the memory allocated with a previous call to InitializeCheckOutInfo.

Delphi (TCDVcsUtils.pas)

```
procedure ReleaseCheckOutInfo( Info: PCheckOutInfo );
```

Parameters

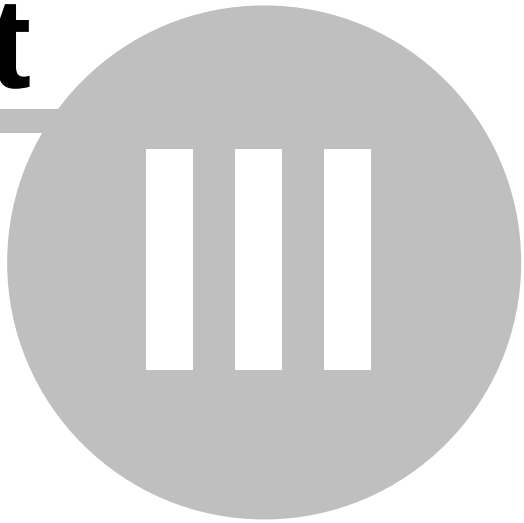
None

Return Value

None

Addins and Triggers

Part



3 Addins and Triggers

As well as supporting an API to allow your own applications and tools to access the TC Version Manager functionality, Version Manager also has support for third-party addins and triggers. These are basically standard Windows DLL's that can be triggered during certain Version Control actions and allow you to either modify the action, extend it, or simply report on the action in some way.

Within any client-side addin, you also have full access to the non-UI API defined in the Direct Interface.

There are basically two types of addin that can be created:

Client-side
Server-side

Sample addins, including source, can be downloaded from our web site and used as the basis for your own.

Linux Users

Currently, only the Direct API is supported under Linux.

3.1 Client-side

Client side addins are DLL's installed in the client application. Depending on the functionality exported from the addin, functions in the DLL's will be triggered before and after certain Version Control actions. Within these DLL's you have full access to the Direct API as defined in the wrapper file TCVcsDirect.pas.

To allow as many development tools as possible to be used to create addins, addins are implemented by exporting certain functions from standard Windows DLL's. For an Addin to be valid, it **must** export the following two functions:

VcsInitEventAddin
VcsReleaseAddin

The rest of the exported functions are optional and are usually called Before and After certain Version Control actions, including:

CheckIn
CheckOut
Get
Lock
Unlock
Promote

When multiple actions occur at the same time, i.e. checking out a folder, these actions are normally bracketed by the following calls:

VcsBeginAction
VcsEndAction

To support the use of code formatters (to allow individual users to format code the way they prefer), the following functions are called:

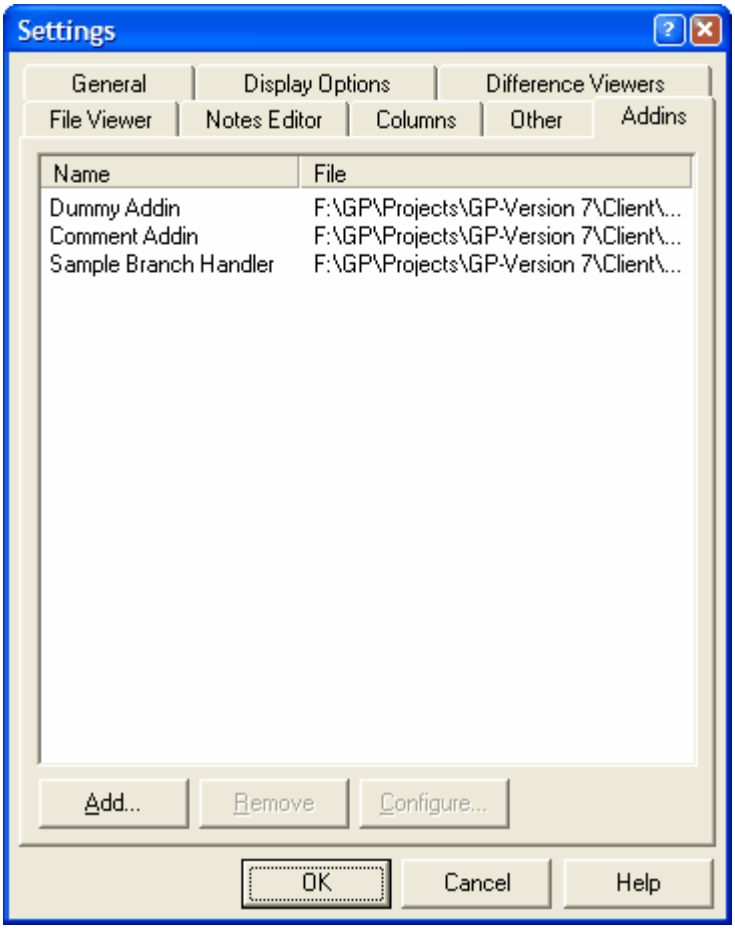
VcsToLocalFormat
VcsToServerFormat

Two other, optional functions can also be exported. These allow configuration of the addin, and the displaying of an About box:

VcsConfigureAddin
VcsShowAbout

Installing Client-side Addins

To install a client-side addin, start up Team Coherence Version Manager and select ***Tools/Options/Addins*** from the main menu. The following dialog will be displayed:



Clicking on the ***Add*** button allows you to select the DLL file that implements your addin. If the addin supports it, you can also click on the Configure button to configure any settings required by the addin.

3.1.1 Check In

If these methods are exported from the DLL, they will be called before and after a checkin event:

VcsBeforeCheckIn

```
function VcsBeforeCheckIn( ProjectID, FolderID, FileID, UserID, ViewID: Cardinal;
                           Path: PChar; const Info: PCheckInInfo;
                           const strError: PChar ): Boolean; stdcall; export;
```

This function is called before a CheckIn action occurs. The parameters define what file is being checked in and where it is being checked in from. You can use this information to validate the Checkin action or, using the Direct API, to modify the action before it occurs. If you want to stop the action occurring, return False from the function, otherwise, return True.

Parameters

Name	Description
ProjectID	ID of the containing Project
FolderID	ID of the Folder that contains the file
FileID	ID of the file
UserID	ID of the user carrying out the action
ViewID	If a View is applied, the ID of the View
Path	The location where the file is being checked in from
Info	A pointer to the TCheckInInfo structure containing information about how the checkin will be handled
strError	If you return False from this function to cause the action to fail, you can assign a description of the error here.

VcsAfterCheckIn

```
procedure VcsAfterCheckIn( ProjectID, FolderID, FileID, UserID, ViewID, RevisionID:
Cardinal;
                           Path: PChar; Result: Integer ); stdcall; export;
```

This procedure is called after a CheckIn action occurs. The parameters define what file that was checked in and where it was checked in from.

Parameters

Name	Description
ProjectID	ID of the containing Project
FolderID	ID of the Folder that contains the file
FileID	ID of the file
UserID	ID of the user that carried out the action
ViewID	If a View is applied, the ID of the View
RevisionID	The ID of the newly created revision
Path	The location where the file was checked in from
Result	The result code for the action. If the action was successful, Result will be Err_OK otherwise it will be one of the standard error codes defined in TCVcsConst.pas

Example (Delphi)

The following Before Checkin handler checks for locally stored comments and uses these to replace the main ones for this file:

```
function VcsBeforeCheckIn( ProjectID, FolderID, FileID, UserID, ViewID: Cardinal; Path:
PChar; const Info: PCheckInInfo; const strError: PChar ): Boolean;
var
  Lines: TStrings;
begin
  // Before we check in the files, we want to replace the default comment with the
  // comment associated with the file. This comment is generate using the
TCD5CommentServer
  // Delphi addin.
  if FileExists( ChangeFileExt( Path, '.cmt' ) ) then
  begin
    Lines := TStringList.Create;
```

```
    try
      Lines.LoadFromFile( ChangeFileExt( Path, '.cmt' ) );
      if Lines.Text <> '' then
      begin
        // Info.Comments has a max size of 65K by default
        if Length( Lines.Text ) < 65536 then
          StrCopy( Info.Comments, PChar( Lines.Text ) );
      end;
      // Remove the comment file as we are done with it...
      DeleteFile( ChangeFileExt( Path, '.cmt' ) )
    finally
      Lines.Free;
    end;
  end;
  Result := True;
end;
```

The following method is called after an automatic merge of a potential branch and simply updates the server with information:

```
procedure VcsAfterCheckIn( ProjectID, FolderID, FileID, UserID, ViewID, RevisionID:
Cardinal; Path: PChar; Result: Integer );
begin
  // Check if a merge took place, and update the server if necessary
  if FMerged and ( FMergedFrom > 0 ) then
    VcsSetMergeInfo( FileID, RevisionID, FMergedFrom );
  // Note that this is a temporary measure, and future releases may have
  // built-in merge functionality.
end;
```

3.1.2 Check Out / Get

If these methods are exported from the DLL, they will be called before and after a CheckOut or Get event:

VcsBeforeCheckOut

```
function VcsBeforeCheckOut( ProjectID, FolderID, FileID, UserID, ViewID: Cardinal;
                            const Info: PCheckOutInfo;
                            const strError: PChar ): Boolean; stdcall; export;
```

This function is called before a CheckOut or Get action occurs. The parameters define what file is being checked out and where it is being checked out to. You can use this information to validate the Checkout/Get action or, using the Direct API, to modify the action before it occurs. If you want to stop the action occurring, return False from the function, otherwise, return True.

Parameters

Name	Description
ProjectID	ID of the containing Project
FolderID	ID of the Folder that contains the file
FileID	ID of the file
UserID	ID of the user carrying out the action
ViewID	If a View is applied, the ID of the View
Info	A pointer to the TCheckOut structure containing information about how the checkout/get will be handled. Note: if Info.Lock is True the action is a CheckOut, otherwise it is a Get.
strError	If you return False from this function to cause the action to fail, you can assign a description of the error here.

VcsAfterCheckOut

```
procedure VcsAfterCheckOut( ProjectID, FolderID, FileID, UserID, ViewID: Cardinal;
                            Path: PChar; Lock: Boolean; Result: Integer ); stdcall;
export;
```

This procedure is called after a CheckOut or Get action occurs.

Parameters

Name	Description
ProjectID	ID of the containing Project
FolderID	ID of the Folder that contains the file
FileID	ID of the file
UserID	ID of the user that carried out the action
ViewID	If a View is applied, the ID of the View
Path	The location where the file was checked in from
Lock	If Lock is True, this was a CheckOut action, otherwise it was a Get.
Result	The result code for the action. If the action was successful, Result will be Err_OK otherwise it will be one of the standard error codes defined in TCVcsConst.pas

Example (Delphi)

The following Before Checkout handler simply modifies the Info structure to force the assigning of a predetermined Version Label during the Checkout:

```
function VcsBeforeCheckOut( ProjectID, FolderID, FileID, UserID, ViewID: Cardinal; const
Info: PCheckOutInfo; const strError: PChar ): Boolean;
begin
  if Info.Lock then
  begin
    // We are checking out the file. Assign a standard label (predetermined elsewhere)
    if Info.AssignVersionID = 0 then
      Info.AssignVersionID := 919283
  end;
  Result := True;
end;
```

The following method is called after a Get action and forces the local file to be writable:

```
procedure VcsAfterCheckOut( ProjectID, FolderID, FileID, UserID, ViewID: Cardinal; Path:
PChar; Lock: Boolean; Result: Integer );
begin
  if ( Result = Err_OK ) and ( not Lock ) then
  begin
    // We are doing a Get, so the file will be readonly. If the file is a dpr, then
    // make it writable
    if CompareText( ExtractFileExt( String( Path ) ), '.dpr' ) = 0 then
      FileSetAttr( String( Path ), FileGetAttr( String( Path ) ) and not faReadOnly );
  end;
end;
```

3.1.3 Lock / Unlock

If these methods are exported from the DLL, they will be called before and after a Lock or Unlock event. Note that these events occur as a result of the user using the Lock/Unlock commands and are ***not*** triggered before and after CheckIn/Out actions.

VcsBeforeLock

```
function VcsBeforeLock( FileID, RevisionID, UserID, ViewID: Cardinal;
                        Path: PChar; const LockComments, LockExtra: PChar;
                        Lock, Forced: Boolean; const strError: PChar ): Boolean; stdcall;
export;
```

This function is called before a Lock or Unlock action occurs. The parameters define what file is being checked out and where it is being checked out to. You can use this information to validate the Lock/Unlock action or, using the Direct API, to modify the action before it occurs. If you want to stop the action occuring, return False from the function, otherwise, return True.

Parameters

Name	Description
FileID	ID of the file being locked/unlocked
RevisionID	The RevisionID of the revision being locked. 0 if it is the tip revision
UserID	ID of the user carrying out the action
ViewID	If a View is applied, the ID of the View
Path	The path to the local copy of the file
LockComments	The comments being applied to this lock. You can modify this field
LockExtra	Currently for internal use only.
Lock	Whether the file is being Locked or Unlocked
Forced	If the file is being unlocked, whether an Admin user is forcing an unlock
strError	If you return False from this function to cause the action to fail, you can assign a description of the error here.

VcsAfterLock

```
procedure VcsAfterLock( FileID, RevisionID, UserID, ViewID: Cardinal;
                        Path: PChar; Lock: Boolean; Result: Integer ); stdcall; export;
```

This procedure is called after a Lock or Unlock action occurs.

Parameters

Name	Description
FileID	ID of the file
RevisionID	The RevisionID of the revision being locked. 0 if it is the tip revision
UserID	ID of the user that carried out the action
ViewID	If a View is applied, the ID of the View
Path	The location of the local file
Lock	If Lock is True, this was a Lock action, otherwise it was an Unlock.
Result	The result code for the action. If the action was successful, Result will be Err_OK otherwise it will be one of the standard error codes defined in TCVcsConst.pas

Example (Delphi)

The following Before Lock handler simply adds a comment to the lock if there are no comments defined:

```
function VcsBeforeLock( FileID, RevisionID, UserID, ViewID: Cardinal; Path: PChar; const
LockComments, LockExtra: PChar; Lock, Forced: Boolean; const strError: PChar ): Boolean;
begin
  if LockComments[ 0 ] = #0 then
  begin
    if Lock then
      StrCopy( LockComments, 'The file is being locked' )
    else
      StrCopy( LockComments, 'The file is being unlocked' );
  end;
  Result := True;
end;
```

The following method is called after a Lock or Unlock action and updates the local file accordingly:

```
procedure VcsAfterLock( FileID, RevisionID, UserID, ViewID: Cardinal; Path: PChar; Lock:
Boolean; Result: Integer );
begin
  if ( Result = Err_OK ) then
  begin
    // By default, Lock and Unlock may not update the status of the local file. Do it
here...
    if Lock then
      FileSetAttr( String( Path ), FileGetAttr( String( Path ) ) or faReadOnly )
    else
      FileSetAttr( String( Path ), FileGetAttr( String( Path ) ) and not faReadOnly );
  end;
end;
```

3.1.4 Promote

If these methods are exported from the DLL, they will be called before and after a Promote event.

VcsBeforePromote

```
function VcsBeforePromote( FileID, RevisionID, PromotionID, UserID, ViewID: Cardinal;
                           Path, Revision, PromoteTo: PChar; const strError: PChar ):
Boolean; stdcall; export;
```

This function is called before a Promote action occurs. The parameters define what file is being Promoted and which level it is being promoted to. You can use this information to validate the

Promote action or, using the Direct API, to modify the action before it occurs. If you want to stop the action occuring, return False from the function, otherwise, return True.

Parameters

Name	Description
FileID	ID of the file being promoted
RevisionID	The ID of the revision being promoted
PromotionID	The ID of the level the file is being promoted to.
UserID	ID of the user carrying out the action
ViewID	If a View is applied, the ID of the View
Path	The path to the local copy of the file
Revision	The name of the revision being promoted
PromoteTo	The name of the level the file is being promoted to
strError	If you return False from this function to cause the action to fail, you can assign a description of the error here.

VcsAfterPromote

```
procedure VcsAfterPromote( FileID, RevisionID, PromotionID, UserID, ViewID: Cardinal;
                           Path, Revision, PromoteTo: PChar; Result: Integer ); stdcall;
export;
```

This procedure is called after a Promote action occurs.

Parameters

Name	Description
FileID	ID of the file that was promoted
RevisionID	The ID of the revision that was promoted
PromotionID	The ID of the level the file was promoted to.
UserID	ID of the user that carried out the action
ViewID	If a View is applied, the ID of the View
Path	The location of the local file
Revision	The name of the revision that was promoted
PromoteTo	The name of the level the file was promoted to
Result	The result code for the action. If the action was successful, Result will be Err_OK otherwise it will be one of the standard error codes defined in TCVcsConst.pas

Example (Delphi)

The following Before Promote handler simply logs the action:

```
function VcsBeforePromote( FileID, RevisionID, PromotionID, UserID, ViewID: Cardinal;
Path, Revision, PromoteTo: PChar; const strError: PChar ): Boolean;
begin
  LogIt( Format( '---> Promoting revision %s of %s to %s', [ String( Revision ),
ExtractFileName( String( Path ) ), String( PromoteTo ) ] ) );
```

```
  Result := True;
end;
```

The following method is called after a Promote action and simply logs the result:

```
procedure VcsAfterPromote( FileID, RevisionID, PromotionID, UserID, ViewID: Cardinal;
Path, Revision, PromoteTo: PChar; Result: Integer );
begin
  if Result = Err_OK then
    LogIt( Format( '<--- Promoted revision %s of %s to %s', [ String( Revision ),
ExtractFileName( String( Path ) ), String( PromoteTo ) ] ) )
  else
    LogIt( Format( '<--- Error (%d) promoting revision %s of %s to %s', [ Result, String(
Revision ), ExtractFileName( String( Path ) ), String( PromoteTo ) ] ) );
end;
```

3.1.5 VcsBeginAction

To allow grouping of multiple actions, this method will be called (if present) before a group of actions occur.

```
procedure VcsBeginAction( ActionRef: Integer; Action: Word ); stdcall; export;
```

This method could be used to, for example, allocate resources that will be used when a set of actions will occur. VcsEndAction will always be called after a group of actions have occurred. Note that all actions for any ActionRef will be of the same type.

Parameters

Name	Description
ActionRef	A unique value that identifies the group of actions
Action	The type of action about to start. See TCVcsConst.pas for a list of the action types.

Example (Delphi)

```
var
  PromoteLog: TStrings = nil;

procedure VcsBeginAction( ActionRef: Integer; Action: Word );
begin
  // This procedure is called just before an action is carried out on multiple
  // files. ActionRef is a unique number passed into the procedure and allows
  // you to group multiple calls to VcsBefore... and VcsAfter... events.
  // See the file TCVcsConst for the act_XXXX values passed in the Action parameter
  if ( Action = act_Promote ) and ( not Assigned( PromoteLog ) ) then
    PromoteLog := TStringList.Create;
end;
```

3.1.6 VcsConfigureAddin

If your addin needs to be configured to work properly (i.e. an addin to E-Mail users), you should export this procedure. If you export this procedure, the **Configure...** button on the Addin dialog box will be enabled when your addin is selected.

```
procedure VcsConfigureAddin; stdcall; export;
```

You can display a dialog box to the user to allow them to configure your addin. Note that you are responsible for storage and retrieval of the configuration information.

Parameters

None

Example (Delphi)

```
procedure VcsConfigureAddin;
begin
  with TMyConfigDlg.Create( Application ) do
  try
    ShowModal;
  finally
    Free;
  end;
end;
```

3.1.7 VcsEndAction

To allow grouping of multiple actions, this method will be called (if present) after a group of actions occur.

```
procedure VcsEndAction( ActionRef: Integer; Action: Word ); stdcall; export;
```

This method should be used to, for example, released resources allocated during a call to VcsBeginAction.

Parameters

Name	Description
ActionRef	A unique value that identifies the group of actions
Action	The type of action just completed. See TCVcsConst.pas for a list of the action types.

Example (Delphi)

```
procedure VcsEndAction( ActionRef: Integer; Action: Word );
begin
  // This procedure is called just after an action is carried out on multiple
  // files.
  if ( Action = act_Promote ) and ( Assigned( PromoteLog ) ) then
    PromoteLog.Free;
  PromoteLog := nil;
end;
```

3.1.8 VcsInitEventAddin

This, required, function should be exported from all addins. It is used to initialize the addin and is called when the Addin is first loaded.

```
procedure VcsInitEventAddin( const pName: PChar;
                             GetUserProc: TGetUserInfo;
                             GetFileProc: TGetFileInfo ); stdcall; export;
```

If you allocate memory or resources in the Addin, you can use the VcsReleaseAddin function to release it.

Parameters

Name	Description
pName (out)	Return a descriptive name for this addin in this parameter.
GetUserProc	TC will pass in a pointer to a function of type TGetUserInfo that can be called from within the addin to get basic user information. In addition you have full access to the Direct API
Name	TC will pass in a pointer to a function of type TGetFileInfo that can be called from within the addin to get basic archive information. In addition you have full access to the Direct API

Example (Delphi)

```
type
  TGetUserInfo = function ( UID: Cardinal; const pName, pFullName, pEMail, pLocation,
pExtra: PChar ): Integer; stdcall;
  TGetFileInfo = function ( FileID: Cardinal; const pName, pLockedBy, pExtra: PChar; var
TimeStamp, FileDate: Integer; var Virtual, Frozen, Removed: Boolean ): Integer; stdcall;

var
  GetUserInfo: TGetUserInfo = nil;
  GetFileInfo: TGetFileInfo = nil;

procedure VcsInitEventAddin( const pName: PChar; GetUserProc: TGetUserInfo; GetFileProc:
TGetFileInfo );
begin
  // This procedure is called when the addin is first loaded
  // Use it to initialize internal variables.
  StrCopy( pName, 'Dummy Addin' );
  // Store the passed function pointers for later use...
  GetUserInfo := GetUserProc;
  GetFileInfo := GetFileProc;
end;
```

3.1.9 VcsReleaseAddin

This, required, function should be exported from all addins. It is called just before the Addin is unloaded and should be used to release any resources allocated while the addin was loaded..

```
procedure VcsReleaseAddin; stdcall; export;
```

Parameters

None

Example (Delphi)

```
procedure VcsReleaseAddin;
begin
  // This procedure is called just before the addin is unloaded.
  // Use it to release memory allocated while the addin is active
end;
```

3.1.10 VcsShowAbout

If you want to supply Copyright and Version information to users, you should export this procedure. If you export this procedure, the **About...** context menu on the Addin dialog box will be enabled when your addin is selected.

```
procedure VcsShowAbout; stdcall; export;
```

You can display a dialog box to the user to allow them to configure your addin. Note that you are responsible for storage and retrieval of the configuration information.

Parameters

None

Example (Delphi)

```
procedure VcsShowAbout;
begin
  with TMyAboutBox.Create( Application ) do
  try
    ShowModal;
  finally
    Free;
  end;
end;
```

3.1.11 VcsToLocalFormat

Team Coherence supports code formatters to allow you to work with shared files using the code layout you are used to. To handle this, you can implement the VcsToLocalFormat and VcsToServerFormat methods to reformat your code when it is retrieved and sent to the server.

VcsToLocalFormat

```
procedure VcsToLocalFormat( pLocalFile: PChar; Action: Word; var Converted: Boolean );
stdcall; export;
```

This function is called after a CheckOut or Get action to allow you to convert the retrieved file into the format the user wants to work with. If you implement this function, you should also implement the reverse function VcsToServerFormat function so that unnecessary revisions are not created.

Parameters

Name	Description
pLocalFile	The name of the local file to format.
Action	The type of action just completed. See TCVcsConst.pas for a list of the action types.
Converted	So that multiple formatters can be chained, you should return True if you modified the file. If the value is already True, it means another addin also modified it.

VcsToServerFormat

```
procedure VcsToServerFormat( pLocalFile: PChar; Action: Word; var Converted: Boolean );
stdcall; export;
```

This function is called before a CheckIn action to allow you to convert the local file back the format the server knows. This function is called before the CRC for the file is calculated so the ToLocal and ToServer file formatting tools should be 100% reversible to prevent unnecessary checkins occurring.

Parameters

Name	Description
pLocalFile	The name of the local file to format.
Action	The type of action about to occur. See TCVcsConst.pas for a list of the action types.
Converted	So that multiple formatters can be chained, you should return True if you modified the file. If the value is already True, it means another addin also modified it.

3.2 Server-side

Server side addins are DLL's installed on the Team Coherence Server. Depending on the functionality exported from the addin, functions in the DLL's will be triggered after certain Version Control actions and also allow you some control of the server in order to carry out maintenance actions.

An important thing to remember about server side addins is that they are running from within a Windows Service and so should **not** prompt users for information while it is running. In addition the addin should do what it needs to do as quickly as possible so as not to adversely affect the performance of the server.

To allow as many development tools as possible to be used to create addins, addins are implemented by exporting certain functions from standard Windows DLL's. For an Addin to be valid, it **must** export the following two functions:

VcsInitAddin
VcsReleaseAddin

The rest of the exported functions are optional and are usually called after certain Version Control actions, including:

VcsAfterCheckIn
VcsAfterCheckOut
VcsAfterGet
VcsAfterLock
VcsAfterUnlock
VcsAfterPromote
VcsAfterRemoveVersion
VcsAfterCreateProject
VcsAfterCreateFolder
VcsAfterDeleteObject
VcsAfterLogin
VcsAfterLogout

When multiple actions occur at the same time, i.e. checking out a folder, these actions are normally bracketed by the following calls:

VcsBeginAction
VcsEndAction

Another, optional function can also be exported. This allows configuration of the addin from the Repository Properties dialog box:

VcsConfigureAddin

During the initialization of the Addin, the server will pass in pointers to functions that can be used to control certain aspects of the server and to allow you access to functions to retrieve additional information. These include:

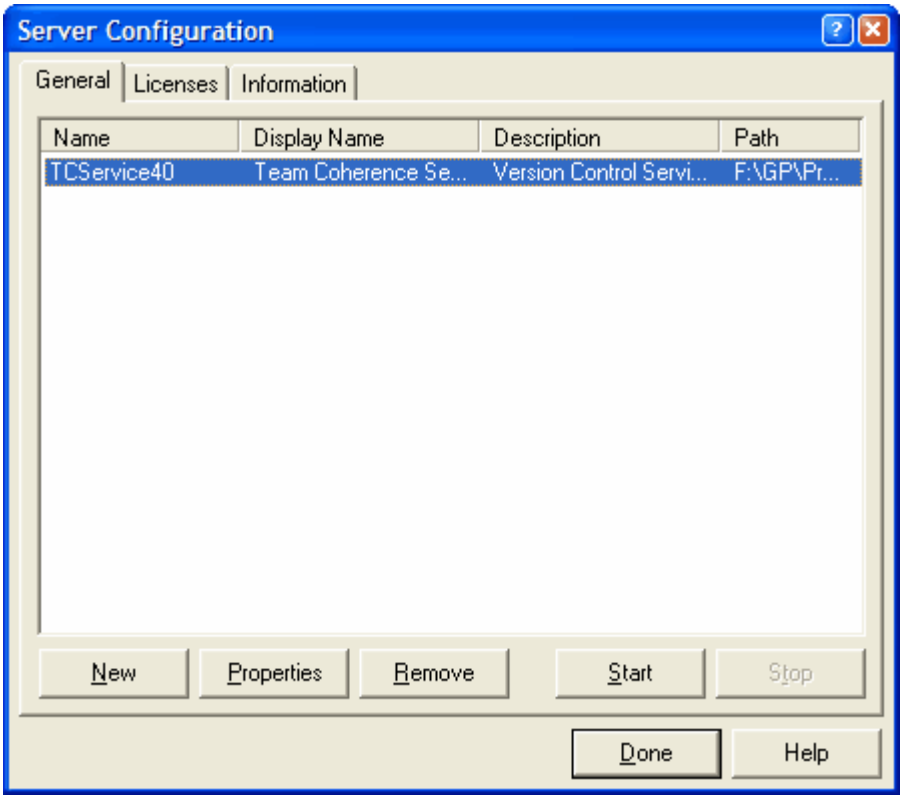
DisableServer
EnableServer
FlushBuffers
Reinitialize
ValidateRepository
EnumerateFiles
GetUserInfo
GetFileInfo

The server dll also exports certain functions which allow basic modification of repository data. These exported functions are wrapped in the file uUtils.pas which can be found in the Common

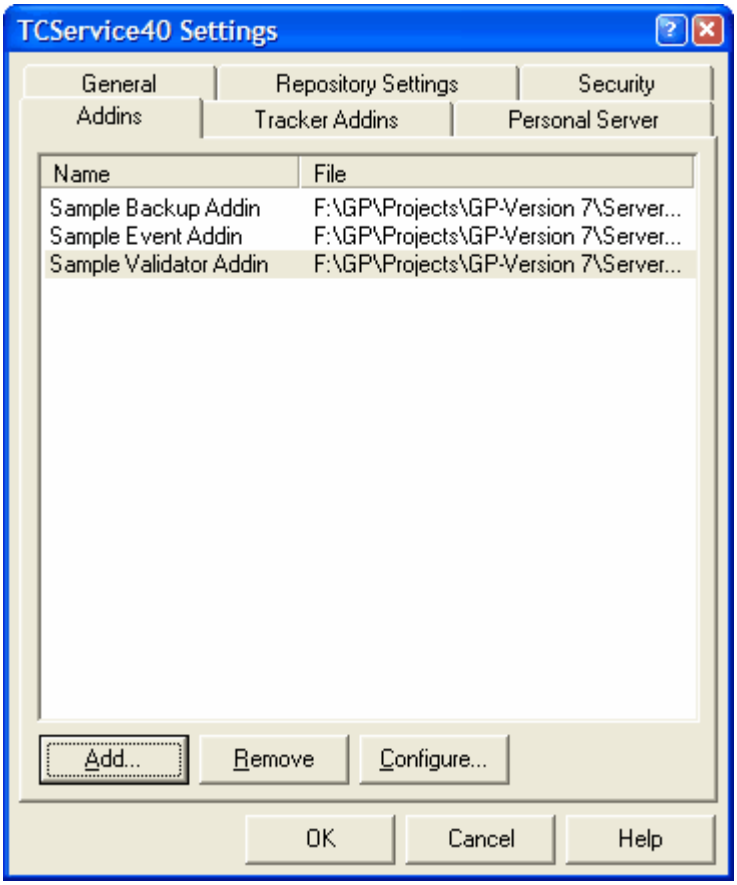
subdirectory of the example trigger source.

Installing Server-side Addins

To install a server-side addin, you must first of all stop the repository running. To do this, start up the Server Configuration dialog and select the repository:



With the repository selected, press ***Stop*** to shut down the running repository. To add an addin to the selected repository, Click on ***Properties*** to display the repository properties dialog:



Clicking on the ***Add*** button allows you to select the DLL file that implements your addin. If the addin supports it, you can also click on the Configure button to configure any settings required by the addin.

3.2.1 TDisableServer

A pointer to a method of this type is passed by the server as a parameter to the VcsInitAddin method. This method can be called to disable the server temporarily to, for example, allow maintenance functions to be called on the server without having to worry about users modifying the repository during the call. Calling this method prevents users carrying out actions.

Calls to this method should always be followed by a call to the method of type TEnableServer.

```
type
  TDisableServer = procedure; stdcall;
```

Parameters

None

Example (Delphi)

```
procedure TMaintenanceThread.Execute;
begin
  DoDisableServer;
  try
    DoFlushBuffers;
    ValidateTheRepository;
  finally
    DoEnableServer;
```

```
  end;
end;
```

3.2.2 TEnableServer

A pointer to a method of this type is passed by the server as a parameter to the VcsInitAddin method. This method can be called to enable the server after a call to the method of type TDisableServer.

```
type
  TEnableServer = procedure; stdcall;
```

Parameters

None

Example (Delphi)

```
procedure TMaintenanceThread.Execute;
begin
  DoDisableServer;
  try
    DoFlushBuffers;
    ValidateTheRepository;
  finally
    DoEnableServer;
  end;
end;
```

3.2.3 TEnumerateFiles

A pointer to a method of this type is passed by the server as a parameter to the VcsInitAddin method. This method can be called to enumerate through all files in the repository. Depending on the parameters you provide, it will enumerate Projects, Archive, or both. As each file is enumerated, it is passed back through a call to the procedure of type TEnumerateFilesProc.

```
type
  TEnumerateFilesProc = procedure( pFile: PChar ); stdcall;

  TEnumerateFiles = procedure ( pPath: PChar; EnumerateProjects, EnumerateArchives:
Boolean;
                                EnumerateProc: TEnumerateFilesProc ); stdcall;
```

Parameters

Name	Description
pPath	The path to the repository you want to enumerate. Normally the path passed into VcsInitAddin
EnumerateProjects	Set this to True to enumerate the project files
EnumerateArchives	Set this to True to enumerate individual file archives
EnumerateProc	Procedure of type TEnumerateFilesProc that is called for each Project or Archive that is enumerated. The parameter pFile contains the full path of the file being enumerated.

Example (Delphi)

```
procedure EnumFiles( pFile: PChar ); stdcall;
var
  BackupFile: String;
begin
  // Each file in the repository (depending on user options) will be passed to this
procedure.
  // Simple backup procedure to copy the passed file to another folder...
  BackupFile := FPath + ExtractFileName( String( pFile ) );
  if FilesDifferent( String( pFile ), BackupFile ) then
    CopyFile( pFile, PChar( BackupFile ), False );
end;

procedure TBackupThread.Backup;
begin
  FBusy := True;
  try
    // Backup can only be executed when the server is disabled
    DoDisableServer;
    try
      // Flush the buffers before running the Backup routine to make sure
      // the files are current
      DoFlushBuffers;
      // The backup will actually be done in the EnumFiles procedure
      DoEnumerateFiles( PChar( RepositoryPath ), True, True, EnumFiles );
    finally
      DoEnableServer;
    end;
  finally
    FBusy := False;
  end;
end;
```

3.2.4 TFlushBuffers

A pointer to a method of this type is passed by the server as a parameter to the VcsInitAddin method. This method can be called to reinitialize the repository. In effect, this call is equivalent to shutting down and restarting the server for the current repository.

```
type
  TReinitialize = function: Boolean; stdcall;
```

Parameters

None

Example (Delphi)

```
procedure TRestoreThread.Execute;
begin
  DoDisableServer;
  try
    DoFlushBuffers;
    // Validate and restore any corrupted archives
    try
      ValidateTheRepository;
    finally
      DoReinitialize;
    end;
  finally
    DoEnableServer;
  end;
end;
```

3.2.5 TGetFileInfo

A pointer to a method of this type is passed by the server as a parameter to the VcsInitAddin method. This method can be called to retrieve information about a file identified by FileID.

```
type
```

```
TGetFileInfo = function ( FileID: Cardinal; const pName, pLockedBy, pExtra: PChar;
                          var TimeStamp, FileDate: Integer;
                          var Virtual, Frozen, Removed: Boolean ): Integer; stdcall;
```

Parameters

Name	Description
FileID	The ID of the file
pName	Returns the name of the file
pLockedBy	Returns a string representing the list of users who have a lock on this file
pExtra	For internal use.
TimeStamp	Returns the timestamp of the tip revision
FileDate	Returns the local date/time of the archive.
Virtual	Returns True if the file is a virtual archive
Frozen	Returns True if the file has been frozen
Removed	Returns True if the file has been removed.

3.2.6 TGetUserInfo

A pointer to a method of this type is passed by the server as a parameter to the VcsInitAddin method. This method can be called to retrieve information about a user identified by UserID.

```
type
  TGetUserInfo = function ( UID: Cardinal; const pName, pFullName, pEMail, pLocation,
pExtra: PChar ): Integer; stdcall;
```

Parameters

Name	Description
UID	The ID of the user
pName	Returns the username of the user
pFullName	Returns the full name of the user
pEMail	Returns the E-Mail address of the user.
pLocation	Returns the Location of the user
pExtra	For internal use only.

Example (Delphi)

```
procedure VcsAfterPromote( pProject, pFolder, pObjectName, pUserName, pRevision,
pPromoteTo: PChar;
                           UserID, FileID, PromotionID: Cardinal );
var
  Name, FullName, EMail, Location, Extra: String;
begin
  // This method is called after a file is successfully promoted.
  GetUserInfo( UserID, Name, FullName, EMail, Location, Extra );
  LogIt( Format( '%s - ''%s (%s)'' Promoted revision %s of file ''%s'' to ''%s''',
                 [ DateTimeToStr( Now ), String( pUserName ), EMail,
                 String( pRevision ), String( pObjectName ),
                 String( pPromoteTo ) ] ) );
end;
```

3.2.7 Thaw

If this method is exported from the DLL, it will be called after an archive has been thawed. Thawing a file allows continued development on the file.

VcsAfterThaw

```
procedure VcsAfterThaw( pProject, pFolder, pObjectName, pUserName: PChar; UserID, FileID:
Cardinal ); stdcall; export; stdcall; export;
```

This procedure is called after an Remove Version Label action occurs. This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
pProject	The name of the Project containing the file
pFolder	The name of the Folder containing the file
pObjectName	The name of the file object that was thawed
pUserName	The name of the user that carried out the action
UserID	The ID of the user that carried out the action
FileID	The ID of the file

Example (Delphi)

The following method is called after a file has been thawed and simply logs the action to a local text file:

```
procedure VcsAfterThaw( pProject, pFolder, pObjectName, pUserName: PChar; UserID, FileID:
Cardinal );
var
  Name, FullName, EMail, Location, Extra: String;
begin
  // This method is called after a file is successfully thawed
  GetUserInfo( UserID, Name, FullName, EMail, Location, Extra );
  LogIt( Format( '%s - ''%s (%s)'' Thawed File ''%s''', [ DateTimeToStr( Now ), String(
pUserName ), EMail, String( pObjectName ) ] ) );
end;
```

3.2.8 TReinitialize

A pointer to a method of this type is passed by the server as a parameter to the VcsInitAddin method. This method can be called to reinitialize the repository. In effect, this call is equivalent to shutting down and restarting the server for the current repository.

```
type
  TReinitialize = function: Boolean; stdcall;
```

Parameters

None

Example (Delphi)

```
procedure TRestoreThread.Execute;
begin
  DoDisableServer;
  try
```

```
    DoFlushBuffers;
    // Validate and restore any corrupted archives
    try
      DoValidateRepository;
    finally
      DoReinitialize;
    end;
  finally
    DoEnableServer;
  end;
end;
```

3.2.9 TValidateRepository

A pointer to a method of this type is passed by the server as a parameter to the VcsInitAddin method. This method can be called to validate the state of a running repository. Depending on the parameters you provide, it will do the validation on the Project or Archive level. Feedback is provided through the callback method, of type TFeedbackProc, that you provide in the method call.

```
type
  TFeedbackProc = procedure( pText: PChar ); stdcall;

  TValidateRepository = procedure ( pPath, pFile: PChar; ValidateProjects: Boolean;
                                    ValidateArchives: Boolean; fbProc: TFeedbackProc );
stdcall;
```

Parameters

Name	Description
pPath	The path to the repository you want to validate. Normally the path passed into VcsInitAddin
pFile	Pass in the name of the file that the Validator can use to output results. Results are output as text and highlight any errors found. Can be **nil**.
ValidateProjects	Set this to True to validate the project files
ValidateArchives	Set this to True to validate individual file archives
fbProc	Procedure of type TFeedbackProc that is called for each Project or Archive that is validated. The parameter pText contains the result of the validation.

Example (Delphi)

```
procedure TValidateThread.Validate;
begin
  FBusy := True;
  try
    // Validation can only be executed when the server is disabled
    DoDisableServer;
    try
      // Flush the buffers before running the Validation routine as it
      // validates the physical files...
      DoFlushBuffers;
      // This addin cannot have a UI, so we will pass in a file
      if FileExists( RepositoryPath + 'Validate.txt' ) then
        DeleteFile( RepositoryPath + 'Validate.txt' );
      DoValidateRepository( PChar( RepositoryPath ), PChar( RepositoryPath +
'Validate.txt' ), True, True, nil );
    finally
      DoEnableServer;
    end;
  finally
    FBusy := False;
  end;
end;
```

3.2.10 uUtils.pas

This file wraps the functions exported from the core DLL of the server. They are basic functions that allow you to manipulate certain information stored in the repository from within your triggers. These functions should be used with care and should always be called in the context of the calling thread:

```
// User/Group functions
function GetUserInfo( UID: Cardinal; var Name, FullName, EMail, Location, Extra: String
): Boolean;
function GetUserCount( GroupID: Cardinal ): Integer;
function GetUserInfoByIndex( GroupID: Cardinal; Index: Integer; var UID: Cardinal; var
Name, FullName, EMail, Location, Extra: String ): Boolean;
function IsSuperuser( UID: Cardinal ): Boolean;
function IsAdminUser( UID: Cardinal ): Boolean;
function GetGroupInfo( GID: Cardinal; var Name, Description, Extra: String ): Boolean;
function GetGroupCount( GroupID: Cardinal ): Integer;
function GetGroupInfoByIndex( GroupID: Cardinal; Index: Integer; var GID: Cardinal; var
Name, Description, Extra: String ): Boolean;

// Security related functions
// Access Level Rights
function GetCanAccessIDs( ObjectID: Cardinal; var IDs: TIDArray ): Boolean;
function GetCanAccessNames( ObjectID: Cardinal; const Names: TStrings ): Boolean;
function SetCanAccessIDs( ObjectID: Cardinal; IDs: TIDArray ): Boolean;
function SetCanAccessNames( ObjectID: Cardinal; Names: TStrings ): Boolean;
// Modify Level Rights
function GetCanModifyIDs( ObjectID: Cardinal; var IDs: TIDArray ): Boolean;
function GetCanModifyNames( ObjectID: Cardinal; const Names: TStrings ): Boolean;
function SetCanModifyIDs( ObjectID: Cardinal; IDs: TIDArray ): Boolean;
function SetCanModifyNames( ObjectID: Cardinal; Names: TStrings ): Boolean;
// CheckInOut Level Rights
function GetCanCheckInOutIDs( ObjectID: Cardinal; var IDs: TIDArray ): Boolean;
function GetCanCheckInOutNames( ObjectID: Cardinal; const Names: TStrings ): Boolean;
function SetCanCheckInOutIDs( ObjectID: Cardinal; IDs: TIDArray ): Boolean;
function SetCanCheckInOutNames( ObjectID: Cardinal; Names: TStrings ): Boolean;
// View Level Rights
function GetCanViewIDs( ObjectID: Cardinal; var IDs: TIDArray ): Boolean;
function GetCanViewNames( ObjectID: Cardinal; const Names: TStrings ): Boolean;
function SetCanViewIDs( ObjectID: Cardinal; IDs: TIDArray ): Boolean;
function SetCanViewNames( ObjectID: Cardinal; Names: TStrings ): Boolean;

// Project, Folder, File functions
function GetFileInfo( FileID: Cardinal; var Name, LockedBy, Extra: String; var TimeStamp,
FileDate: Integer; var Virtual, Frozen, Removed: Boolean ): Boolean;
procedure EnumerateProjects( EnumProc: TEnumProjects; pData: Pointer );
procedure EnumerateFolders( RootID: Cardinal; EnumProc: TEnumFolders; pData: Pointer;
Recursive: Boolean );
procedure EnumerateFiles( RootID: Cardinal; EnumProc: TEnumFiles; pData: Pointer;
Recursive: Boolean );

// General
procedure ReleaseUtils;
```

These functions are documented in the file **uUtils.pas**.

3.2.11 VcsAfterAssignVersion

If this method is exported from the DLL, it will be called after a Version Label has been assigned to an archive.

VcsAfterAssignVersion

```
procedure VcsAfterAssignVersion( pProject, pFolder, pObjectName, pUserName, pRevision,
pVersionLabel: PChar; UserID, FileID, RevisionID, VersionID: Cardinal ); stdcall; export;
```

This procedure is called after an Assign Version action occurs. This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
pProject	The name of the Project containing the file
pFolder	The name of the Folder containing the file
pObjectName	The name of the file object that the Version Label was assigned to
pUserName	The name of the user that carried out the action
pRevision	The name of the revision the Version Label was assigned to
pVersionLabel	The name of the Version Label
UserID	The ID of the user that carried out the action
FileID	The ID of the file
RevisionID	The ID of the revision the Version Label was assigned to
VersionID	The ID of the Version Label being assigned

Example (Delphi)

The following method is called after a Version Label is successfully assigned to a file and simply logs the action to a local text file:

```
procedure VcsAfterAssignVersion( pProject, pFolder, pObjectName, pUserName, pRevision,
pVersionLabel: PChar; UserID, FileID, RevisionID, VersionID: Cardinal );
var
  Name, FullName, EMail, Location, Extra: String;
begin
  // This method is called after a Version Label is successfully assigned to a file
  GetUserInfo( UserID, Name, FullName, EMail, Location, Extra );
  LogIt( Format( '%s - ''%s (%s)'' Assigned Version ''%s'' to revision ''%s'' of ''%s''',
[ DateTimeToStr( Now ), String( pUserName ), EMail, String( pVersionLabel ), String(
pRevision ), String( pObjectName ) ] ) );
end;
```

3.2.12 VcsAfterCheckIn

If this method is exported from the DLL, it will be called after a Check In occurs:

VcsAfterCheckIn

```
procedure VcsAfterCheckIn( pProject, pFolder, pObjectName, pUserName, pComments,
pNewRevision: PChar;
                           Branched: Boolean; TimeStamp, FileDate, Flags: Integer;
                           UserID, FileID: Cardinal ); stdcall; export;
```

This procedure is called after a CheckIn action occurs. This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
pProject	The name of the Project containing the file
pFolder	The name of the Folder containing the file
pObjectName	The name of the file object that was checked in
pUserName	The name of the user that carried out the action
pComments	The comments assigned to the new revision
pNewRevision	The name of the newly created revision
Branched	Indicates whether the checkin caused a branch revision to be created
Timestamp	The date/time that the file was checked in
FileDate	The date/time that the local file was last modified
Flags	The flags used for the checkin action. See TCVcsConst.pas
UserID	The ID of the user that carried out the action.
FileID	The ID of the file that was checked in

Example (Delphi)

The following method is called after checkin of a file and simply logs the action to a local text file:

```
procedure VcsAfterCheckIn( pProject, pFolder, pObjectName, pUserName, pComments,
pNewRevision: PChar;
                           Branched: Boolean; TimeStamp, FileDate, Flags: Integer;
                           UserID, FileID: Cardinal );
var
  Name, FullName, EMail, Location, Extra: String;
begin
  // This method is called after a successful Check In.
  DoGetUserInfo( UserID, Name, FullName, EMail, Location, Extra );
  LogIt( Format( '%s - ''%s (%s)'' Checked In file ''%s'' - Comments: %s',
                 [ DateTimeToStr( FileDateToDateTime( TimeStamp ) ),
                 String( pUserName ), EMail, String( pObjectName ),
                 String( pComments ) ] ) );
end;
```

3.2.13 VcsAfterCheckOut

If this method is exported from the DLL, it will be called after a Check Out occurs:

VcsAfterCheckOut

```
procedure VcsAfterCheckOut( pProject, pFolder, pObjectName, pUserName, pLockComments,
pRevision: PChar;
                            TimeStamp, Flags: Integer; UserID, FileID: Cardinal );
stdcall; export;
```

This procedure is called after a CheckOut action occurs. This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
pProject	The name of the Project containing the file
pFolder	The name of the Folder containing the file
pObjectName	The name of the file object that was Checked Out
pUserName	The name of the user that carried out the action
pLockComments	The comments assigned to the lock
pRevision	The name of the revision being checked out
Timestamp	The date/time of the revision being checked out
Flags	The flags used for the checkin action. See TCVcsConst.pas
UserID	The ID of the user that carried out the action.
FileID	The ID of the file that was checked out

Example (Delphi)

The following method is called after checkout of a file and simply logs the action to a local text file:

```
procedure VcsAfterCheckOut( pProject, pFolder, pObjectName, pUserName, pLockComments,
pRevision: PChar;
                            TimeStamp, Flags: Integer; UserID, FileID: Cardinal );
begin
  // This method is called after a successful Check Out.
  LogIt( Format( '%s - ''%s'' Checked Out file ''%s'' - Lock Comments: %s',
         [ DateTimeToStr( FileDateToDateTime( TimeStamp ) ), String( pUserName ),
         String( pObjectName ), String( pLockComments ) ] ) );
end;
```

3.2.14 VcsAfterCreateFolder

If this method is exported from the DLL, it will be called after a new Folder has been created.

VcsAfterCreateFolder

```
procedure VcsAfterCreateFolder( pProject, pFolder, pFolderPath, pUserName: PChar; UserID,
ProjectID, FolderID: Cardinal ); stdcall; export;
```

This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
pProject	The name of the Project that the Folder was created in
pFolder	The name of the newly created Folder
pFolderPath	The Default working path for this folder
pUserName	The name of the user that carried out the action
UserID	The ID of the user that carried out the action
ProjectID	The ID of the Project that the Folder was created in
FolderID	The ID of the newly created Folder

Example (Delphi)

The following method is called after a Project is successfully created and simply logs the action to a local text file:

```
procedure VcsAfterCreateFolder( pProject, pFolder, pFolderPath, pUserName: PChar; UserID,
ProjectID, FolderID: Cardinal );
begin
  // This method is called after a Folder is created
  LogIt( Format( '%s - ''%s'' Created Folder ''%s'', mapping to: %s', [ DateTimeToStr(
Now ), String( pUserName ), String( pFolder ), String( pFolderPath ) ] ) );
end;
```

3.2.15 VcsAfterCreateProject

If this method is exported from the DLL, it will be called after a new Project has been created.

VcsAfterCreateProject

```
procedure VcsAfterCreateProject( pProject, pUserName: PChar; UserID, ProjectID: Cardinal
); stdcall; export;
```

This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
pProject	The name of the newly created Project
pUserName	The name of the user that carried out the action
UserID	The ID of the user that carried out the action
ProjectID	The ID assigned to the new Project

Example (Delphi)

The following method is called after a Project is successfully created and simply logs the action to a local text file:

```
procedure VcsAfterCreateProject( pProject, pUserName: PChar; UserID, ProjectID: Cardinal
);
begin
  // This method is called after a Project is created
  LogIt( Format( '%s - ''%s'' Created Project ''%s''', [ DateTimeToStr( Now ), String(
pUserName ), String( pProject ) ] ) );
end;
```

3.2.16 VcsAfterDeleteObject

If this method is exported from the DLL, it will be called after an object is deleted from the repository. This method is called only when the object is permanently deleted and not when it is added to the recycle bin.

VcsAfterDeleteObject

```
procedure VcsAfterDeleteObject( pObjectName, pObjectParent, pUserName: PChar; UserID,
ObjectID, ObjectParentID: Cardinal; ObjectType: Integer ); stdcall; export;
```

This is a notification event only and none of the values passed in can be modified. See below for a list of Object Types and their ID's

Parameters

Name	Description
pObjectName	The name of the object that has been deleted
pObjectParent	If the object that has been deleted is a Folder, File, or Revision, this references the name of the parent object.
pUserName	The name of the user that carried out the action
UserID	The ID of the user that carried out the action
ObjectID	The ID of the object that was deleted
ObjectParentID	If the object that has been deleted is a Folder, File, or Revision, this references the ID of the parent object.
ObjectType	The type of object that was deleted. See below

Example (Delphi)

The following method is called after a Project is successfully created and simply logs the action to a local text file:

```
const
  // Object types
  ot_Project        = 1;
  ot_Folder         = 2;
  ot_File           = 3;
  ot_Revision       = 4;
  ot_VersionLabel   = 5;
  ot_PromotionLevel = 6;

procedure VcsAfterDeleteObject( pObjectName, pObjectParent, pUserName: PChar; UserID,
ObjectID, ObjectParentID: Cardinal; ObjectType: Integer );
begin
  // This method is called after an object has been permanently deleted from the
repository
  // See the constants defined above.
  case ObjectType of
    ot_Project: LogIt( Format( '%s - ''%s'' Deleted Project ''%s''', [ DateTimeToStr( Now
), String( pUserName ), String( pObjectName ) ] ) );
    ot_Folder: LogIt( Format( '%s - ''%s'' Deleted Folder ''%s'' from Project ''%s''', [
DateTimeToStr( Now ), String( pUserName ), String( pObjectName ), String( pObjectParent )
] ) );
    ot_File: LogIt( Format( '%s - ''%s'' Deleted File ''%s'' from Folder ''%s''', [
DateTimeToStr( Now ), String( pUserName ), String( pObjectName ), String( pObjectParent )
] ) );
    ot_Revision: LogIt( Format( '%s - ''%s'' Deleted Revision ''%s'' from File ''%s''', [
DateTimeToStr( Now ), String( pUserName ), String( pObjectName ), String( pObjectParent )
] ) );
    ot_VersionLabel: LogIt( Format( '%s - ''%s'' Deleted Version Label ''%s''', [
DateTimeToStr( Now ), String( pUserName ), String( pObjectName ) ] ) );
    ot_PromotionLevel: LogIt( Format( '%s - ''%s'' Deleted Promotion Level ''%s''', [
DateTimeToStr( Now ), String( pUserName ), String( pObjectName ) ] ) );
  end;
end;
```

3.2.17 VcsAfterFreeze

If this method is exported from the DLL, it will be called after an archive has been frozen. Freezing an archive prevents further development of the file.

VcsAfterFreeze

```
procedure VcsAfterFreeze( pProject, pFolder, pObjectName, pUserName: PChar; UserID,
FileID: Cardinal ); stdcall; export; stdcall; export;
```

This procedure is called after an Remove Version Label action occurs. This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
pProject	The name of the Project containing the file
pFolder	The name of the Folder containing the file
pObjectName	The name of the file object that was frozen
pUserName	The name of the user that carried out the action
UserID	The ID of the user that carried out the action
FileID	The ID of the file

Example (Delphi)

The following method is called after a file has been frozen and simply logs the action to a local text file:

```
procedure VcsAfterFreeze( pProject, pFolder, pObjectName, pUserName: PChar; UserID,
FileID: Cardinal );
var
  Name, FullName, EMail, Location, Extra: String;
begin
  // This method is called after a file is successfully frozen
  GetUserInfo( UserID, Name, FullName, EMail, Location, Extra );
  LogIt( Format( '%s - ''%s (%s)'' Froze File ''%s''', [ DateTimeToStr( Now ), String(
pUserName ), EMail, String( pObjectName ) ] ) );
end;
```

3.2.18 VcsAfterGet

If this method is exported from the DLL, it will be called after a Get action occurs:

VcsAfterGet

```
procedure VcsAfterGet( pProject, pFolder, pObjectName, pUserName, pRevision: PChar;
                       TimeStamp, Flags: Integer; UserID, FileID: Cardinal ); stdcall;
export;
```

This procedure is called after a Get action occurs. This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
pProject	The name of the Project containing the file
pFolder	The name of the Folder containing the file
pObjectName	The name of the file object
pUserName	The name of the user that carried out the action
pRevision	The name of the revision being checked out
Timestamp	The date/time of the revision
Flags	The flags used for the get action. See TCVcsConst.pas
UserID	The ID of the user that carried out the action.
FileID	The ID of the file

Example (Delphi)

The following method is called after a get of a file and simply logs the action to a local text file:

```
procedure VcsAfterGet( pProject, pFolder, pObjectName, pUserName, pRevision: PChar;
                       TimeStamp, Flags: Integer; UserID, FileID: Cardinal );
begin
  // This method is called after a successful Get.
  LogIt( Format( '%s - ''%s'' Got revision ''%s'' of ''%s''',
                 [ DateTimeToStr( FileDateToDateTime( TimeStamp ) ),
                 String( pUserName ), String( pRevision ),
                 String( pObjectName ) ] ) );
end;
```

3.2.19 VcsAfterLock

If this method is exported from the DLL, it will be called after a Lock occurs. Note that this event occurs as a result of the user using the Lock commands and is ***not*** triggered after a CheckOut action.

VcsAfterLock

```
procedure VcsAfterLock( pProject, pFolder, pObjectName, pUserName, pLockComments,
pRevision: PChar;
                        TimeStamp: Integer; UserID, FileID: Cardinal ); stdcall; export;
```

This procedure is called after a Lock action occurs. This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
pProject	The name of the Project containing the file
pFolder	The name of the Folder containing the file
pObjectName	The name of the file object that was Locked
pUserName	The name of the user that carried out the action
pLockComments	The comments assigned to the lock
pRevision	The name of the revision being locked
Timestamp	The date/time of the revision being locked
UserID	The ID of the user that carried out the action.
FileID	The ID of the file that was locked

Example (Delphi)

The following method is called after a file is locked and simply logs the action to a local text file:

```
procedure VcsAfterLock( pProject, pFolder, pObjectName, pUserName, pLockComments,
pRevision: PChar;
                        TimeStamp: Integer; UserID, FileID: Cardinal );
begin
  // This method is called after a successful Lock. This does not include files locked
during Check Out.
  LogIt( Format( '%s - ''%s'' Locked revision ''%s'' of ''%s''',
                [ DateTimeToStr( FileDateToDateTime( TimeStamp ) ),
                String( pUserName ), String( pRevision ),
                String( pObjectName ) ] ) );
end;
```

3.2.20 VcsAfterLogin

If this method is exported from the DLL, it will be called after a user has successfully logged in.

VcsAfterLogin

```
procedure VcsAfterLogin( UserID: Cardinal; pName, pComputer: PChar ); stdcall; export;
```

This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
UserID	The ID of the user
pName	The Name of the user
pComputer	The name of the machine that the user logged in from

Example (Delphi)

The following method is called after a User has successfully logged in and simply logs the action to a local text file:

```
procedure VcsAfterLogin( UserID: Cardinal; pName, pComputer: PChar );
begin
  LogIt( Format( '%s - ''%s'' Logged IN from Computer ''%s''', [ DateTimeToStr( Now ),
```

```
String( pName ), String( pComputer ) ] ) );
end;
```

3.2.21 VcsAfterLogout

If this method is exported from the DLL, it will be called after a user has successfully logged in.

VcsAfterLogout

```
procedure VcsAfterLogout( UserID: Cardinal; pName, pComputer: PChar ); stdcall; export;
```

This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
UserID	The ID of the user
pName	The Name of the user
pComputer	The name of the machine that the user originally logged in from

Example (Delphi)

The following method is called after a User has successfully logged out and simply logs the action to a local text file:

```
procedure VcsAfterLogout( UserID: Cardinal; pName, pComputer: PChar );
begin
  LogIt( Format( '%s - ''%s'' Logged OUT from Computer ''%s''', [ DateTimeToStr( Now ),
String( pName ), String( pComputer ) ] ) );
end;
```

3.2.22 VcsAfterPromote

If this method is exported from the DLL, it will be called after a file is promoted.

VcsAfterPromote

```
procedure VcsAfterPromote( pProject, pFolder, pObjectName, pUserName, pRevision,
pPromoteTo: PChar;
                           UserID, FileID, PromotionID: Cardinal ); stdcall; export;
```

This procedure is called after a Promote action occurs. This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
pProject	The name of the Project containing the file
pFolder	The name of the Folder containing the file
pObjectName	The name of the file object that was Promoted
pUserName	The name of the user that carried out the action
pRevision	The name of the revision that was promoted
pPromoteTo	The Promotion Label that the file was promoted to
UserID	The ID of the user that carried out the action.
FileID	The ID of the file that was promoted
PromotionID	The ID of the Promotion Label that the file was promoted to

Example (Delphi)

The following method is called after a file is promoted and simply logs the action to a local text file:

```
procedure VcsAfterPromote( pProject, pFolder, pObjectName, pUserName, pRevision,
pPromoteTo: PChar;
                           UserID, FileID, PromotionID: Cardinal );
var
  Name, FullName, EMail, Location, Extra: String;
begin
  // This method is called after a file is successfully promoted.
  GetUserInfo( UserID, Name, FullName, EMail, Location, Extra );
  LogIt( Format( '%s - ''%s (%s)'' Promoted revision %s of file ''%s'' to ''%s''',
                 [ DateTimeToStr( Now ), String( pUserName ), EMail,
                 String( pRevision ), String( pObjectName ),
                 String( pPromoteTo ) ] ) );
end;
```

3.2.23 VcsAfterRemoveVersion

If this method is exported from the DLL, it will be called after a Version Label has been removed from an archive.

VcsAfterRemoveVersion

```
procedure VcsAfterRemoveVersion( pProject, pFolder, pObjectName, pUserName, pRevision,
pVersionLabel: PChar; UserID, FileID, RevisionID, VersionID: Cardinal ); stdcall; export;
```

This procedure is called after an Remove Version Label action occurs. This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
pProject	The name of the Project containing the file
pFolder	The name of the Folder containing the file
pObjectName	The name of the file object that the Version Label removed from
pUserName	The name of the user that carried out the action
pRevision	The name of the revision the Version Label was removed from
pVersionLabel	The name of the Version Label being removed
UserID	The ID of the user that carried out the action
FileID	The ID of the file
RevisionID	The ID of the revision the Version Label was assigned to
VersionID	The ID of the Version Label being assigned

Example (Delphi)

The following method is called after a Version Label is successfully removed from a file and simply logs the action to a local text file:

```
procedure VcsAfterRemoveVersion( pProject, pFolder, pObjectName, pUserName, pRevision,
pVersionLabel: PChar; UserID, FileID, RevisionID, VersionID: Cardinal );
var
  Name, FullName, EMail, Location, Extra: String;
begin
  // This method is called after a Version Label is successfully removed
  GetUserInfo( UserID, Name, FullName, EMail, Location, Extra );
  LogIt( Format( '%s - ''%s (%s)'' Removed Version ''%s'' from revision ''%s'' of
''%s''', [ DateTimeToStr( Now ), String( pUserName ), EMail, String( pVersionLabel ),
String( pRevision ), String( pObjectName ) ] ) );
end;
```

3.2.24 VcsAfterUnlock

If this method is exported from the DLL, it will be called after an Unlock occurs. Note that this event occurs as a result of the user using the Unlock commands and is ***not*** triggered after a CheckIn action.

VcsAfterUnlock

```
procedure VcsAfterUnlock( pProject, pFolder, pObjectName, pUserName, pRevision: PChar;
                          TimeStamp: Integer; UserID, FileID: Cardinal ); stdcall;
export;
```

This procedure is called after an Unlock action occurs. This is a notification event only and none of the values passed in can be modified.

Parameters

Name	Description
pProject	The name of the Project containing the file
pFolder	The name of the Folder containing the file
pObjectName	The name of the file object that was Unlocked
pUserName	The name of the user that carried out the action
pRevision	The name of the revision being unlocked
Timestamp	The date/time of the revision being unlocked
UserID	The ID of the user that carried out the action.
FileID	The ID of the file that was unlocked

Example (Delphi)

The following method is called after a file is unlocked and simply logs the action to a local text file:

```
procedure VcsAfterUnlock( pProject, pFolder, pObjectName, pUserName, pRevision: PChar;
                          TimeStamp: Integer; UserID, FileID: Cardinal );
begin
  // This method is called after a successful Lock. This does not include files unlocked
during Check In.
  LogIt( Format( '%s - ''%s'' Unlocked revision ''%s'' of ''%s''',
                 [ DateTimeToStr( FileDateToDateTime( TimeStamp ) ),
                 String( pUserName ), String( pRevision ),
                 String( pObjectName ) ] ) );
end;
```

3.2.25 VcsBeginAction

To allow grouping of multiple actions, this method will be called (if present) before a group of actions occur.

```
procedure VcsBeginAction( UserID: Cardinal; ActionRef: Integer; Action: Word ); stdcall;
export;
```

This method could be used to, for example, allocate resources that will be used when a set of actions will occur. VcsEndAction will always be called after a group of actions have occurred. Note that all actions, for a particular ActionRef, will be of the same type.

Parameters

Name	Description
UserID	The ID of the User carrying out the actions
ActionRef	A unique value that identifies the group of actions
Action	The type of action about to start. See TCVcsConst.pas for a list of the action types.

Example (Delphi)

```
var
  PromoteLog: TStrings = nil;

procedure VcsBeginAction( UserID: Cardinal; ActionRef: Integer; Action: Word );
begin
  // This procedure is called just before an action is carried out on multiple
  // files. ActionRef is a unique number passed into the procedure and allows
  // you to group multiple calls to VcsBefore... and VcsAfter... events.
```

```
  // See the file TCVcsConst for the act_XXXX values passed in the Action parameter
  if ( Action = act_Promote ) and ( not Assigned( PromoteLog ) ) then
    PromoteLog := TStringList.Create;
end;
```

3.2.26 VcsConfigureAddin

If your addin needs to be configured to work properly (i.e. an addin to E-Mail users), you should export this procedure. If you export this procedure, the **Configure...** button on the Addin dialog box will be enabled when your addin is selected.

```
procedure VcsConfigureAddin; stdcall; export;
```

You can display a dialog box to the user to allow them to configure your addin. Note that you are responsible for storage and retrieval of the configuration information.

This is the only point that your addin can request feedback from the user. This function is called from the repository configuration dialog at a time when the repository service is not running.

Parameters

None

Example (Delphi)

```
procedure VcsConfigureAddin;
begin
  with TMyConfigDlg.Create( Application ) do
  try
    ShowModal;
  finally
    Free;
  end;
end;
```

3.2.27 VcsEndAction

To allow grouping of multiple actions, this method will be called (if present) after a group of actions occur.

```
procedure VcsEndAction( UserID: Cardinal; ActionRef: Integer; Action: Word ); stdcall;
export;
```

This method should be used to, for example, released resources allocated during a call to VcsBeginAction.

Parameters

Name	Description
UserID	The ID of the user performing the actions
ActionRef	A unique value that identifies the group of actions
Action	The type of action just completed. See TCVcsConst.pas for a list of the action types.

Example (Delphi)

```
procedure VcsEndAction( UserID: Cardinal; ActionRef: Integer; Action: Word );
begin
```

```
  // This procedure is called just after an action is carried out on multiple
  // files.
  if ( Action = act_Promote ) and ( Assigned( PromoteLog ) ) then
    PromoteLog.Free;
  PromoteLog := nil;
end;
```

3.2.28 VcsInitAddin

This, required, function should be exported from all server-side addins. It is used to initialize the addin and is called when the Addin is first loaded.

```
procedure VcsInitAddin( const pName: PChar; pPath: PChar;
                        DisableServerProc: TDisableServer;
                        EnableServerProc: TEnableServer;
                        FlushBuffersProc: TFlushBuffers;
                        ReinitializeProc: TReinitialize;
                        ValidateProc: TValidateRepository;
                        FileEnumeratorProc: TEnumerateFiles;
                        GetUserProc: TGetUserInfo;
                        GetFileProc: TGetFileInfo;
                        Offline: Boolean ); stdcall; export;
```

If you allocate memory or resources in the Addin, you can use the VcsReleaseAddin function to release it.

Parameters

Name	Description
pName (out)	Return a descriptive name for this addin in this parameter.
pPath	This is the full path to the server repository.
DisableServerProc	A pointer to a method of type TDisableServer that can be called from within the addin.
EnableServerProc	A pointer to a method of type TEnableServer that can be called from within the addin.
FlushBuffersProc	A pointer to a method of type TFlushBuffers that can be called from within the addin.
ReinitializeProc	A pointer to a method of type TReinitialize that can be called from within the addin.
ValidateProc	A pointer to a method of type TValidateRepository that can be called from within the addin.
FileEnumeratorProc	A pointer to a method of type TEnumerateFiles that can be called from within the addin.
GetUserProc	A pointer to a method of type TGetUserInfo that can be called from within the addin.
GetFilesProc	A pointer to a method of type TGetFileInfo that can be called from within the addin.
Offline	Indicates whether the server is currently offline.

Example (Delphi)

```
type
  // Callback for Validation / File Enumeration
  TFeedbackProc = procedure( pText: PChar ); stdcall;
  TEnumerateFilesProc = procedure( pFile: PChar ); stdcall;
  // Interface to GPVSCore.dll
  TDisableServer = procedure; stdcall;
  TEnableServer = procedure; stdcall;
  TFlushBuffers = procedure; stdcall;
```

```
  TReinitialize = function: Boolean; stdcall;
  TValidateRepository = procedure ( pPath, pFile: PChar; ValidateProjects: Boolean;
ValidateArchives: Boolean; fbProc: TFeedbackProc ); stdcall;
  TEnumerateFiles = procedure ( pPath: PChar; EnumerateProjects, EnumerateArchives:
Boolean; EnumerateProc: TEnumerateFilesProc ); stdcall;
  TGetUserInfo = function ( UID: Cardinal; const pName, pFullName, pEMail, pLocation,
pExtra: PChar ): Integer; stdcall;
  TGetFileInfo = function ( FileID: Cardinal; const pName, pLockedBy, pExtra: PChar; var
TimeStamp, FileDate: Integer; var Virtual, Frozen, Removed: Boolean ): Integer; stdcall;

var
  DoDisableServer: TDisableServer = nil;
  DoEnableServer: TEnableServer = nil;
  DoFlushBuffers: TFlushBuffers = nil;
  DoReinitialize: TReinitialize = nil;
  DoValidateRepository: TValidateRepository = nil;
  DoEnumerateFiles: TEnumerateFiles = nil;
  DoGetUserInfo: TGetUserInfo = nil;
  DoGetFileInfo: TGetFileInfo = nil;
  RepositoryPath: String = '';

procedure VcsInitAddin( const pName: PChar; pPath: PChar;
                        DisableServerProc: TDisableServer;
                        EnableServerProc: TEnableServer;
                        FlushBuffersProc: TFlushBuffers;
                        ReinitializeProc: TReinitialize;
                        ValidateProc: TValidateRepository;
                        FileEnumeratorProc: TEnumerateFiles;
                        GetUserProc: TGetUserInfo;
                        GetFileProc: TGetFileInfo;
                        Offline: Boolean );
begin
  // This procedure is called when the addin is first loaded
  // Use it to initialize internal variables.
  StrCopy( pName, 'Sample Addin' );
  // Store the passed function pointers for later use...
  DoDisableServer := DisableServerProc;
  DoEnableServer := EnableServerProc;
  DoFlushBuffers := FlushBuffersProc;
  DoReinitialize := ReinitializeProc;
  DoValidateRepository := ReinitializeProc;
  DoEnumerateFiles := FileEnumeratorProc;
  DoGetUserInfo := GetUserProc;
  DoGetFileInfo := GetFileProc;
  // Remember the location of the repository
  SetLength( RepositoryPath, StrLen( pPath ) );
  RepositoryPath := String( pPath );
  if Copy( RepositoryPath, Length( RepositoryPath ), 1 ) <> '\' then
    RepositoryPath := RepositoryPath + '\';
end;
```

3.2.29 VcsReleaseAddin

This, required, function should be exported from all server side addins. It is called just before the Addin is unloaded and should be used to release any resources allocated while the addin was loaded..

```
procedure VcsReleaseAddin; stdcall; export;
```

Parameters

None

Example (Delphi)

```
procedure VcsReleaseAddin;
begin
  // This procedure is called just before the addin is unloaded.
  // Use it to release memory allocated while the addin is active
end;
```

Index

- C -

Client side Addins 76
Check In 76
Get 76
Lock 78
Unlock 78
VcsAfterCheckIn 74
VcsAfterCheckOut 76
VcsAfterLock 78
VcsAfterPromote 79
VcsBeforeCheckIn 74
VcsBeforeCheckOut 76
VcsBeforeLock 78
VcsBeforePromote 79
VcsBeginAction 81
VcsConfigureAddin 81
VcsEndAction 82
VcsInitEventAddin 82
VcsReleaseAddin 83
VcsShowAbout 83
VcsToLocalFormat 84
VcsToServerFormat 84
Constants 10

- D -

Direct Interface 34
TCDVcsAddFolder 36
TCDVcsAddProject 36
TCDVcsAddVersionLabel 37
TCDVcsAttachVersionLabel 37
TCDVcsCheckInFile 38
TCDVcsCheckOutFile 39
TCDVcsCheckOutFileEx 39
TCDVcsConnect 40
TCDVcsCurrentConnection 41
TCDVcsDeleteFile 42
TCDVcsDeleteFolder 42
TCDVcsDeleteProject 43
TCDVcsDeleteVersionLabel 43
TCDVcsDetachVersionLabel 44
TCDVcsDisconnect 44
TCDVcsEnumConnections 45
TCDVcsEnumFiles 45
TCDVcsEnumFolders 47
TCDVcsEnumLabels 48
TCDVcsEnumLockedFiles 49
TCDVcsEnumProjects 50
TCDVcsEnumRevisions 51
TCDVcsEnumViews 52
TCDVcsErrorString 53
TCDVcsFileStatus 54
TCDVcsGetFileCheckoutPath 54
TCDVcsGetFileGroupExt 55
TCDVcsGetFileID 56
TCDVcsGetFileWorkingPath 56
TCDVcsGetObjectNotes 57
TCDVcsModifyVersionLabel 58
TCDVcsMoveFile 58
TCDVcsNextPromotionLabel 59
TCDVcsPromote 59
TCDVcsRemoveObject 60
TCDVcsRenameFolder 60
TCDVcsRenameProject 61
TCDVcsSetObjectNotes 61
TCDVcsSetView 62
TCDVcsShareFile 62
TCDVcsUncheckOutFile 63
TCDVcsUnshareFile 63

- I -

InitializeCheckInInfo 69
InitializeCheckOutInfo 69

- M -

Main Interface 12
TCVcsAbout 12
TCVcsBrowseFolder 12
TCVcsCheckInFile 13
TCVcsCheckInFiles 13
TCVcsCheckInFilesEx 14
TCVcsCheckOutFile 15
TCVcsCheckOutFiles 15
TCVcsCheckOutFilesEx 16
TCVcsCreateProject 16
TCVcsCurrentConnection 17
TCVcsCurrentUserName 17
TCVcsErrorString 17
TCVcsFileMainExtension 18
TCVcsGetCheckInInfo 18
TCVcsGetCheckOutInfo 19
TCVcsGetFileID 19
TCVcsGetFileList 20
TCVcsGetFileListForFolders 20

Main Interface 12
TCVcsGetFileStatus 21
TCVcsGetObjectType 22
TCVcsGetObjectVersions 22
TCVcsGetViewBaseID 23
TCVcsGetViews 23
TCVcsHaveLock 24
TCVcsInitialize 24
TCVcsIsFileArchived 25
TCVcsLogin 25
TCVcsProjectIDByName 26
TCVcsRefreshCache 26
TCVcsRelease 27
TCVcsRemoveFiles 27
TCVcsRenameFile 27
TCVcsSelectConnection 28
TCVcsSelectProject 28
TCVcsSelectView 29
TCVcsShowArchiveManager 29
TCVcsShowConnectionInfo 30
TCVcsShowHistory 30
TCVcsShowProperties 31
TCVcsShowWorkfileDifferences 32
TCVcsUndoCheckOutFile 32
TCVcsUndoCheckOutFiles 33

- R -

ReleaseCheckInInfo 70
ReleaseCheckOutInfo 70

- S -

Server side Addins 88
TDisableServer 88
TEnableServer 89
TEnumerateFiles 89
TFlushBuffers 90
TGetFileInfo 90
TGetUserInfo 91
TReinitialize 92
TValidateRepository 93
VcsAfterAssignVersion 94
VcsAfterCheckIn 95
VcsAfterCheckOut 96
VcsAfterCreateFolder 97
VcsAfterCreateProject 98
VcsAfterDeleteObject 98
VcsAfterFreeze 99
VcsAfterGet 100
VcsAfterLock 101
VcsAfterLogin 102
VcsAfterLogout 103
VcsAfterPromote 103
VcsAfterRemoveVersion 104
VcsAfterThaw 92
VcsAfterUnlock 105
VcsBeginAction 106
VcsConfigureAddin 107
VcsEndAction 107
VcsInitAddin 108
VcsReleaseAddin 109

- T -

TCDVcsAddFolder 36
TCDVcsAddProject 36
TCDVcsAddVersionLabel 37
TCDVcsAttachVersionLabel 37
TCDVcsCheckInFile 38
TCDVcsCheckOutFile 39
TCDVcsCheckOutFileEx 39
TCDVcsConnect 40
TCDVcsCurrentConnection 41
TCDVcsDeleteFile 42
TCDVcsDeleteFolder 42
TCDVcsDeleteProject 43
TCDVcsDeleteVersionLabel 43
TCDVcsDetachVersionLabel 44
TCDVcsDisconnect 44
TCDVcsEnumConnections 45
TCDVcsEnumFiles 45
TCDVcsEnumFolders 47
TCDVcsEnumLabels 48
TCDVcsEnumLockedFiles 49
TCDVcsEnumProjects 50
TCDVcsEnumRevisions 51
TCDVcsEnumViews 52
TCDVcsErrorString 53
TCDVcsFileStatus 54
TCDVcsGetFileCheckoutPath 54
TCDVcsGetFileGroupExt 55
TCDVcsGetFileID 56
TCDVcsGetFileWorkingPath 56
TCDVcsGetObjectNotes 57
TCDVcsModifyVersionLabel 58
TCDVcsMoveFile 58
TCDVcsNextPromotionLabel 59
TCDVcsPromote 59
TCDVcsRemoveObject 60
TCDVcsRenameFolder 60

TCDVcsRenameProject 61
TCDVcsSetObjectNotes 61
TCDVcsSetView 62
TCDVcsShareFile 62
TCDVcsUncheckOutFile 63
TCDVcsUnshareFile 63
TCVcsAbout 12
TCVcsBrowseFolder 12
TCVcsCheckInFile 13
TCVcsCheckInFiles 13
TCVcsCheckInFilesEx 14
TCVcsCheckOutFile 15
TCVcsCheckOutFiles 15
TCVcsCheckOutFilesEx 16
TCVcsCreateProject 16
TCVcsCurrentConnection 17
TCVcsCurrentUserName 17
TCVcsErrorString 17
TCVcsFileMainExtension 18
TCVcsGetCheckInInfo 18
TCVcsGetCheckOutInfo 19
TCVcsGetFileID 19
TCVcsGetFileList 20
TCVcsGetFileListForFolders 20
TCVcsGetFileStatus 21
TCVcsGetObjectType 22
TCVcsGetObjectVersions 22
TCVcsGetViewBaseID 23
TCVcsGetViews 23
TCVcsHaveLock 24
TCVcsInitialize 24
TCVcsIsFileArchived 25
TCVcsLogin 25
TCVcsProjectIDByName 26
TCVcsRefreshCache 26
TCVcsRelease 27
TCVcsRemoveFiles 27
TCVcsRenameFile 27
TCVcsSelectConnection 28
TCVcsSelectProject 28
TCVcsSelectView 29
TCVcsShowArchiveManager 29
TCVcsShowConnectionInfo 30
TCVcsShowHistory 30
TCVcsShowProperties 31
TCVcsShowWorkfileDifferences 32
TCVcsUndoCheckOutFile 32
TCVcsUndoCheckOutFiles 33
TDisableServer (Server) 88
TEnableServer (Server) 89
TEnumerateFiles (Server) 89
TFlushBuffers (Server) 90
TGetFileInfo (Server) 90
TGetUserInfo (Server) 91
TReinitialize (Server) 92
TValidateRepository (Server) 93
Type Definitions 68

- U -

Utilities 69

- V -

VcsAfterAssignVersion (Server) 94
VcsAfterCheckIn (Client) 74
VcsAfterCheckIn (Server) 95
VcsAfterCheckOut (Client) 76
VcsAfterCheckOut (Server) 96
VcsAfterCreateFolder (Server) 97
VcsAfterCreateProject (Server) 98
VcsAfterDeleteObject (Server) 98
VcsAfterFreeze (Server) 99
VcsAfterGet (Server) 100
VcsAfterLock (Client) 78
VcsAfterLock (Server) 101
VcsAfterLogin (Server) 102
VcsAfterLogout (Server) 103
VcsAfterPromote (Client) 79
VcsAfterPromote (Server) 103
VcsAfterRemoveVersion (Server) 104
VcsAfterThaw (Server) 92
VcsAfterUnlock (Server) 105
VcsBeforeCheckIn (Client) 74
VcsBeforeCheckOut (Client) 76
VcsBeforeLock (Client) 78
VcsBeforePromote (Client) 79
VcsBeginAction (Client) 81
VcsBeginAction (Server) 106
VcsConfigureAddin (Client) 81
VcsConfigureAddin (Server) 107
VcsEndAction (Client) 82
VcsEndAction (Server) 107
VcsInitAddin (Server) 108
VcsInitEventAddin (Client) 82
VcsReleaseAddin (Client) 83
VcsReleaseAddin (Server) 109
VcsShowAbout (Client) 83

VcsToLocalFormat (Client) 84
VcsToServerFormat (Client) 84