

C++ TraceTool API documentation

Table of contents	
Introduction.....	1
Sample use.....	2
API documentation.....	5
1. TraceNode	5
2. TraceNodeEx	6
3. TMemberNode.....	7
4. TTrace.....	8
5. TTraceOptions	9
6. WinWatch	10
7. WinTrace	11
Extend TraceTool with plugins.....	12

Introduction

License information:

You are free to distribute the viewer (GPL) and to incorporate the library (LGPL) in your product (commercial or not).
The only thing you can't do is to sell the viewer (or a modified version) and the API as a tracing solution.

Four frameworks are currently supported: Dot net (C#), Java , Delphi for window and C++
All are NATIVE libraries.

Since theses languages are different, the syntax and the options are a little bit different.

Dot Net:	TTrace.Warning.Send ("Hello");
Java:	TTrace.warning ().send ("hello");
Delphi:	TTrace.Warning.Send ('hello');
C++:	TTrace::Debug()->Send ("Hello");

See the Java Dot net and Delphi TraceTool API documentation's for more details on theses languages.

Important: if you want to use full TraceTool framework under Dot net, use the managed TraceTool library, not this one!

The C++ library (for non managed applications) is a little bit different from the other libraries.
C++ language don't have garbage collector and it's not possible to inspect classes.
Some C++ Tracetool functions return void in place of trace node.

The entry point of the framework is the TTrace class.
That class provide 3 invisibles parents node (Debug, Warning and Error) with specific icon and with the "Enabled" property set to true.
TTrace as some global functions, like Clear () to clear all the traces and Show () to show or hide the viewer
The framework gives the possibility to create multiple Tab traces, see the WinTrace class.

TraceTool propose 2 ways to send traces:

- TraceNode
- TraceNodeEx

In TraceNode mode, traces are sent from a Parent node. (debug, warning and error)
It's the quicker way to send simple traces.

The problem with TraceNode is that the class cannot provide hundreds of overload methods to match the developer needs.
For example Send ("Col1", "Col2", Type) or Send ("col1", IconIndex) .
It's where TraceNodeEx is needed.


In this mode, TraceNodeEx is created first, filled by the developer and then sent.
Using TraceNodeEx, you can specify left and right column separately, give an icon and add some information onto the "member" pane, like dumps, on a single trace.

If you want to separate your application traces and databases trace or Exceptions traces, you can create multiple windows traces
See the WinTrace classes and demos

This C++ library is compatible with pocket PC 2003. See the WinCeDemo workspace (Embedded C++ 4.0)

Sample use

Simple traces

	Time	Thld	Traces	Comment
	15:09:14:671	0xEF0	hello	world

Using TraceNode




```
TTrace::Debug()->Send ("Hello world");  
TTrace::Debug()->Send ("Hello", "world");
```

Note that the icon is the same as TTrace::Debug().
To specify icon, use TraceNodeEx.

Using TraceNodeEx

```
TraceNodeEx * Node ;  
Node = new TraceNodeEx () ;  
Node->SetIconIndex (CST_ICO_CONTROL) ;  
Node->leftMsg = "Hello" ;  
Node->rightMsg = "world" ;  
Node ->Send() ;
```

Hierarchical traces

	Time	Thld	Traces	Comment
	15:05:31:203	0xEF0	 Hello	
	15:05:31:203	0xEF0	World	









Using TraceNode

Unlike the C#, Java and Delphi tracetool API, it's not possible to send hierarchical traces using the Send() method.
The reason is that the Send() method return "void".
The Programmer don't have to free object when calling Send().
To create hierarchical traces, use the TraceNode.Indent() method or use the TraceNodeEx class.

Using TraceNodeEx

```
// create a new trace with no parent  
TraceNodeEx * Node1 = new TraceNodeEx () ;  
Node1->leftMsg = "Hello" ;  
Node1->Send() ;  
  
// create Node2 as a child of Node1  
TraceNodeEx * Node2 = Node1->CreateChildEx ("world") ;  
Node2->Send() ;  
  
// create Node3 as a child of Node2  
TraceNodeEx * Node3 = new TraceNodeEx (Node2) ;  
Node3->leftMsg = "from C++" ;  
Node3->Send() ;  
  
delete Node1 ;  
delete Node2 ;  
delete Node3 ;
```

Hierarchical traces using Ident and UnIndent

	Time	Thld	Traces	Comment
	09:40:006	0x8A0	 Before	some work
	09:40:006	0x8A0	 Level1	
	09:40:006	0x8A0	Level2	
	09:40:006	0x8A0	More level2	
	09:40:006	0x8A0	Done level 1	
	09:40:006	0x8A0	Work is done	

The Ident() function add a message to the specified node.
 UnIndent with or without a message return to previous level.
 This can be used for call stack tracing or loop traces.
 Indent can be used on a specific node or one of the root (debug, warning, error)

Using TraceNode

```
TTrace::Debug()->Indent ("Before", "some work");
TTrace::Debug()->Indent ("Level1") ;
TTrace::Debug()->Send ("Level2") ;
TTrace::Debug()->Send ("More level2") ;
TTrace::Debug()->UnIndent ("Done level 1") ;
TTrace::Debug()->UnIndent ("Work is done") ;
```

Overrides traces : Use one of the ResendXXX or AppendXXX to modify existing text node

Using TraceNode

Resend and Append is not possible with the C++ version, use TraceNodeEx

Using TraceNodeEx

```
TraceNodeEx * node1 = TTrace::Debug()->CreateChildEx("Start 1 ..") ;
node1->Send() ;

node1->Append ("Done 1") ;          // append left part
node1->Resend (NULL, "Done 2") ;    // resend right part
```

Dump

Time	Thld	Traces	Comment
16:07:18.656	0x904	Some dumps	

Using TraceNode

```
char buffer [100] ;
sprintf (buffer, "%u" , GetCurrentProcessId()) ;

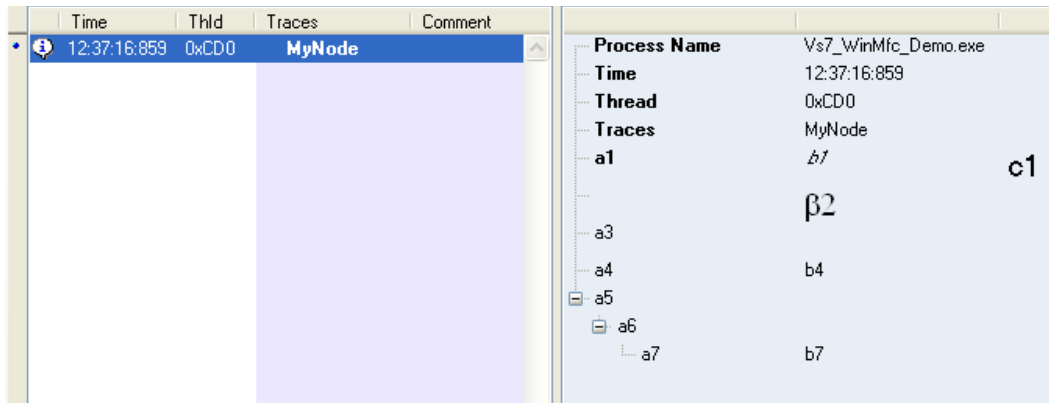
TTrace::Debug()->SendDump ("Dump test", NULL , "Dump" , buffer , 30) ;
```

Using TraceNodeEx

```
char buffer [100] ;
sprintf (buffer, "%u" , GetCurrentProcessId()) ;

TraceNodeEx * node ;
node = TTrace::Debug()->CreateChildEx("Dump test") ;
node-> AddDump ("Dump", buffer, 0 , 30) ;
node->Send() ;
```

TraceNodeEx Members



```
// extended Node with members
TraceNodeEx * node = TTrace::Debug()->CreateChildEx("MyNode") ;
node->AddFontDetail (3,true) ; // change the "MyNode" trace caption to bold
node->Members()
->Add ("a1" , "b1" , "c1") // add member // a1 | b1 | c1
->SetFontDetail (0,true) // a1 bold
->SetFontDetail (1,false,true) // a2 Italic
->SetFontDetail (2,false,false,-1,15) ; // a3 Size 15
node->Members()
->Add (NULL , "b2" ) // add member // | b2 |
->SetFontDetail (1,false,false,-1,15,"Symbol") ; // b2 font Symbol
node->Members()
->Add ("a3") ; // add sub member // a3 | |
node->Members()
->Add ("a4" , "b4" ) ; // add sub member // a4 | b4 |
node->Members()
->Add ("a5") // add member // a5 | |
->Add ("a6") // add sub member // a6 | |
->Add ("a7", "b7") ; // add sub sub member // a7 | b7 |
// finally send the node
node->Send() ;
delete node ;
```

Multiple windows sample

```
WinTrace * MyWinTrace ;

MyWinTrace = new WinTrace("MyId" , "MyTitle") ; // create a new window
MyWinTrace->DisplayWin() ; // ask the viewer to display it in front
MyWinTrace->Debug()->Send ("Hello", "World") ; // send traces to a specific window
MyWinTrace->SaveToXml("c:\\log2.xml") ; // save all the traces to XML
```

Note : the main page is also a WinTrace object. To save all messages use the following command:

```
TTrace::WindowTrace()->SaveToXml ("Main traces.xml") ;
```

Multiple Column support

Multi column is not supported in the main trace window, since many applications (and in different languages) cannot share the main window in "classic" mode and in "multi column" mode.

You must then create a window trace and set the columns titles. (with just a few lines)

In that mode, of course, you lose the automatic thread name and date. You must send them yourself if you need to.

Sending many columns of text is performed with the same API as for simple trace.

Just add a tabulation ("t") between the different columns.

Here is an example :

```
WinTrace * MulticolWintrace ;

MulticolWintrace = new WinTrace("MCOlID" , "MultiCol trace window") ; // create the window
MulticolWintrace->SetMultiColumn () ; // set the window to multi column mode.
MulticolWintrace->SetColumnsTitle("Column A \t Column B \t Column C ") ; // add columns title
MulticolWintrace->DisplayWin() ; // change the active window (optional)
```

Once initialized, use the WinTrace instance to send data:

```
MulticolWintrace->Debug()->Send ("Col1 \t Col2 \t Col3") ; // columns data must separated by tabulations
```

API documentation

The framework uses some public, but you need to learn only the `TraceNode` class for to the majority of the traces.
Traces can be done in 1 line of code :

```
TTrace::Debug()->Send ("Hello C++");
```

The “Win32LibAndDemo.sln” solution demonstrates the different possibilities.

1. `TraceNode`

That class perform the formatting of the message to send.
Nothing is sent by this class (it's `TTrace` class job)

Sample use :

```
TTrace::Debug()->Send ("Hello C++");
```

Constructor: Construct a new trace node, derived from an optional parent node

```
TraceNode (TraceNode * parentNode = NULL, bool generateUniqueId = true);  
TraceNode (TraceNode * parentNode , const char * newNodeId ) ;  
TraceNode (TraceNode * parentNode , const wchar_t * newNodeId );
```

Public fields and get/set methods:

char * id	The unique ID. Normally it's a GUID, but can be replaced by something else for interprocess traces.
bool enabled	When enabled is false, all traces are disabled. Default is true. All nodes have an Enabled property that lets you define group of Enabled trace. For example set the <code>TTrace::Debug()->enabled</code> to false but continue to accept Error and Warning traces
WinTrace * winTrace	The parent win tree
int tag	User variable, provided for the convenience of developers
int GetIconIndex (void)	The Index of the icon to use

Public methods:

void Send (char *leftMsg, char *rightMsg = NULL) ; void Send (wchar_t *wLeftMsg, wchar_t *wRightMsg = NULL) ;	Send simple trace with left and/or right column
void SendDump (char *leftMsg, char *rightMsg, char * title, char * memory, unsigned byteCount) ; void SendDump (wchar_t *leftMsg, wchar_t *rightMsg, wchar_t * title, char * memory, unsigned byteCount) ;	Send memory dump
void Indent (char *leftMsg, char *rightMsg = NULL) ; void Indent (wchar_t *leftMsg, wchar_t *rightMsg = NULL) ;	Send a node based on this node then change the indentation
void UnIndent (char *leftMsg = NULL, char *rightMsg = NULL) ; void UnIndent (wchar_t *leftMsg = NULL, wchar_t *rightMsg = NULL);	Decrement indentation

Create `TraceNodeEx` instances directly from a `TraceNode` object

```
TraceNodeEx * CreateChildEx (char *leftMsg = NULL, char *rightMsg = NULL) ;  
TraceNodeEx * CreateChildEx (wchar_t *leftMsg = NULL, wchar_t *rightMsg = NULL) ;
```

Unlike the C#, Java and Delphi libraries, the `Send` and `Indent` methods don't return a trace node.
It's then not possible to change the trace using `Append` and `Resend` methods.
Theses operations are possible using the `TraceNodeEx` class.

You can however use the `Ident()` method (from `TraceNodeBase`) to send sub traces, see samples uses.

2. TraceNodeEx

Alternate ways to send traces: prepare a TraceNodeEx with all properties and then send it.
TraceNodeEx inherits from TraceNode

Constructors:

```
TraceNodeEx (TraceNode * parentNode = NULL , bool generateUniqueId = true)
TraceNodeEx (TraceNode * parentNode , const char * newNodeId )
TraceNodeEx (TraceNode * parentNode , const wchar_t * newNodeId )
```

If ParentNode parameter is give, the new created object inherit the enabled, iconIndex and winTrace properties.
If the newNodeId is give, the trace id will receive a copy of it; else a new GUID is created.
You can also call CreateChildEx () from a TraceNode or TraceNodeEx instance to create a TraceNodeEx object.

Added Public fields and get/set/add methods :

string rightMsg	Right message
string leftMsg	Left message
TMemberNode * Members()	Members info. See Member chapter
void SetIconIndex (int newIconIndex) ;	Set the Icon index
TraceNodeEx *AddDump(char * Title , char * memory , unsigned index , unsigned byteCount) TraceNodeEx *AddDump(wchar_t * Title , char * memory , unsigned index , unsigned byteCount) ;	add dump to the members info
TraceNodeEx * AddFontDetail(const int ColId, const bool Bold , const bool Italic = false, const int Color = -1 , const int Size = 0 , const char * FontName = NULL) ;	Change font detail for an item in the trace ColId is the Column index : Icon=0, Time=1, thread=2, left msg=3, right msg =4 or user defined column

Added Public methods:

void Send (void)	send the node
------------------	---------------

When the node is send, you can use theses methods to manage the node:

void Resend (char *LeftMsg, char *RightMsg = NULL)	resend left and right traces
void Resend (wchar_t *LeftMsg, wchar_t *RightMsg = NULL)	
void Append (char *LeftMsg, char *RightMsg = NULL)	append left and right traces
void Append (char *LeftMsg, char *RightMsg = NULL)	
void Show ()	ensure the trace is visible in the viewer
void SetSelected()	set the node as selected
void Delete()	delete the node and all children
void DeleteChildren()	delete the trace children, not the node itself

3. TMemberNode

TMemberNode represent a node information in the "info" trace tree. Members are available in the TraceNodeEx class.
You should normally not create TmemberNode directly. Use one of the Add methods to create members.

Public constructor :

<code>TMemberNode();</code>	Create a TMemberNode with no text in the 3 columns
<code>TMemberNode(char * Col1 = NULL, char * Col2 = NULL, char * Col3 = NULL); TMemberNode(wchar_t * Col1 , wchar_t * Col2 = NULL, wchar_t * Col3 = NULL)</code>	Create a TMemberNode with 1,2 or 3 columns

Public fields :

<code>public string Col1; public string Col2; public string Col3; public ArrayList Members; public int Tag;</code>	The 3 columns to display
	An array of sub members (TMemberNode)
	User defined tag, NOT SEND to the viewer

Public methods :

<code>TMemberNode * Add (char * Col1, char * Col2 = NULL, char * Col3 = NULL); TMemberNode * Add (wchar_t * Col1, wchar_t * Col2 = NULL, wchar_t * Col3 = NULL)</code>	Create a TMemberNode with with 1,2 or 3 columns
<code>TMemberNode * Add (TMemberNode * member)</code>	Add a member to the members list
<code>TMemberNode * SetFontDetail(const int ColId, const bool Bold , const bool Italic = false , const int Color = -1 , const int Size = 0 , const char * FontName = NULL)</code>	Set member font ColId is the column number (0..2)
<code>void AddToStringList(ArrayList CommandList);</code>	Recursively add members to the node commandList

4. TTrace

TTrace is the entry point for all traces. He give 3 'TraceNode' doors: Warning, Error and Debug.
Theses 3 doors are displayed with a special icon. All of them have the 'enabled' property set to true.
You can set it to false to limit logging. That class is fully static.

Public methods :

TraceNode * Warning()	Invisible trace node, with the CST_ICO_WARNING icon index. Shortcut to TTrace::WindowTrace->Warning()
TraceNode * Error()	Invisible trace node, with the CST_ICO_ERROR icon index. Shortcut to TTrace::WindowTrace->Error()
TraceNode * Debug()	Invisible trace node, with the CST_ICO_INFO icon index. Shortcut to TTrace::WindowTrace->Debug()
TraceOptions * Options()	The TTrace Options.
WinTrace * WindowTrace()	The WinTrace where traces are send
static void ClearAll ()	Clear all traces on the main windows trace. Shortcut to TTrace.WinTrace.ClearAll () ;
static void Show (bool IsVisible)	Show or hide the trace program.
static void CloseSocket()	Close the socket (Socket is automatically closed when destroyed. No need to create destructor)
static void Stop ()	Stop tracetool sub system. Must be called before exiting plugin
static char * CreateTraceID ()	Creates unique GUID.

For example, to clear all traces in the main window, call the clearAll method :
TTrace.ClearAll () ;

To save all messages from the main window, use the WinTrace object (see WinTrace documentation class):
TTrace.WinTrace.SaveToXml ("Main traces.xml") ;

5. TTraceOptions

Options for the traces, accessible from TTrace::Options()

Public fields and get/set methods:

SendMode sendMode	Change SendMode to Mode.Socket to use it under ASP. Deault is SendMode.WinMsg
void SetSocketHost (char * Host) void SetSocketHost (wchar_t * Host)	The Socket Host address. Default is 127.0.0.1
int socketPort	The socket port. Default is 8090.
bool SendProcessName	Indicate if the process name must be send. Displayed on the status bar. Default is false.
const char * GetProcessName()	return the process name

The default Host is LocalHost on the 8090 port.

6. WinWatch

Represent a Watch window in the viewer. The main Watch window can be accessed using `TTrace::Watches()`

Watches are languages independents: you can send a C++ Watch to the viewer that can be changed by a Java application. Watches are displayed in a tree

Unlike the other languages supported by TraceTool, only string value can be displayed.

Public constructor :

<code>WinWatch ()</code>	Used to attach to an existing watch window.
<code>WinWatch (char * WinWatchID , char * WinWatchText)</code> <code>WinWatch (wchar_t * WinWatchID , wchar_t * WinWatchText)</code>	Create a new WinWatch on the viewer.

Public fields :

<code>String id</code>	The "Required" Id of the window, can be any string, or a guid.
<code>bool enabled</code>	When enabled is false, watches are disabled.
<code>int tag</code>	User variable, provided for the convenience of developers

Public methods :

<code>void ClearAll (void)</code>	Clear all watches in that window
<code>void DisplayWin (void)</code>	Switch viewer to this window
<code>void Send(char *WatchName , char *WatchValue)</code> <code>void Send (wchar_t * WatchName , wchar_t * WatchValue)</code>	Send a watch. The first parameter is the watch name.

Samples :

```
// clear main watch window
TTrace::Watches()->ClearAll() ;

// display main watch window
TTrace::Watches()->DisplayWin() ;

// send watch to main watch window
TTrace::Watches()->Send ("test2" , TTrace::CreateTraceID()) ;

// create a new watch window
WinWatch * MyWinWatch = NULL ;
MyWinWatch = new WinWatch ("MyWinWatchID" , "My watches") ;

// send watch to the new winwatch window

SYSTEMTIME Time;
char buffer [MAX_PATH] ;

GetLocalTime(&Time);
sprintf(buffer, "%02d:%02d:%02d:%03d", Time.wHour, Time.wMinute, Time.wSecond, Time.wMilliseconds);

MyWinWatch->Send ("Now" , buffer) ;

// display the new watch window
MyWinWatch->DisplayWin() ;
```

7. WinTrace

Represent a trace window in the viewer. TTrace use a WinTrace object for the main trace window.

Public constructor:

WinTrace(void)	You can map a WinTrace to an existing window. Nothing Is send to the viewer.
WinTrace(char * winTraceID, char * Title)	The Window Trace is create on the viewer (if not already done)
WinTrace (wchar_t * winTraceID , wchar_t * Title)	

Public fields and properties :

TraceNode * Debug() TraceNode * Warning() TraceNode * Error()	Warning, Error and Debug are the 3 doors to send traces
string id	The "Required" Id of the window tree. Can be any string, or a GUID. The Main window trace Id is empty
int tag	User variable, provided for the convenience of developers

Public methods:

void SaveToTextfile (char * FileName) void SaveToTextfile (wchar_t * FileName)	Save the window tree traces to a text file (Path is relative to the viewer)
void SaveToXml (char * FileName) void SaveToXml (wchar_t * FileName)	Save the window tree traces to an XML file (Path is relative to the viewer)
void LoadXml (char * FileName) void LoadXml (wchar_t * FileName)	Load an XML file to the window tree traces (Path is relative to the viewer)
void DisplayWin ()	Show the window tree Tab on the viewer
void ClearAll ()	Clear all trace for the window tree
void SetMultiColumn(int MainColIndex = 0)	Change the tree to display user defined multiple columns Must be called before calling setColumnsTitle The optional MainColIndex parameter indicate the tree column index
void SetColumnsTitle (char * Titles) void SetColumnsTitle (wchar_t * Titles)	Set columns title (titles must be separated by tabulations)
void SetColumnsWidth (char * Widths) void SetColumnsWidth (wchar_t * Widths)	Tab separated columns width The format for each column is width[:Min[:Max]] Example : 100:20:80 \t 200:50 \t 100
void SetLogFile (char * FileName, int Mode) void SetLogFile (wchar_t * FileName, int Mode)	Set the log file name and file type (Path is relative to the viewer) Mode can be 0 : No log 1 : Log enabled, no size limit 2 : Log enabled, daily file. The file name include the Filename param and the date

WinTrace Plugin API : See The plugin chapter.

Samples :

```
WinTrace * MyWinTrace ;  
// create a new trace window  
MyWinTrace = new WinTrace("MyId" , "MyTitle") ;  
  
// Ask the viewer switch to the specified window (optional)  
MyWinTrace->DisplayWin() ;  
  
// send a message to the trace window  
MyWinTrace->Debug()->Send ("Hello", "World") ;
```

MultiColum sample :

```
WinTrace * MulticolWintrace ;  
MulticolWintrace = new WinTrace("MCOlID" , "MultiCol trace window") ;  
MulticolWintrace->SetMultiColumn () ; // must be called before calling setColumnsTitle  
MulticolWintrace->SetColumnsTitle("Column A \t Column B \t Column C ") ;  
MulticolWintrace->DisplayWin() ;  
  
int col1 , col2 , col3 ;  
col1 = rand() ;  
col2 = rand() ;  
col3 = rand() ;  
sprintf (buffer,"%d\t%d\t%d", col1 ,col2 ,col3) ;  
MulticolWintrace->Debug()->Send (buffer) ;
```

Extend TraceTool with plugins

You can extend TraceTool functionalities using plugins.

Plugins can be written in Dot net, Java or in any other language that can create a classic win32 DLL (C++ , Delphi,...)

With plugins, you can add for example key logger, registry logger or protocol analyser to TraceTool without having to create a separated application.

Plugins receive some user interface events. You can change the original behaviour or prevent them to occur.

For example, when a user when to clear a “fixed” node, the plugin receive the event , and discard the action.

With the plugin API , you can create new labels, buttons or menu items for a specified trace window or discard them.

A clipboard logger plugin can, for example, disable the COPY button in the ‘Clipboard’ trace window.

An ODBC logger plugin can add an ‘Option’ button on the ‘ODBC’ trace window. The plugin will receive an event when the user click on the button and show a dialog box.

Plugins send command to TraceTool using the classic TraceTool framework (WinTrace object).

The plugin API manipulate resources and actions.

Tracetool resources:

CST_ACTION_CUT	Cut. Same as copy then delete
CST_ACTION_COPY	Copy
CST_ACTION_DELETE	Delete selected
CST_ACTION_SELECT_ALL	Select all
CST_ACTION_RESIZE_COLS	Resize columns
CST_ACTION_VIEW_INFO	View trace info
CST_ACTION_VIEW_PROP	View properties
CST_ACTION_PAUSE	Pause
CST_ACTION_SAVE	SaveToFile
CST_ACTION_CLEAR_ALL	Clear all
CST_ACTION_CLOSE_WIN	Close win
CST_ACTION_LABEL_INFO	TracesInfo label
CST_ACTION_LABEL_LOGFILE	LabelLogFile label
CST_ACTION_VIEW_MAIN	View Main trace
CST_ACTION_VIEW_ODS	ODS
CST_ACTION_OPEN_XML	XML trace -> Tracetool XML traces
CST_ACTION_EVENTLOG	Event log
CST_ACTION_TAIL	Tail

Resource type :

CST_RES_BUT_RIGHT	Button on right
CST_RES_BUT_LEFT	Button on left
CST_RES_LABEL_RIGHT	Label on right
CST_RES_LABELH_RIGHT	Label on right HyperLink
CST_RES_LABEL_LEFT	Label on left
CST_RES_LABELH_LEFT	Label on left hyperlink
CST_RES_MENU_ACTION	Item menu in the Actions Menu
CST_RES_MENU_WINDOW	Item menu in the Windows Menu. Call CreateResource on the main win trace to create this menu item

Here are the 4 specifics WinTrace functions :

```
void CreateResource (int ResId , int ResType , int ResWidth , char * ResText = NULL)
void CreateResource (int ResId , int ResType , int ResWidth , wchar_t * ResText = NULL)
```

Create a ‘Plugin’ resource (Button , label or menu item) for the window.

ResId is the resource Id . Must be >= 100

ResType is the resource

ResWidth is the resource width. Applicable only to button and labels

ResText is the resource text

```
void DisableResource (int ResId)
```

Disable tracetool or user created resources.

ResId is the user or tracetool resource Id to disable.

```
void SetTextResource (int ResId, char * ResText)
```

```
void SetTextResource (int ResId, wchar * ResText)
```

Set the text of a resource (Tracetool or user created).

ResId is the resource Id . Must be >= 100

ResText is the resource text

```
void LinkToPlugin (char * PluginName, int flags)
void LinkToPlugin (wchar * PluginName, int flags)
```

Attach a winTrace to a plugin.

Many winTrace can be attached to a plugin. Note that a plugin don't need to be attached to a WinTrace.

The plugin is identified by his internal name (not dll name). When linked, the plugin can receive event (see ITracePlugin).

PluginName is the plugin name

flags is a combinaison of CST_PLUGIN_ONACTION , CST_PLUGIN_ONBEFOREDELETE , CST_PLUGIN_ONTIMER.

To reduce plugin communication, you can ask TraceTool to call only the “OnBeforeDelete” plugin function

Communication between TraceTool and the plugin.

C++ plugins are a special case : you can create plugins using the TraceTool for Dot Net framework or create a classic DLL and use the non managed C++ framework.

You must create a DLL that export theses 6 functions.

OnAction(), OnBeforeDelete() and OnTimer() functions events are NOT mandatory.

procedure GetPlugName (lpPlugName: PChar) stdcall ;

Get the plugin name

function OnAction (WinId : PChar ; ResourceId : integer; NodeId : PChar) : BOOL stdcall ;

Called when the user click on a button, label or menu on a WinTrace.

The plugin must call WinTrace.LinkToPlugin with CST_PLUGIN_ONACTION in order to receive this event.

WinId is the Wintrace Id

ResourceId is the User created resource or a tracetool resource. See the Tracetool resources table above

NodeId is the node id of the current selected trace (can be empty)

Return value :

when true : tracetool perform the default action

when false : tracetool don't perform any action

function OnBeforeDelete (WinId : PChar ; NodeId : PChar) : BOOL stdcall ;

Called when a node is to be deleted on a WinTrace

The plugin must call WinTrace.LinkToPlugin with CST_PLUGIN_ONBEFOREDELETE in order to receive this event.

WinId is the WinTrace Id

NodeId is the node id of the current selected trace

Return value :

when true : TraceTool delete the node

when false : TraceTool don't delete the node

procedure OnTimer() stdcall ;

Called every 500 ms. Can be used for example to refresh labels

The plugin must call LinkToPlugin with CST_PLUGIN_ONTIMER in order to receive this event

procedure Start() stdcall ;

Initialise the plugin. Called after TraceTool load the plugin

procedure Stop() stdcall ;

Stop the plugin (free any resources before unloading)

Installation.

Unlike Dot Net and Java plugins, non managed plugins don't need specific loader.

Your plugin can be at any location. Use the TraceTool Option dialog to add plugin or change the TraceTool XML configuration file.

Sample plugin.

The “SamplePlugin” folder , in the CPP distribution , contain a short C++ sample plugin.

Here are some functionalities demonstrate in the DLL:

- The plugin add a label, a button and an item menu to a WinTrace object and receive user events.
- Some nodes are protected by the plugin : no suppression is allowed.
- User Actions are added under the ‘Action’ trace node.
- The ‘Close’ action is not permitted by the plugin
- The last deleted node Id is displayed.
- The plugin receive the Timer event and display the time on a specific node and on a custom Label resource.

The ‘CppNetPlugin’ project in the Dot net distribution is a mixed DLL that implement plugin as a classic DLL but use the managed TraceTool library to communicate with the viewer.