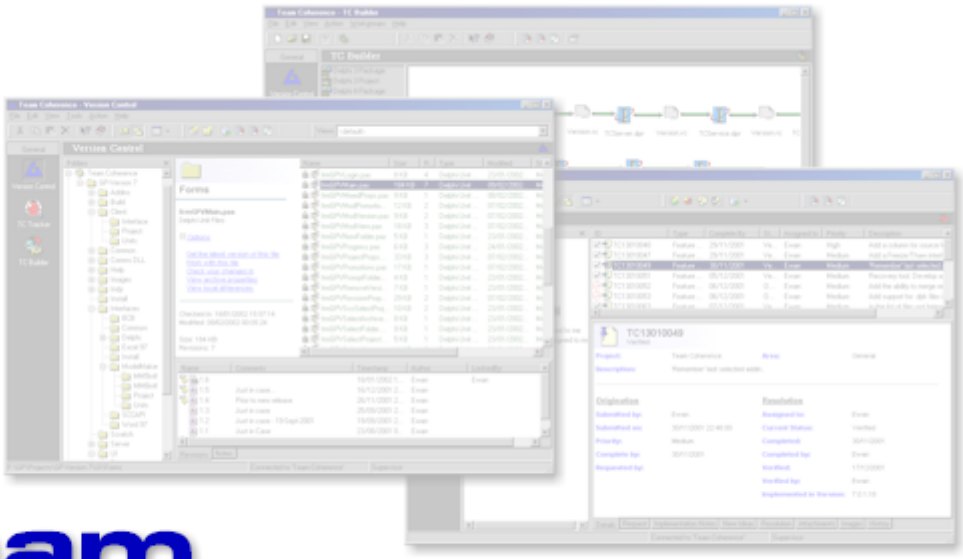
MCN Software Solutions



Team Coherence

www.teamcoherence.com

TC Tracker API Reference

MCN Software Solutions

Tracker 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

Introduction

Part I

1 Introduction

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

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.

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
TCTrkConst.pas	Defines the constants, including error codes, used in the API
TCTrkTypes.pas	Defines the types used by the API
TCTrkUtils.pas	Useful utility functions that help with conversion and initializing.
TrkIntf.pas	The main API wrapper file for **TrkMain.dll**. This DLL contains the User Interface for TC Tracker and provides a higher-level interface. A future release of the API will give lower level access.

Calling Convention

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

1.1 What is Tracker?

TC Tracker is an issue tracking tool designed to integrate with the popular Team Coherence Software Configuration Management (SCM) tool. TC Tracker either runs standalone, or integrates tightly with Team Coherence to add Issue tracking and to link issues with file archives.

Using TC Tracker, you can log and control the progress of several different types of Issue, including Bug Reports and Feature Requests, and maintain a history of how these issues progress. The tight integration with Team Coherence means that Issues can be associated with project files, and vice versa.

1.2 How to buy Tracker

You can order TC Tracker online directly from our home page.

Home page

www.teamcoherence.com

Email support

support@teamcoherence.com

Post

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

Fax

+44 (0)1786 834908

Part II

2 API

2.1 Non-UI Interface (TrkMain.dll)

This DLL exports functions that allow direct, non-UI, access to the main issue management functionality of TC Tracker. 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

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.

2.1.1 TCDTrkConnect

Logon a user to a specific repository.

Delphi (TrkIntf.pas)

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

2.1.2 TCDTrkCreateFolder

This function can be used to create a new Folder.

Delphi (TrkIntf.pas)

```
function TrkCreateFolder( ProjectID: Cardinal; var NewID: Cardinal; Name: String ):
Integer;
```

Parameters

Name	Description
ProjectID	ID of the project that will contain this Folder
NewID	Returns the ID of the newly created Folder
Name	The name to assign to the new folder

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Folders that are created under one user account will not be visible under other accounts.

Exported

```
function TCDTrkCreateFolder( ProjectID: Cardinal; var NewID: Cardinal; pName: PChar ):
Integer; stdcall;
```

2.1.3 TCDTrkCreateIssue

This function can be used to create a new Issue.

Delphi (TrkIntf.pas)

```
function TrkCreateIssue( ProjectID: Cardinal; var NewID: Cardinal; IssueType: Integer;
Description: String; CompleteBy: Integer ): Integer;
```

Parameters

Name	Description
ProjectID	The ID for the project that the new Issue will be assigned to.
NewID	Returns the ID of the newly created Issue
IssueType	The type of Issue to create. See Also: Constants
Description	The text that describes the Issue
CompleteBy	The date (UTC format) that the Issue is to be completed by

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

If you pass 0 to this function for the CompleteBy parameter, Tracker will assign a completeby date of tomorrow.

Exported

```
function TCDTrkCreateIssue( ProjectID: Cardinal; var NewID: Cardinal; IssueType: Integer;
pDescription: PChar; CompleteBy: Integer ): Integer; stdcall;
```

2.1.4 TCDTrkCreateProject

This function can be used to create a new Project.

Delphi (TrkIntf.pas)

```
function TrkCreateProject( var NewID: Cardinal; Name: String ): Integer;
```

Parameters

Name	Description
NewID	Returns the ID of the newly created Project
Name	The name to assign to the new project

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDTrkCreateProject( var NewID: Cardinal; pName: PChar ): Integer; stdcall;
```

2.1.5 TCDTrkCurrentConnection

Returns information about the current connection

Delphi (TrkIntf.pas)

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

2.1.6 TCDTrkDeleteFolder

This function can be used to delete an existing Folder.

Delphi (TrkIntf.pas)

```
function TrkDeleteFolder( 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 method does not delete Issues contained by the folder being deleted. Any Issues contained by the folder will be reassigned to the **Intray**.

Exported

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

2.1.7 TCDTrkDeleteIssue

This function can be used to delete an existing Issue.

Delphi (TrkIntf.pas)

```
function TrkDeleteIssue( IssueID: Cardinal ): Integer;
```

Parameters

Name	Description
IssueID	The ID of the Issue to delete

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDTrkDeleteIssue( IssueID: Cardinal ): Integer; stdcall;
```

2.1.8 TCDTrkDeleteProject

This function can be used to delete an existing Project..

Delphi (TrkIntf.pas)

```
function TrkDeleteProject( ProjectID: Cardinal ): Integer;
```

Parameters

Name	Description
ProjectID	The ID of the Project to delete

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Calling this method will also delete all Issues contained by the Project.

Exported

```
function TCDTrkDeleteProject( ProjectID: Cardinal ): Integer; stdcall;
```

2.1.9 TCDTrkDisconnect

Logs out the current user and disconnects from the repository.

Delphi (TrkIntf.pas)

```
function TrkDisconnect: 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 TCDTrkDisconnect: Integer; stdcall;
```

2.1.10 TCDTrkEnumConnections

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

Delphi (TrkIntf.pas)

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

function TrkEnumConnections( 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 TrkEnumConnections. 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 TCDTrkEnumConnections( Context, Data: Pointer; EnumProc: TIntEnumConnections ):
Integer; stdcall;
```

2.1.11 TCDTrkEnumFields

Enumerates through the list of fields of the issue identified by IssueID, including User Defined fields.

Delphi (TrkIntf.pas)

```
type
  TEnumFields = function ( Data: Pointer; FieldName, FieldValue: String; FieldType: Word;
UserDefined: Boolean ): Boolean;

function TrkEnumFields( IssueID: Cardinal; EnumProc: TEnumFields; Data: Pointer ):
Integer;
```

Parameters

Name	Description
IssueID	The ID of the Issue to enumerate.
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

TEnumFields is the prototype of the procedure that is called by TrkEnumFields. 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 Fields:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
FieldName	Name of the field
FieldValue	Current value of the field
FieldType	The type of the field. See also Type Definitions
UserDefined	True if the field is a User Defined field.

Exported

```
type
  TIntEnumFields = function ( Context, Data: Pointer; pFieldName, pFieldValue: PChar;
FieldType: Word; UserDefined: Boolean ): Boolean; stdcall;

function TCDTrkEnumFields( IssueID: Cardinal; Context, Data: Pointer; EnumProc:
TIntEnumFields ): Integer; stdcall;
```

2.1.12 TCDTrkEnumFolders

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

Delphi (TrkIntf.pas)

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

```
function TrkEnumFolders( RootID: Cardinal; EnumProc: TEnumFolders; Data: Pointer ):
Integer;
```

Parameters

Name	Description
RootID	The ID of the Project 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.

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 TrkEnumFolders. 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
ID	The unique ID used to identify this Folder
Intray	Returns True if the folder is an Intray

Note that, since Folders are user specific, only the folders defined by the User you connected as will be enumerated..

Exported

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

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

2.1.13 TCDTrkEnumIssues

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

Delphi (TrkIntf.pas)

```
type
  TEnumIssues = function ( Data: Pointer; Name: String; ID: Cardinal; IssueData:
PTrkBaseIssue ): Boolean;

function TrkEnumIssues( RootID: Cardinal; EnumProc: TEnumFiles; Data: Pointer ): Integer;
```

Parameters

Name	Description
RootID	The ID of the Project, Folder, or Query that the enumeration starts from.
EnumProc	The procedure (prototype TEnumIssues) that will be called for each Issue 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

TEnumIssues is the prototype of the procedure that is called by TrkEnumIssues. This function should return True to continue enumerating the issues, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each Issue:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
Name	Name of the Issue
ID	The unique ID used to identify this issue
IssueData	Record (of type PTrkBaseIssue) holding basic information about the issue.

Exported

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

function TCDTrkEnumIssues( RootID: Cardinal; Context, Data: Pointer; EnumProc:
TIntEnumIssues ): Integer; stdcall;
```

2.1.14 TCDTrkEnumProjects

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

Delphi (TrkIntf.pas)

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

function TrkEnumProjects( 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 TrkEnumProjects. 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.

Exported

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

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

2.1.15 TCDTrkEnumTemplates

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

Delphi (TrkIntf.pas)

```
type
  TEnumTemplates = function ( Data: Pointer; ID: Cardinal; Name, Description: String;
Scope: Word; Default: Boolean ): Boolean;

function TrkEnumTemplates( EnumProc: TEnumTemplates; Data: Pointer ): Integer;
```

Parameters

Name	Description
EnumProc	The procedure (prototype TEnumTemplates) that will be called for each Template 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

TEnumTemplates is the prototype of the procedure that is called by TrkEnumTemplates. This function should return True to continue enumerating the templates, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each Template:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
ID	Unique ID used to identify the Template within the repository.
Name	Name of the Template
Description	Description of the Template
Scope	The scope of the Template. 0 = Screen, 1 = Page Report, 2 = List Report
Default	True if the template is the default template for the indicated Scope.

Exported

```
type
  TIntEnumTemplates = function ( Context, Data: Pointer; ID: Cardinal; pName,
pDescription: PChar; Scope: Word; Default: Boolean ): Boolean; stdcall;

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

2.1.16 TCDTrkEnumQueries

Enumerates through the list of Queries defined in the repository. This will include Temporary queries created during the session.

Delphi (TrkIntf.pas)

```
type
  TEnumQueries = function ( Data: Pointer; Name: String; ID: Cardinal; Description,
Statement: String; Temporary: Boolean; Shared: Boolean ): Boolean;

function TrkEnumQueries( EnumProc: TEnumQueries; Data: Pointer ): Integer;
```

Parameters

Name	Description
EnumProc	The procedure (prototype TEnumQueries) that will be called for each query 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

TEnumQueries is the prototype of the procedure that is called by TrkEnumQueries. This function should return True to continue enumerating the queries, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each Query:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
Name	Name of the Query
ID	Unique ID used to identify the query
Description	Description of the query
Statement	String representing the criteria of the query
Temporary	True if this is a temporary query
Shared	True if this is a shared query

Exported

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

function TCDTrkEnumQueries( Context, Data: Pointer; EnumProc: TIntEnumQueries ): Integer;
stdcall;
```

2.1.17 TCDTrkGetFieldOptions

Returns the choices available to users, if any, when entering data in a field.

Delphi (TrkIntf.pas)

```
function TrkGetFieldOptions( FieldName: String; const OptionList: TStrings; var Default:
String ): Integer;
```

Parameters

Name	Description
FieldName	The name of the field
OptionList	Returns the selection list for this particular field, or an empty list if the field has no choices
Default	Returns the default value for this field

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Certain fields may have a selection list (eg RequestedBy), but the user does not have to select an item from the list if another value applies.

Exported

```
function TCDTrkGetFieldOptions( pFieldName: PChar; const pOptionList, pDefault: PChar;
var Size: Integer ): Integer; stdcall;
```

2.1.18 TCDTrkGetIssueField

Gets the value of the specified field for the Issue identified by IssueID

Delphi (TrkIntf.pas)

```
function TrkGetIssueField( IssueID: Cardinal; FieldName: String; var FieldValue: String
): Integer;
```

Parameters

Name	Description
IssueID	The ID of the Issue
FieldName	The name of the field to retrieve the value of (not case sensitive)
FieldValue	The value of the field

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The value is always returned as a string, but does not always represent a string. For example, **CompleteBy** returns an integer date in UTC format and should be converted to the local format.

Exported

```
function TCDTrkGetIssueField( IssueID: Cardinal; pFieldName: PChar; const pFieldValue:
PChar; var FieldSize: Integer ): Integer; stdcall;
```

2.1.19 TCDTrkRefreshCache

Reloads the internal cache.

Delphi (TrkIntf.pas)

```
function TrkRefreshCache: Integer;
```

Parameters

None

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Since changes made to issues, and additions/deletions, by other users may not immediately notified to Tracker on your machine, occasional refreshing can help keep this instance up to date.

Exported

```
function TCDTrkRefreshCache: Integer; stdcall;
```

2.1.20 TCDTrkRenameFolder

This function can be used to rename an existing Folder

Delphi (TrkIntf.pas)

```
function TrkRenameFolder( FolderID: Cardinal; Name: String ): Integer;
```

Parameters

Name	Description
FolderID	The ID of the Folder to rename
Name	The new name to assign to the Folder

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

You cannot rename the **Intray** folder.

Exported

```
function TCDTrkRenameFolder( ProjectID: Cardinal; pName: PChar ): Integer; stdcall;
```

2.1.21 TCDTrkRenameProject

This function can be used to rename an existing Project

Delphi (TrkIntf.pas)

```
function TrkRenameProject( ProjectID: Cardinal; Name: String ): Integer;
```

Parameters

Name	Description
ProjectID	The ID of the project to rename
Name	The new name to assign to the project

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDTrkRenameProject( ProjectID: Cardinal; pName: PChar ): Integer; stdcall;
```

2.1.22 TCDTrkSetIssueField

Sets the value of the specified field for the Issue identified by IssueID

Delphi (TrkIntf.pas)

```
function TrkSetIssueField( IssueID: Cardinal; FieldName: String; FieldValue: String ):
Integer;
```

Parameters

Name	Description
IssueID	The ID of the Issue
FieldName	The name of the field to set the value of (not case sensitive)
FieldValue	The new value of the field

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The value should be converted to a string before passing it to this function, for example using the **IntToStr** functions in Delphi. Particular attention should be made to converting dates to UTC format before storing them.

Exported

```
function TCDTrkSetIssueField( IssueID: Cardinal; pFieldName, pFieldValue: PChar ):
Integer; stdcall;
```

2.1.23 TCDTrkSetIssueFields

Uses Name/Value pairs to apply updates to multiple fields at a time.

Delphi (TrkIntf.pas)

```
function TrkSetIssueFields( IssueID: Cardinal; UpdatePairs: String ): Integer;
```

Parameters

Name	Description
IssueID	The ID of the Issue
UpdatePairs	Name/Value pairs separated by a semi-colon (;). e.g. `AssignedTo=Ewan;Status=On Hold`

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The values should be converted to a string before passing them to this function, for example using

the **IntToStr** functions in Delphi. Particular attention should be made to converting dates to UTC format before storing them.

Exported

```
function TCDTrkSetIssueFields( IssueID: Cardinal; pUpdatePairs: PChar ): Integer;
stdcall;
```

2.1.24 TCDTrkUserByID

Returns details of the user defined by an ID.

Delphi (TrkIntf.pas)

```
function TrkUserByID( UID: Cardinal; var Name, FullName, Email, Location: String ):
Integer;
```

Parameters

Name	Description
UID	The ID of the User to get the details for
Name	Returns the Name of the user
FullName	Returns the full name of the user
Email	Returns the Email address of the user
Location	Returns the Location of the User

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDTrkUserByID( UID: Cardinal; const pName, pFullName, pEMail, pLocation: PChar
): Integer; stdcall;
```

2.1.25 TCDTrkUserByName

Returns details of the user defined by a Name.

Delphi (TrkIntf.pas)

```
function TrkUserByName( Name: String; var UID: Cardinal; var FullName, Email, Location:
String ): Integer;
```

Parameters

Name	Description
Name	Name of the user to locate
UID	Returns the unique ID for the user
FullName	Returns the full name of the user
Email	Returns the Email address of the user
Location	Returns the Location of the user

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDTrkUserByID( pName: PChar; var UID: Cardinal; const pFullName, pEMail,
pLocation: PChar ): Integer; stdcall;
```

2.1.26 TrkImportFromXML

Imports issues that have previously been exported or have been created by other tools.

Delphi (TrkIntf.pas)

```
function TrkImportFromXML( FileName: String; CreateUsers, CreateProjects, AsNew: Boolean;
var ErrMsg: String ): Integer;
```

Parameters

Name	Description
FileName	The name of the XML file
CreateUsers	If this value is set to True then any users not already existing in the target repository will be created.
CreateProjects	If this value is set to True then any Projects not already existing in the target repository will be created.
AsNew	If this is set to True, the import routine will ignore the ID of the source issue and create a new Issue in the repository
ErrMsg	Returns a string stating the reason the Import failed

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Unless AsNew is set to True, the import routine will try and match the ID's of the issues being imported to existing Issues in the repository. If you are not updating existing Issues, leave the ID field of the Issue being imported blank.

Prior to importing any Issues, the routine will validate the XML being imported. Any error will cause the import to be abandoned and information about the reason will be returned in ErrMsg.

Exported

```
function TCDTrkImportFromXML( pFile: PChar; CreateUsers, CreateProjects, AsNew: Boolean;
const pErrMsg: PChar ): Integer; stdcall;
```

2.1.27 TrkIssueAsHTMLFile

Exports an issue to a file in HTML format.

Delphi (TrkIntf.pas)

```
function TrkIssueAsHTMLFile( IssueID: Cardinal; FileName: String ): Integer;
```

Parameters

Name	Description
IssueID	The ID of the Issue to export
FileName	Name of the file to create

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This method does not check the existence of the file and if a file with the same name exists, it will be overwritten without a prompt.

Exported

```
function TCDTrkIssueAsHTML( IssueID: Cardinal; pFile: PChar; const pBuffer: PChar; var
Size: Integer ): Integer; stdcall;
```

2.1.28 TrkIssueAsHTMLStr

Exports an issue to a string in HTML format.

Delphi (TrkIntf.pas)

```
function TrkIssueAsHTMLStr( IssueID: Cardinal; var HTMLStr: String ): Integer;
```

Parameters

Name	Description
IssueID	The ID of the Issue to export
HTMLStr	Returns the issue contents expressed in HTML

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Many issues contain large amounts of data and the returned string can be very large. See also TrkIssueAsHTMLFile.

Exported

```
function TCDTrkIssueAsHTML( IssueID: Cardinal; pFile: PChar; const pBuffer: PChar; var
Size: Integer ): Integer; stdcall;
```

2.1.29 TrkIssueAsXMLFile

Exports an issue to a file in XML format.

Delphi (TrkIntf.pas)

```
function TrkIssueAsXMLFile( IssueID: Cardinal; FileName: String; Full: Boolean ):
Integer;
```

Parameters

Name	Description
IssueID	The ID of the Issue to export
FileName	Name of the file to create
Full	If true, this function will export extended fields as well as the core ones.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This method does not check the existence of the file and if a file with the same name exists, it will be overwritten without a prompt.

Exported

```
function TCDTrkIssueAsXML( IssueID: Cardinal; pFile: PChar; const pBuffer: PChar; var
Size: Integer; Full: Boolean ): Integer; stdcall;
```

2.1.30 TrkIssueAsXMLStr

Exports an issue to a string in XML format.

Delphi (TrkIntf.pas)

```
function TrkIssueAsXMLStr( IssueID: Cardinal; var XMLStr: String; Full: Boolean ):
Integer;
```

Parameters

Name	Description
IssueID	The ID of the Issue to export
XMLStr	Returns the issue contents expressed in XML
Full	If true, this function will export extended fields as well as the core ones.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Many issues contain large amounts of data and the returned string can be very large. See also TrkIssueAsXMLFile.

Exported

```
function TCDTrkIssueAsXML( IssueID: Cardinal; pFile: PChar; const pBuffer: PChar; var
Size: Integer; Full: Boolean ): Integer; stdcall;
```

2.1.31 TrkReportAsHTMLFile

Exports an issue to a file in HTML format.

Delphi (TrkIntf.pas)

```
function TrkReportAsHTMLFile( QueryID, TemplateID: Cardinal; PageReport: Boolean; Title,
FileName: String ): Integer;
```

Parameters

Name	Description
QueryID	The ID of the Query to use as the basis of the Report
TemplateID	The ID of the Template to use for the output.
PageReport	True if this is a Page report, False if it is a List report
Title	The Title of the report
FileName	Name of the file to create

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This method does not check the existence of the file and if a file with the same name exists, it will be overwritten without a prompt.

Exported

```
function TTCDTrkExecuteReport( QueryID, TemplateID: Cardinal; PageReport: Boolean;
pTitle, pFile: PChar; const pBuffer: PChar; var Size: Integer ): Integer; stdcall;
```

2.1.32 TrkReportAsHTMLStr

Exports an issue to a file in HTML format.

Delphi (TrkIntf.pas)

```
function TrkReportAsHTMLStr( QueryID, TemplateID: Cardinal; PageReport: Boolean; Title:
String; var HTMLStr: String ): Integer;
```

Parameters

Name	Description
QueryID	The ID of the Query to use as the basis of the Report
TemplateID	The ID of the Template to use for the output.
PageReport	True if this is a Page report, False if it is a List report
Title	The Title of the report
HTMLStr	Returns the contents of the report as a string

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Many issues contain large amounts of data and the returned string can be very large.

Exported

```
function TTCDTrkExecuteReport( QueryID, TemplateID: Cardinal; PageReport: Boolean;
pTitle, pFile: PChar; const pBuffer: PChar; var Size: Integer ): Integer; stdcall;
```

2.1.33 TrkExportAsXMLFile

Exports issues to a file in XML format.

Delphi (TrkIntf.pas)

```
function TrkExportAsXMLFile( RootID: Cardinal; FileName: String ): Integer;
```

Parameters

Name	Description
RootID	The ID of an Issue, Project, or Query
FileName	Name of the file to create

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This method does not check the existence of the file and if a file with the same name exists, it will be overwritten without a prompt.

Exported

```
function TCDTrkExportToXML( RootID: Cardinal; pFile: PChar; const pBuffer: PChar; var
Size: Integer ): Integer; stdcall;
```

2.1.34 TrkExportAsXMLStr

Exports issues to a file in XML format.

Delphi (TrkIntf.pas)

```
function TrkExportAsXMLStr( RootID: Cardinal; var XMLStr: String ): Integer;
```

Parameters

Name	Description
RootID	The ID of an Issue, Project, or Query
XMLStr	Returns the contents of the Issues in XML format.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Many issues contain large amounts of data and the returned string can be very large.

Exported

```
function TCDTrkExportToXML( RootID: Cardinal; pFile: PChar; const pBuffer: PChar; var
Size: Integer ): Integer; stdcall;
```

2.1.35 TrkGetErrorString

Returns a description of the passed ErrorCode..

Delphi (TrkIntf.pas)

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

Parameters

Name	Description
ID	An ErrorCode generated by calling the API

Return Value

A string giving the description of the Error.

Exported

```
function TCDTrkGetErrorString( ID: Integer; const pMsg: PChar; var Size: Integer ):
Integer; stdcall;
```

2.2 Handle-Based API

The Handle-based API is designed to allow you to create server-side interfaces to Tracker information. For example it could be used to create a CGI or ISAPI wrapper to create a Web-based client interface to allow users to View and Create issues through a browser.

The handle-based API is a subset of the main API and does not support Folders, User Queries, or Temporary Queries. The API is implemented through the TrkMain.dll dynamic link library.

Certain settings need to be configured before the API can be used. Since the API is installed on a remote server and all connections through this API share the same underlying account information in the connection to the Team Coherence Server, the DLL that implements the API must have enough information to create this connection.

In order to enable connections through the API, the following information should be added to a file called TrkMain.ini located in the same directory as the TrkMain.dll file:

```
[ServerSide]
Connection Name=<Connection>
User=<Username>
Password=<Password>
```

where the values enclosed in <> marks are replaced with actual valid values.

Even though the API uses this connection information to communicate with the server, all actions are carried out and validated using the Username and Password supplied during the call to TCDSTrkConnect.

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.

2.2.1 TCDSTrkConnect

Logon a user and return a valid Handle for use with the rest of the API.

Delphi (TrkIntf.pas)

```
function TrkConnectSv( Name, Password: String; ExpiresAfter: Integer; var Handle: Cardinal
): Integer;
```

Parameters

Name	Description
Name	The username to login using
Password	The password associated with the user
ExpiresAfter	The number of milliseconds that the handle returned by this function is valid for. If the user is inactive for this period, they will automatically be disconnected.
Handle (Out)	If the login is successful, a handle will be returned that can be used in calls to the other API functions.

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 TCDSTrkConnect( pName, pPassword: PChar; ExpiresAfter: Integer; var Handle:
Cardinal ): Integer; stdcall;
```

2.2.2 TCDSTrkCreateIssue

This function can be used to create a new Issue.

Delphi (TrkIntf.pas)

```
function TrkCreateIssue( Handle, ProjectID: Cardinal; var NewID: Cardinal; IssueType:
Integer; Description: String; CompleteBy: Integer ): Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
ProjectID	The ID for the project that the new Issue will be assigned to.
NewID	Returns the ID of the newly created Issue
IssueType	The type of Issue to create. See Also: Constants
Description	The text that describes the Issue
CompleteBy	The date (UTC format) that the Issue is to be completed by

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

If you pass 0 to this function for the CompleteBy parameter, Tracker will assign a completeby date of tomorrow.

If the lease has expired for the handle, the function will fail and return `Err_UnknownUserOrLeaseExpired`.

Exported

```
function TCDSTrkCreateIssue( Handle, ProjectID: Cardinal; var NewID: Cardinal; IssueType:
Integer; pDescription: PChar; CompleteBy: Integer ): Integer; stdcall;
```

2.2.3 TCDSTrkCreateProject

This function can be used to create a new Project.

Delphi (TrkIntf.pas)

```
function TrkCreateProjectSv( Handle: Cardinal; var NewID: Cardinal; Name: String ):
Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
NewID	Returns the ID of the newly created Project
Name	The name to assign to the new project

Return Value

Err_OK if successful or an error code indicating the error.

If the lease has expired for the handle, the function will fail and return `Err_UnknownUserOrLeaseExpired`.

Exported

```
function TCDSTrkCreateProject( Handle: Cardinal; var NewID: Cardinal; pName: PChar ):
Integer; stdcall;
```

2.2.4 TCDSTrkDeleteIssue

This function can be used to delete an existing Issue.

Delphi (TrkIntf.pas)

```
function TrkDeleteIssueSv( Handle, IssueID: Cardinal ): Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
IssueID	The ID of the Issue to delete

Return Value

Err_OK if successful or an error code indicating the error.

If the lease has expired for the handle, the function will fail and return `Err_UnknownUserOrLeaseExpired`.

Exported

```
function TCDSTrkDeleteIssue( Handle, IssueID: Cardinal ): Integer; stdcall;
```

2.2.5 TCDSTrkDeleteProject

This function can be used to delete an existing Project..

Delphi (TrkIntf.pas)

```
function TrkDeleteProjectSv( Handle, ProjectID: Cardinal ): Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
ProjectID	The ID of the Project to delete

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Calling this method will also delete all Issues contained by the Project.

Exported

```
function TCDSTrkDeleteProject( Handle, ProjectID: Cardinal ): Integer; stdcall;
```

2.2.6 TCDSTrkDisconnect

Logs out the current user and disconnects the user identified by Handle from the repository and releases the internal resources associated with Handle.

Delphi (TrkIntf.pas)

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

Parameters

Name	Description
Handle	A valid Handle used to identify the User.

Return Value

Err_OK if successful or an error code indicating the error.

Exported

```
function TCDSTrkDisconnect( Handle: Cardinal ): Integer; stdcall;
```

2.2.7 TCDSTrkEnumFields

Enumerates through the list of fields of the issue identified by IssueID, including User Defined fields.

Delphi (TrkIntf.pas)

```
type
  TEnumFields = function ( Data: Pointer; FieldName, FieldValue: String; FieldType: Word;
UserDefined: Boolean ): Boolean;

function TrkEnumFieldsSv( Handle, IssueID: Cardinal; EnumProc: TEnumFields; Data: Pointer
): Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
IssueID	The ID of the Issue to enumerate.
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

TEnumFields is the prototype of the procedure that is called by TrkEnumFieldsSv. 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 Fields:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
FieldName	Name of the field
FieldValue	Current value of the field
FieldType	The type of the field. See also Type Definitions
UserDefined	True if the field is a User Defined field.

Exported

```
type
  TIntEnumFields = function ( Context, Data: Pointer; pFieldName, pFieldValue: PChar;
FieldType: Word; UserDefined: Boolean ): Boolean; stdcall;

function TCDSTrkEnumFields( Handle, IssueID: Cardinal; Context, Data: Pointer; EnumProc:
TIntEnumFields ): Integer; stdcall;
```

2.2.8 TCDSTrkEnumIssues

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

Delphi (TrkIntf.pas)

```
type
  TEnumIssues = function ( Data: Pointer; Name: String; ID: Cardinal; IssueData:
PTrkBaseIssue ): Boolean;

function TrkEnumIssuesSv( Handle, RootID: Cardinal; EnumProc: TEnumFiles; Data: Pointer
): Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
RootID	The ID of the Project, Folder, or Query that the enumeration starts from.
EnumProc	The procedure (prototype TEnumIssues) that will be called for each Issue 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

TEnumIssues is the prototype of the procedure that is called by TrkEnumIssuesSv. This function should return True to continue enumerating the issues, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each Issue:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
Name	Name of the Issue
ID	The unique ID used to identify this issue
IssueData	Record (of type PTrkBaseIssue) holding basic information about the issue.

Exported

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

function TCDSTrkEnumIssues( Handle, RootID: Cardinal; Context, Data: Pointer; EnumProc:
TIntEnumIssues ): Integer; stdcall;
```

2.2.9 TCDSTrkEnumProjects

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

Delphi (TrkIntf.pas)

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

function TrkEnumProjectsSv( Handle: Cardinal; EnumProc: TEnumProjects; Data: Pointer ):
Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
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 TrkEnumProjectsSv. 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.

Exported

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

function TCDSTrkEnumProjects( Handle: Cardinal; Context, Data: Pointer; EnumProc:
TIntEnumProjects ): Integer; stdcall;
```

2.2.10 TCDSTrkEnumTemplates

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

Delphi (TrkIntf.pas)

```
type
  TEnumTemplates = function ( Data: Pointer; ID: Cardinal; Name, Description: String;
Scope: Word; Default: Boolean ): Boolean;

function TrkEnumTemplates( Handle: Cardinal; EnumProc: TEnumTemplates; Data: Pointer ):
```

```
Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
EnumProc	The procedure (prototype TEnumTemplates) that will be called for each Template 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

TEnumTemplates is the prototype of the procedure that is called by TrkEnumTemplates. This function should return True to continue enumerating the templates, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each Template:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
ID	Unique ID used to identify the Template within the repository.
Name	Name of the Template
Description	Description of the Template
Scope	The scope of the Template. 0 = Screen, 1 = Page Report, 2 = List Report
Default	True if the template is the default template for the indicated Scope.

Exported

```
type
  TIntEnumTemplates = function ( Context, Data: Pointer; ID: Cardinal; pName,
pDescription: PChar; Scope: Word; Default: Boolean ): Boolean; stdcall;

function TCDTrkEnumTemplates( Handle: Cardinal; Context, Data: Pointer; EnumProc:
TIntEnumProjects ): Integer; stdcall;
```

2.2.11 TCDSTrkEnumQueries

Enumerates through the list of Queries defined in the repository. This will include Temporary queries created during the session.

Delphi (TrkIntf.pas)

```
type
  TEnumQueries = function ( Data: Pointer; Name: String; ID: Cardinal; Description,
Statement: String; Temporary: Boolean; Shared: Boolean ): Boolean;

function TrkEnumQueriesSv( Handle: Cardinal; EnumProc: TEnumQueries; Data: Pointer ):
Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
EnumProc	The procedure (prototype TEnumQueries) that will be called for each query 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

TEnumQueries is the prototype of the procedure that is called by TrkEnumQueriesSv. This function should return True to continue enumerating the queries, or False to terminate the enumeration. The Enumeration procedure receives the following parameters, describing each Query:

Name	Description
Data	The Data parameter as passed into the enumerator. Can be **nil**
Name	Name of the Query
ID	Unique ID used to identify the query
Description	Description of the query
Statement	String representing the criteria of the query
Temporary	True if this is a temporary query
Shared	True if this is a shared query

Exported

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

function TCDSTrkEnumQueries( Handle: Cardinal; Context, Data: Pointer; EnumProc:
TIntEnumQueries ): Integer; stdcall;
```

2.2.12 TCDSTrkGetFieldOptions

Returns the choices available to users, if any, when entering data in a field.

Delphi (TrkIntf.pas)

```
function TrkGetFieldOptionsSv( Handle: Cardinal; FieldName: String; const OptionList:
TStrings; var Default: String ): Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
FieldName	The name of the field
OptionList	Returns the selection list for this particular field, or an empty list if the field has no choices
Default	Returns the default value for this field

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Certain fields may have a selection list (eg RequestedBy), but the user does not have to select an item from the list if another value applies.

If the lease has expired for the handle, the function will fail and return `Err_UnknownUserOrLeaseExpired`.

Exported

```
function TCDSTrkGetFieldOptions( Handle: Cardinal; pFieldName: PChar; const pOptionList,
pDefault: PChar; var Size: Integer ): Integer; stdcall;
```

2.2.13 TCDSTrkGetIssueField

Gets the value of the specified field for the Issue identified by IssueID

Delphi (TrkIntf.pas)

```
function TrkGetIssueField( Handle, IssueID: Cardinal; FieldName: String; var FieldValue:
String ): Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
IssueID	The ID of the Issue
FieldName	The name of the field to retrieve the value of (not case sensitive)
FieldValue	The value of the field

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The value is always returned as a string, but does not always represent a string. For example, **CompleteBy** returns an integer date in UTC format and should be converted to the local format.

If the lease has expired for the handle, the function will fail and return `Err_UnknownUserOrLeaseExpired`.

Exported

```
function TCDSTrkGetIssueField( Handle, IssueID: Cardinal; pFieldName: PChar; const
pFieldValue: PChar; var FieldSize: Integer ): Integer; stdcall;
```

2.2.14 TCDSTrkRefreshCache

Reloads the internal cache.

Delphi (TrkIntf.pas)

```
function TrkRefreshCacheSv: Integer;
```

Parameters

None

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Since changes made to issues, and additions/deletions, by other users may not immediately notified to Tracker on your machine, occasional refreshing can help keep this instance up todate.

Exported

```
function TCDSTrkRefreshCache: Integer; stdcall;
```

2.2.15 TCDSTrkRenameProject

This function can be used to rename an existing Project

Delphi (TrkIntf.pas)

```
function TrkRenameProjectSv( Handle, ProjectID: Cardinal; Name: String ): Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
ProjectID	The ID of the project to rename
Name	The new name to assign to the project

Return Value

Err_OK if successful or an error code indicating the error.

If the lease has expired for the handle, the function will fail and return `Err_UnknownUserOrLeaseExpired`.

Exported

```
function TCDSTrkRenameProject( Handle, ProjectID: Cardinal; pName: PChar ): Integer;
```

```
stdcall;
```

2.2.16 TCDSTrkSetIssueField

Sets the value of the specified field for the Issue identified by IssueID

Delphi (TrkIntf.pas)

```
function TrkSetIssueFieldSv( Handle, IssueID: Cardinal; FieldName: String; FieldValue:
String ): Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
IssueID	The ID of the Issue
FieldName	The name of the field to set the value of (not case sensitive)
FieldValue	The new value of the field

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

The value should be converted to a string before passing it to this function, for example using the **IntToStr** functions in Delphi. Particular attention should be made to converting dates to UTC format before storing them.

If the lease has expired for the handle, the function will fail and return `Err_UnknownUserOrLeaseExpired`.

Exported

```
function TCDSTrkSetIssueField( Handle, IssueID: Cardinal; pFieldName, pFieldValue: PChar
): Integer; stdcall;
```

2.2.17 TCDSTrkReportAsHTMLFile

Exports an issue to a file in HTML format.

Delphi (TrkIntf.pas)

```
function TrkReportAsHTMLFileSv( Handle, QueryID, TemplateID: Cardinal; PageReport:
Boolean; Title, FileName: String ): Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
QueryID	The ID of the Query to use as the basis of the Report
TemplateID	The ID of the Template to use for the output.
PageReport	True if this is a Page report, False if it is a List report
Title	The Title of the report
FileName	Name of the file to create

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This method does not check the existence of the file and if a file with the same name exists, it will be overwritten without a prompt.

Exported

```
function TTCDSTrkExecuteReport( Handle, QueryID, TemplateID: Cardinal; PageReport:
Boolean; pTitle, pFile: PChar; const pBuffer: PChar; var Size: Integer ): Integer;
stdcall;
```

2.2.18 TCDSTrkReportAsHTMLStr

Exports an issue to a file in HTML format.

Delphi (TrkIntf.pas)

```
function TrkReportAsHTMLStrSv( Handle, QueryID, TemplateID: Cardinal; PageReport:
Boolean; Title: String; var HTMLStr: String ): Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
QueryID	The ID of the Query to use as the basis of the Report
TemplateID	The ID of the Template to use for the output.
PageReport	True if this is a Page report, False if it is a List report
Title	The Title of the report
HTMLStr	Returns the contents of the report as a string

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Many issues contain large amounts of data and the returned string can be very large.

Exported

```
function TTCDSTrkExecuteReport( Handle, QueryID, TemplateID: Cardinal; PageReport:
Boolean; pTitle, pFile: PChar; const pBuffer: PChar; var Size: Integer ): Integer;
stdcall;
```

2.2.19 TCDSTrkExportAsXMLFile

Exports issues to a file in XML format.

Delphi (TrkIntf.pas)

```
function TrkExportAsXMLFileSv( Handle, RootID: Cardinal; FileName: String ): Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
RootID	The ID of an Issue, Project, or Query
FileName	Name of the file to create

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

This method does not check the existence of the file and if a file with the same name exists, it will be overwritten without a prompt.

Exported

```
function TCDSTrkExportToXML( Handle, RootID: Cardinal; pFile: PChar; const pBuffer:
PChar; var Size: Integer ): Integer; stdcall;
```

2.2.20 TCDSTrkExportAsXMLStr

Exports issues to a file in XML format.

Delphi (TrkIntf.pas)

```
function TrkExportAsXMLStrSv( Handle, RootID: Cardinal; var XMLStr: String ): Integer;
```

Parameters

Name	Description
Handle	A valid Handle used to identify the User.
RootID	The ID of an Issue, Project, or Query
XMLStr	Returns the contents of the Issues in XML format.

Return Value

Err_OK if successful or an error code indicating the error.

Remarks

Many issues contain large amounts of data and the returned string can be very large.

Exported

```
function TCDSTrkExportToXML( Handle, RootID: Cardinal; pFile: PChar; const pBuffer:
PChar; var Size: Integer ): Integer; stdcall;
```

2.2.21 TCDSTrkGetErrorString

Returns a description of the passed errorcode.

Delphi (TrkIntf.pas)

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

Parameters

Name	Description
ID	An ErrorCode generated by calling the API

Return Value

A string giving the description of the Error.

Exported

```
function TCDSTrkGetErrorString( ID: Integer; const pMsg: PChar; var Size: Integer ):
Integer; stdcall;
```

2.3 Constants (TCTrkConst.pas)

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

Issue Types

```
it_Bug              = 1;
it_Todo             = 2;
it_Enhancement      = 4;
it_Task             = 8;
it_WorkRequest      = 16;
it_InfoRequest      = 32;
it_ChangeRequest    = 64;
it_Requirement      = 128;
it_Idea             = 256;
```

Field Types

```
ftUDString   = 0;
ftUDInteger  = 1;
ftUDDate     = 2;
ftUDFloat    = 3;
```

Error Codes

```
Err_OK                           = 0;
Err_InsufficientAccess           = 14;
Err_InvalidName                  = 39;
Err_ObjectNotFound               = 40;
Err_NotLoggedIn                  = 48;
Err_ConnectionNotDefined         = 49;
Err_ObjectNameAlreadyInUse   = 54;
Err_Offline                      = 59;
Err_ServerError                  = 61;
Err_PersonalLicense          = 74;
Err_NoLicenses                   = 75;
Err_CouldNotCreateFile           = 76;
Err_InvalidConnection            = 78;
Err_AlreadyConnected             = 79;
Err_NotConnected                 = 80;
Err_CouldNotConnectToServer  = 81;
Err_TCNotInstalled               = 101;
Err_ErrorUpdatingImage           = 102;
Err_ErrorUpdatingObject          = 103;
Err_ObjectAlreadyIncluded        = 104;
Err_ErrorUpdatingIssue           = 105;
Err_InvalidRecordSize            = 106;
Err_AddinPreventedModification   = 107;
Err_AddinPreventedDeletion   = 108;
Err_InvalidFieldName             = 109;
Err_InvalidFieldValue            = 110;
Err_CannotModifyField            = 111;
Err_ErrorUpdatingFile            = 112;
Err_InvalidUsername          = 113;
Err_OwnerNotDefined          = 114;
Err_ProjectNotDefined            = 115;
Err_NoDescription                = 116;

Err_InvalidServerVersion         = 150;
Err_UnknownUserOrLeaseExpired    = 151;
```

2.4 Type Definitions (TCTrkTypes.pas)

There is one main type definitions for the API. This is the record used to pass basic issue data to certain function calls:

TrkBaseIssue

```
type
  TTrkBaseIssue = record
    IssueType: Integer;
    // Users
    ID: Cardinal;
    Owner: Cardinal;              // ID of the Owner
    AssignedTo: Cardinal;         // ID of the user the issue is currently assigned to
    CompletedBy: Cardinal;        // ID of the user that completed the issue
    VerifiedBy: Cardinal;         // ID of the user that verified completion of the issue
    // Timestamps (in UTC format)
    Created: Integer;             // Creation timestamp
    CompleteBy: Integer;          // Date/Time that the issue needs to be completed by
    Completed: Integer;           // Completion timestamp
    Verified: Integer;            // Verification timestamp
    // Pointers to string data
    Name: PChar;
    Priority: PChar;
    Status: PChar;
    Description: PChar;
    Area: PChar;
    Severity: PChar;
    Version: PChar;
    ImplementedInVersion: PChar;
    ReportedBy: PChar;
    Email: PChar;
    ClosureCat: PChar;
    // Access settings
    CanModify: Boolean;
    CanDelete: Boolean;
    CanComplete: Boolean;
    CanVerify: Boolean;
    // Actual string data
    FData: PChar;
  end;
  PTrkBaseIssue = ^TTrkBaseIssue;
```

2.5 Utilities (TCTrkUtils.pas)

This file contains utility functions that may be useful in the context of the TC Tracker API. They are provided simply to make things a bit easier and are not documented in this API.

The following two functions are particularly important because all date/times are stored and returned by the API in UTC format and need to be converted on occasion.

```
function LocalDateToUTCDate( Date: TDateTime ): TDateTime;
function UTCDateToLocalDate( Date: TDateTime ): TDateTime;
```

When reading a date value using the API, it can be converted to a local TDateTime format by making the following call:

```
LocalDateTime := UTCDateToLocalDate( FileDateToDateTime( TrackerDate ) );
```

similarly, before storing a date, it shoudl be converted to UTC using a call similar to the following:

```
TrackerDate := DateTimeToFileDate( LocalDateToUTCDate( LocalDateTime ) );
```

Index

- C -

- D -

- H -

- T -

- U -