Technical Note

Using the REP++ Trace System

Author: R&D Department

Publication date: December 4, 2006

Revision date: May 2010

Using the REP++ Trace System

Overview

REP++*studio* comes with a trace system that allows a user to follow the execution of a program that uses the REP++ class library. The REP++ class library contains configurable, built-in trace messages divided into multiple categories. The trace messages can be configured using the REP++ Trace Configuration Panel. The REP++ Trace Configuration Panel allows a user to select which traces to activate and how the trace messages will be presented (file, monitor, etc.). For example, the user can choose to view the trace of the **Execution** method of the *Database* trace category on the monitor. In this case, a message will be displayed each time REP++ is called to execute a SQL command.

REP++*studio* also includes a program that displays the REP++ trace messages, the REP++ Trace Monitor. Furthermore, it is also possible for a user to add custom traces programmatically using the REP++ **Trace** object.

This article describes how to use the REP++ Trace system and take advantage of its flexible, configurable trace messages to record or view various aspects of the execution of an application.

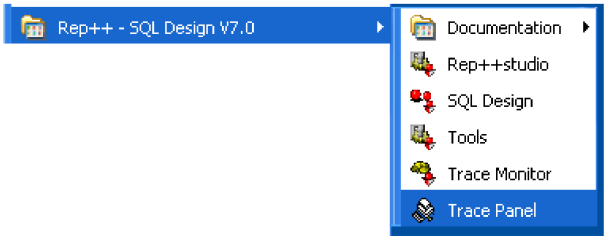
Note: code samples are given in C#.

The REP++ Trace Configuration Panel

REP++ Trace Configuration Panel is used to set up the REP++ traces. These traces are grouped by categories and subcategories, and are displayed as a tree. To configure one or several traces, you simply select the related nodes in the tree view and specify the state and destination of the trace messages. When the node selected is a trace category or subcategory, the configuration entered by the user applies for the all the traces they contain.

Launching the REP++ Trace Configuration Panel

To launch REP++Trace Configuration Panel, you can use the shortcut installed with REP++ in the Windows® Start menu:



You can also launch the REP++Trace Configuration Panel from the command prompt. The name of the executable file is *sdpanel.exe*, which is located in the *Rep++InstallDir*/Bin/Sdwin32 directory.

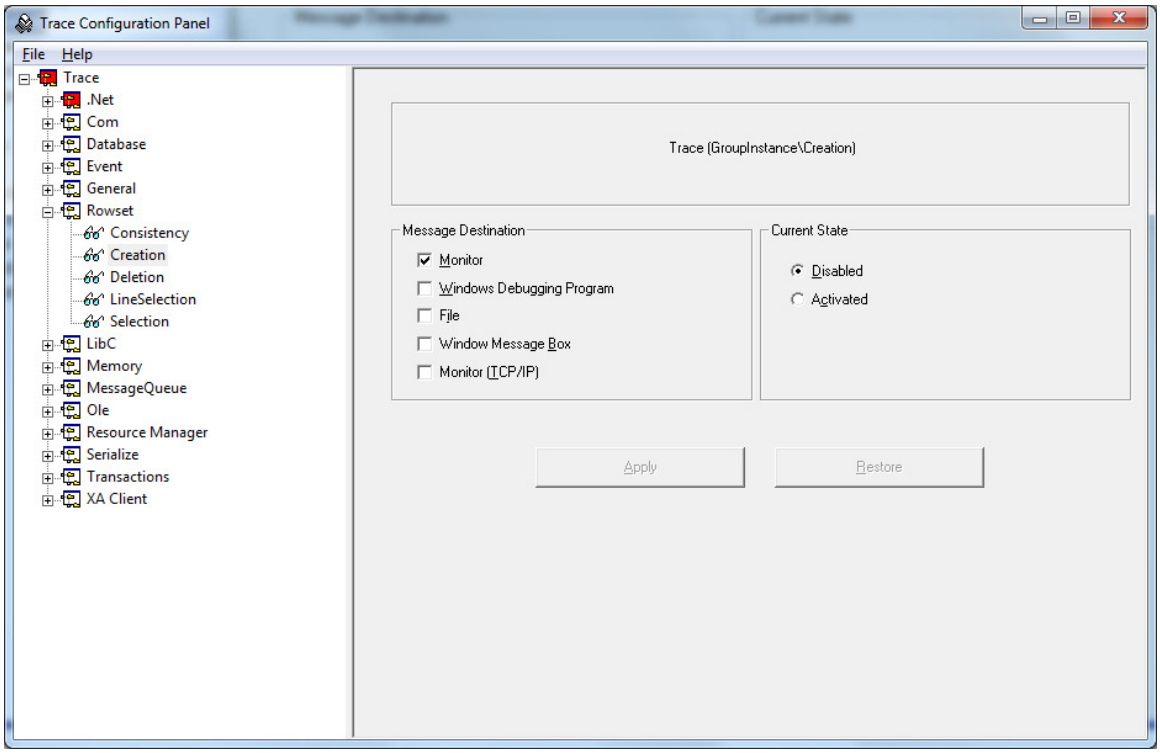


Figure 1. The Trace Configuration Panel. The categories can be expanded to show the subcategories and traces that can be activated or disabled. The destination of the trace messages can also be selected.

REP++ Trace categories

The REP++ Trace categories appearing in the configuration panel are:

- .Net: The traces for the REP++ .Net layer.
- Com: The traces for the REP++ COM layer.
- Database: The traces for the database access.
- Event: The traces for the REP++ events.
- General: General traces like the debugging trace.
- Rowset: The traces for the REP++ Rowset object.
- LibC: The traces for C++ REP++ library.
- Memory: The traces for the memory access.
- Ole: Traces used only for SQL design.
- Passerelle SNA: The traces for REP++ integration server.
- Resource Manager: Traces related to the two-phase commit.
- Serialize: Traces related to the serialization of REP++ **Field** objects using XML.
- Transactions: Traces related to the two-phase commit.
- XA Client: Traces related to the two-phase commit.

The most used trace categories for REP++ .Net users are:

- .Net
- Database
- Event
- General/Debugging Trace
- Rowset
- LibC.

Configuring traces

Global Configuration

You can set the global configuration by selecting the root node of the tree view. You also can set the header of all the displayed messages. This header can contain one or more of the following:

- Complete Name: The complete name of the trace.
- Date: The date of the message.
- Hour: The time of the message.
- Tick Count: The time of the message in tick format.

If you choose to display trace messages in a file, you can specify a file name. You can also specify the IP address and port number of a remote machine if you choose to generate trace messages on a remote machine.

Trace configuration

To configure a trace, you have to select it in the tree view and set its state and the destination of the messages. There are three states for a trace:

- ***Disabled***. The trace is disabled and the messages related to this trace will not be displayed.
- ***Activated***. The trace is activated and the messages related to this trace will be displayed.
- ***Detail***. The trace is activated and all the messages related to this trace will be displayed in more details.

You can specify one or more destinations for an active trace.

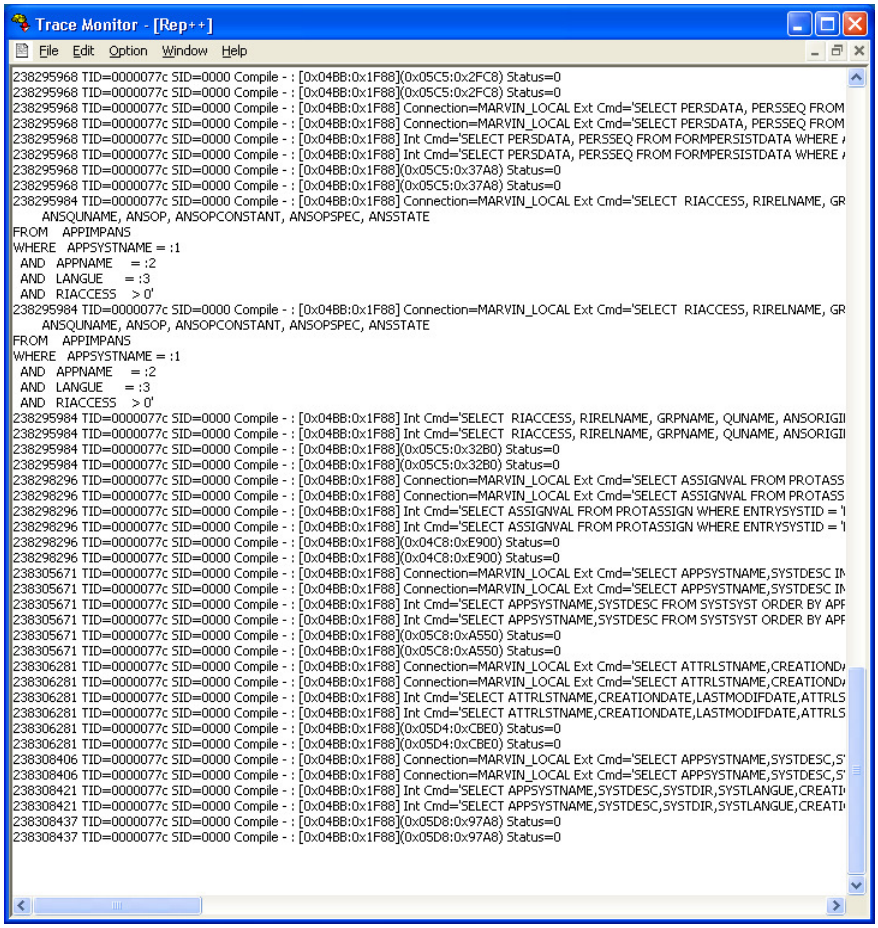
- REP++Trace Monitor.
- Windows Debugging Program. The trace messages will be displayed in a Visual Studio® output window.
- File. The trace messages will be displayed in the file specified by the global configuration.
- Window message box.
- TCP/IP Monitor. The trace messages will be displayed in a REP++ Trace monitor executed on a remote machine. The IP address of the machine and the port number of REP++ Trace Monitor are defined in the global configuration.

The REP++ Trace Monitor

REP++ Trace Monitor displays the trace messages for the activated REP++ traces. Once displayed, these messages can be saved in a file.

To launch the REP++ Trace Monitor you can use the shortcut installed with REP++ in the Windows® Start menu. You can also launch the REP++Trace Monitor from the command prompt. The name of the executable file is *sdmon.exe*, which is located in the *REP++InstallDir*/Bin/Sdwin32 directory (or *REP++InstallDir*/Bin/Sdwin64 in 64bits).

REP++ Trace Monitor can be also launched through REP++ Trace Configuration Panel. Simply select the root node of the tree view (*Trace*) and click **Start Trace Monitor**.



The trace messages displayed in this example are formatted as follow:

- Tick Count: The time of the message as a tick.
- TID: The thread identification.
- SID: The session identification. Helpful only for Web applications.
- Trace name.
- Message text.

Using the REP++ Trace system

The examples presented in this section use the DEMO system that is provided with the REP++ user training or reference guides. Make sure that you do not use the **Delayed Reading** option for each RowsetTreeDefNode of the CLIENT RowsetTreeDef.

Displaying the SQL commands

Suppose you want to display the SQL commands executed when you use the CLIENT RowsetTreeDef to read the data from the database into a RowsetTree:

```
RepPP.Application   app;
RepPP.RowsetTreeDef rstDef;
RepPP.RowsetTree    rsTree;

app = RepPP.Application.CreateFromRes();
rstDef = app.RowsetTreeDefs["CLIENT"];
```

```
rstDef.BuildSqlCommand();
rsTree = rstDef.RowsetTrees.Add();
rsTree.ReadFromDb();

. . .

rsTree.Dispose();
app.Release();
```

To visualize the SQL commands executed in this case, you have to activate the Database/Execute trace:

1. Launch the REP++ Trace Configuration Panel.
2. Expand the *Database* trace category and select the *Execute* trace in the tree view.
3. Activate this trace by clicking the **Activated** option.
4. Select the *Monitor* check box to display the messages related to this trace in REP++ Trace Monitor.
5. Click **Apply**.
6. Select the root node of the tree view (*Trace*) and launch the REP++ Trace Monitor by clicking **Start Trace Monitor**.
7. Run your program and check the messages displayed in REP++ Trace Monitor.

```
241744046 TID=00000aec SID=0000 Execute – : (0x0434:0x9AE0) Int Cmd='SELECT
CLIENTCODE,CLIENTFIRSTNAME,CLIENTLASTNAME,CLIENTTYPE,CLIENTSALESTODATE,CIECODE,CREATIONDA
TE,MODIFICATIONDATE,DESCRIPTION FROM DEMO_CLIENT ORDER BY CLIENTCODE'
241744046 TID=00000aec SID=0000 Execute – : (0x0434:0x9AE0) Int Cmd='SELECT
CLIENTCODE,CLIENTFIRSTNAME,CLIENTLASTNAME,CLIENTTYPE,CLIENTSALESTODATE,CIECODE,CREATIONDA
TE,MODIFICATIONDATE,DESCRIPTION FROM DEMO_CLIENT ORDER BY CLIENTCODE'
241744046 TID=00000aec SID=0000 Execute – : (0x0434:0x9AE0) Row Count=-1 Status=0
241744046 TID=00000aec SID=0000 Execute – : (0x0434:0x9AE0) Row Count=-1 Status=0
241744062 TID=00000aec SID=0000 Execute – : (0x0435:0xA2B0) Int Cmd='SELECT
ADDRESSCODE,ADDRESS_LINE1,ADDRESS_LINE2,CITY,POSTALCODE,PROVINCE FROM DEMO_ADDRESS WHERE
CLIENTCODE=? ORDER BY ADDRESSCODE'
241744062 TID=00000aec SID=0000 Execute – : (0x0435:0xA2B0) Int Cmd='SELECT
ADDRESSCODE,ADDRESS_LINE1,ADDRESS_LINE2,CITY,POSTALCODE,PROVINCE FROM DEMO_ADDRESS WHERE
CLIENTCODE=? ORDER BY ADDRESSCODE'
241744062 TID=00000aec SID=0000 Execute – : (0x0435:0xA2B0) Row Count=-1 Status=0
241744062 TID=00000aec SID=0000 Execute – : (0x0435:0xA2B0) Row Count=-1 Status=0
241744062 TID=00000aec SID=0000 Execute – : (0x0427:0x6D60) Int Cmd='SELECT
PHONECODE,PHONETYPE,PHONENUMBER FROM DEMO_PHONE WHERE CLIENTCODE=? AND ADDRESSCODE=?
ORDER BY PHONECODE'
241744062 TID=00000aec SID=0000 Execute – : (0x0427:0x6D60) Int Cmd='SELECT
PHONECODE,PHONETYPE,PHONENUMBER FROM DEMO_PHONE WHERE CLIENTCODE=? AND ADDRESSCODE=?
ORDER BY PHONECODE'
241744062 TID=00000aec SID=0000 Execute – : (0x0427:0x6D60) Row Count=-1 Status=0
241744062 TID=00000aec SID=0000 Execute – : (0x0427:0x6D60) Row Count=-1 Status=0
241744078 TID=00000aec SID=0000 Execute – : (0x0427:0x8790) Int Cmd='SELECT
PHONECODE,PHONETYPE,PHONENUMBER FROM DEMO_PHONE WHERE CLIENTCODE=? AND ADDRESSCODE=?
ORDER BY PHONECODE'
241744078 TID=00000aec SID=0000 Execute – : (0x0427:0x8790) Int Cmd='SELECT
PHONECODE,PHONETYPE,PHONENUMBER FROM DEMO_PHONE WHERE CLIENTCODE=? AND ADDRESSCODE=?
ORDER BY PHONECODE'
241744078 TID=00000aec SID=0000 Execute – : (0x0427:0x8790) Row Count=-1 Status=0
241744078 TID=00000aec SID=0000 Execute – : (0x0427:0x8790) Row Count=-1 Status=0
241744078 TID=00000aec SID=0000 Execute – : (0x0427:0xB938) Int Cmd='SELECT
ADDRESSCODE,ADDRESS_LINE1,ADDRESS_LINE2,CITY,POSTALCODE,PROVINCE FROM DEMO_ADDRESS WHERE
CLIENTCODE=? ORDER BY ADDRESSCODE'
241744078 TID=00000aec SID=0000 Execute – : (0x0427:0xB938) Int Cmd='SELECT
ADDRESSCODE,ADDRESS_LINE1,ADDRESS_LINE2,CITY,POSTALCODE,PROVINCE FROM DEMO_ADDRESS WHERE
CLIENTCODE=? ORDER BY ADDRESSCODE'
241744078 TID=00000aec SID=0000 Execute – : (0x0427:0xB938) Row Count=-1 Status=0
241744078 TID=00000aec SID=0000 Execute – : (0x0427:0xB938) Row Count=-1 Status=0
```

```
241744093 TID=00000aec SID=0000 Execute - : (0x0427:0xD398) Int Cmd='SELECT
PHONECODE,PHONETYPE,PHONENUMBER FROM DEMO_PHONE WHERE CLIENTCODE=? AND ADDRESSCODE=?
ORDER BY PHONECODE'
241744093 TID=00000aec SID=0000 Execute - : (0x0427:0xD398) Int Cmd='SELECT
PHONECODE,PHONETYPE,PHONENUMBER FROM DEMO_PHONE WHERE CLIENTCODE=? AND ADDRESSCODE=?
ORDER BY PHONECODE'
241744093 TID=00000aec SID=0000 Execute - : (0x0427:0xD398) Row Count=-1 Status=0
241744093 TID=00000aec SID=0000 Execute - : (0x0427:0xD398) Row Count=-1 Status=0
```

The generated trace contains all the SQL SELECT commands executed when reading the client data from the database, the addresses of each client and the phone numbers for each address. With each SQL command displayed, the trace also contains the number of records returned by the execution of this SQL command (row count, which depends the DBMS and is affected by Insert, Update or Delete SQL command).

Displaying the SQL commands' binding values

In the last example, the SQL command used to load the addresses of a client is a parameterized query. The parameter of this query is CLIENTCODE.

```
Execute - : (0x0435:0xA2B0) Int Cmd='SELECT
ADDRESSCODE,ADDRESS_LINE1,ADDRESS_LINE2,CITY,POSTALCODE,PROVINCE FROM DEMO_ADDRESS WHERE
CLIENTCODE=? ORDER BY ADDRESSCODE'
```

To display the binding values of this query, you must activate the *Bind* trace from the *Database* category:

1. Use the REP++ Trace Configuration Panel to activate the *Bind* trace under the *Database* category.
2. Launch the REP++ Trace Monitor.
3. Run your program.

Note that trace messages are now generated for the binding of the client addresses' SELECT command. These messages show the name of the binding variable and the substitution value of this variable.

```
248613015 TID=00000254 SID=0000 Bind - : (0x0435:0xA2B0) Variable='CLIENT.CLIENTCODE'
Type=10 Size=0 Value='1'
248613015 TID=00000254 SID=0000 Bind - : (0x0435:0xA2B0) Variable='CLIENT.CLIENTCODE'
Type=10 Size=0 Value='1'
248613015 TID=00000254 SID=0000 Bind - : (0x0435:0xA2B0) Variable='CLIENT.CLIENTCODE'
Status=0
248613015 TID=00000254 SID=0000 Bind - : (0x0435:0xA2B0) Variable='CLIENT.CLIENTCODE'
Status=0
```

Tracing the creation and deletion of Rowsets

When you call the **RowsetTree.ReadFromDB** method to read data from the database, the Rowsets of the RowsetTree are automatically created. Moreover, when you call the **RowsetTree.Dispose** method, these Rowsets are automatically deleted. You will now trace the creation and deletion of the Rowsets.

1. Disable all the traces activated during the previous examples by selecting the root node (*Trace*) and clicking **Disable All Traces** in the REP++ Configuration Panel.
2. Activate the *Creation* and *Deletion* traces from the Rowset trace category.
3. Run your program.

The trace generated is shown below.

```
251743015 TID=00000778 SID=0000 Creation - : Instance=0x0428:0x5138 Father = 0x0000:0x0000
       CLIENT  horizontal     CLIENT
251743078 TID=00000778 SID=0000 Creation - : Instance=0x0428:0x8B10 Father = 0x0428:0x5138
       ADDRESS horizontal     CLIENT
251743078 TID=00000778 SID=0000 Creation - : Instance=0x0427:0x5630 Father = 0x0428:0x8B10
       PHONE   horizontal     CLIENT
251743093 TID=00000778 SID=0000 Creation - : Instance=0x0427:0x6D60 Father = 0x0428:0x8B10
       PHONE   horizontal     CLIENT
251743093 TID=00000778 SID=0000 Creation - : Instance=0x0435:0xA2B0 Father = 0x0428:0x5138
       ADDRESS horizontal     CLIENT
251743093 TID=00000778 SID=0000 Creation - : Instance=0x0427:0xC210 Father = 0x0435:0xA2B0
       PHONE   horizontal     CLIENT
251743125 TID=00000778 SID=0000 Deletion - : Instance = 0x0427:0x5630
       Father=0x0428:0x8B10   PHONE   CLIENT  fRetVal=0
251743125 TID=00000778 SID=0000 Deletion - : Instance = 0x0427:0x6D60
       Father=0x0428:0x8B10   PHONE   CLIENT  fRetVal=0
251743125 TID=00000778 SID=0000 Deletion - : Instance = 0x0428:0x8B10
       Father=0x0428:0x5138   ADDRESS CLIENT  fRetVal=0
251743125 TID=00000778 SID=0000 Deletion - : Instance = 0x0427:0xC210
       Father=0x0435:0xA2B0   PHONE   CLIENT  fRetVal=0
251743125 TID=00000778 SID=0000 Deletion - : Instance = 0x0435:0xA2B0
       Father=0x0428:0x5138   ADDRESS CLIENT  fRetVal=0
251743125 TID=00000778 SID=0000 Deletion - : Instance = 0x0428:0x5138
       Father=0x0000:0x0000   CLIENT  CLIENT  fRetVal=0
```

The trace shows all the Rowsets created or deleted during the execution of the program. For each Rowset, its memory address, the memory address of its father Rowset and its RowsetDef name are displayed.

Tracing the lifetime of the REP++ Application object

To trace the lifetime of the REP++ **Application** object in a .Net application, you need to:

1. Disable all the traces activated during the previous examples.
2. Activate the AppLifeTime trace from the .Net trace category.
3. Run your program.

The trace generated shows that a REP++ **Application** object is created, initialized and disposed of.

```
250114046 TID=00000fa8 SID=0000 AppLifeTime - : [010fe528] Wrapper Created
250114515 TID=00000fa8 SID=0000 AppLifeTime - : [010fe528] CreateFromRes - New object -
Sys='DEMO' Prg='DEMOCLIENT' Conn='MARVIN_LOCAL' User='$SUPERUSER' eLang='0'
Method='Normal' Design='false'
250114515 TID=00000fa8 SID=0000 AppLifeTime - : [010fe528] CreateFromRes - Ref Count = 1
Design = false
250114593 TID=00000fa8 SID=0000 AppLifeTime - : [010fe528] Disposing.
```

Defining your own traces

REP++ allows you to create your own traces programmatically. You can add messages to an existing trace or create your own trace and add messages to it. The example code used in this article was modified to create a new trace and add messages before each instruction in the code.

```
RepPP.Application     app;
RepPP.RowsetTreeDef   rstDef;
RepPP.RowsetTree      rsTree;
RepPP.Trace           trace;

app          = RepPP.Application.CreateFromRes();
```

```
trace        = app.Traces.Create("MyTrace", RepPP.TrcType.sdTrcTypeThreeState);
trace.State  = RepPP.TrcState.sdTrcStateGeneral;
trace.Output = RepPP.TrcOutput.sdTrcOutputMonitor;
trace.SetTrace(RepPP.TrcDisplay.sdTrcDispGeneral, "Get the CLIENT RowsetTreeDef
object\n");
rstDef = app.RowsetTreeDefs["CLIENT"];
trace.SetTrace(RepPP.TrcDisplay.sdTrcDispGeneral, "Build the RowsetTreeDef SQL
commands\n");
rstDef.BuildSqlCommand();
trace.SetTrace(RepPP.TrcDisplay.sdTrcDispGeneral, "Create a new CLIENT RowsetTree\n");
rsTree = rstDef.RowsetTrees.Add();
trace.SetTrace(RepPP.TrcDisplay.sdTrcDispGeneral, "Read data from the database\n");
rsTree.ReadFromDb();

. . .

trace.SetTrace(RepPP.TrcDisplay.sdTrcDispGeneral, "Dispose of the CLIENT RowsetTree\n");
rsTree.Dispose();
trace.SetTrace(RepPP.TrcDisplay.sdTrcDispGeneral, "Dispose of the Rep++ Application
object\n");
app.Release();
```

The execution of this program generates the following trace in the REP++Trace Monitor.

```
254754625 TID=00000a2c SID=0000 MyTrace - : Get the CLIENT RowsetTreeDef object
254754656 TID=00000a2c SID=0000 MyTrace - : Build the RowsetTreeDef SQL commands
254754656 TID=00000a2c SID=0000 MyTrace - : Create a new CLIENT RowsetTree
254754687 TID=00000a2c SID=0000 MyTrace - : Read data from the database
254754734 TID=00000a2c SID=0000 MyTrace - : Dispose of the CLIENT RowsetTree
254754750 TID=00000a2c SID=0000 MyTrace - : Dispose of the Rep++ Application object
```