LightCycler® 480 Instrument

Addendum 2 to Operator's Manual, Version 3.0

Software Version 1.5 August 2012

Addendum 2 to LightCycler® 480 Gene Scanning Software Guide, Version 2.0

Software Version 1.5 August 2012

Addendum 1 to LightCycler® 480 LIMS Interface Module Guide, Version 2.0

Software Version 1.5 August 2012

Addendum 1 to LightCycler® 480 LIMS Module SP1 Reference Manual, Version 03

Software Version 1.5 August 2012

Information regarding LightCycler® 480 Instrument

Dear Valued User of the LightCycler® 480 Instrument,

Roche Diagnostics Ltd. has merged into Roche Diagnostics International Ltd and therefore the company name has changed to

Roche Diagnostics International Ltd

If you have any further questions regarding this matter, please do not hesitate to contact your Roche Diagnostics representative. To call, write, fax, or email us, visit the Roche Applied Science home page, http://www.roche-applied-science.com and select your home country. Country-specific contact information will be displayed.

The name of the legal manufacturer in section "Prologue (Introduction)/Contact Addresses" changes as follows:

Old adress	New adress
Roche Diagnostics Ltd. Forrenstrasse CH-6343 Rotkreuz Switzerland	Roche Diagnostics International Ltd Forrenstrasse 2 6343 Rotkreuz Switzerland

06845053001 ① 0812

Roche Applied Science

LightCycler®480
LIMS Module 1.5 sp1
programming reference manual

rev. 03.

Roche Applied Science

LightCycler®480 LIMS Module 1.5 sp1 programming reference manual

---

LIMS Software Version 1.5 sp1 Manual rev.03.

This is the paper version of the LIMS manual for the Roche LightCycler 480

.

An electronic version in the HTML format is also maintained and available from the installation CD delivered with the instruments, or available on request to a Roche representative.

Table of Contents

1 Introduction

1.1 Target audience

The intended audience for this manual is programmers possessing a good knowledge with Windows API programming.

The present manual will help such programmers to understand the basics of the LightCycler 480 LIMS interface and build efficient applications to control the instrument and collect the results of experiments.

Code examples are presented in this manual, to assist in the learning process. The functionality of those examples has been tested, however, as they have been edited to accommodate the manual constraints, there is no guaranty that they will directly work by simply using a "cut and paste" process into a practical application. In some instances, a slight adaptation might be required.

1.2 Document Revision History

Document Version	Revision Date
1.0	January 2008
1.01 (LIMS 1.5 Manual rev. 01)	October 2008
1.02 (LIMS 1.5 Manual rev. 02)	March 2009
1.03 (LIMS 1.5 sp1 Manual rev. 03)	November 2009

Changes in rev. 02 apply to the manual and not the LIMS module software.

Added Note about Active Database in General information 13
Added Note about LIMS Activity timeout in General information 13
Added info about the delay between Login and Reserve Instrument
Added info about life of communication objects: Know issues 23, ILIMSConnection 27
Added a timing flowchart 18 - please read this most important information
Added some keywords in the Index

Changes in rev. 03 apply to manual and **LIMS 1.5 SP1**.

Added explanation about the Macros and Experiment names 51. See Also Other Issues 25
Added important information and a flow chart. 18

mechanical, for any purpose, without the express written permission of Roche Diagnostics GmbH.

Questions or comments regarding the contents of this manual can be directed to the address below or to your Roche representative.

Roche Diagnostics Ltd.
Roche Applied Science
Global Service Support
Forrenstrasse
6343 Rotkreuz, Switzerland

Every effort has been made to ensure that all the information contained in this manual is correct at the time of printing.
However, Roche Diagnostics Ltd. reserves the right to make any changes necessary without notice as part of ongoing product development.

1.3 Contacts, Trademarks

Contact Addresses

	Manufacturer	Roche Diagnostics Ltd. Forrenstrasse CH-6343 Rotkreuz Switzerland
	Distribution	Roche Diagnostics GmbH Sandhofer Straße 116 D-68305 Mannheim Germany
	Distribution in the US	Roche Diagnostics 9115 Hague Road PO Box 50457 Indianapolis, IN 46250 USA

Trademarks

LIGHTCYCLER and LC are trademarks of Roche.

Other brands or product names are trademarks of their respective holders.

1.4 Glossary

EXOR	The database used by the LightCycler software. Multiple databases can be used on one instrument. This is to be avoided when using the LIMS software. see Other known problems 23
Experiment	A run on the instrument that results into various measurements (data points) being taken (Fluorescence and temperature data). These data are stored into the database and can be further analysed using various SW analyzing modules. The experiment data can be called via the LIMS interface.
IXO file	The format used by Exor to import/export objects like experiment results or other Database entities.
Macro	A set of parameters used to run an experiment. A macro is an Exor database object. The name of a valid macro must be passed to the LIMS interface to be able to start an experiment
MWP = Micro Well Plate	The container used to hold the samples to be analysed. The plates exist in two sizes, 96 (8x12) and 384 (16x24) wells. The samples are identified with coordinates : The 96-wells plate is numbered from A1 to H12; Rows A to H , columns 1 to 12. The 384-wells plate is numbered from A1 to P24; Rows A to P, columns 1 to 24.

2 LIMSClientLib Reference

This section of help provides reference information for the API elements provided by LIMSClientLib.
The LIMSClientLib reference information can be divided into following categories.

2.1 General information and debugging

2.1.1 General information

****** Conventions used in the manual *******

In some instances, the code on one line might be too long to fit properly on the page. To prevent the line from extending beyond pages margins, it has been split in the following way:

```
Public Function Login(PW As String, Host As String, ->
Port As String) As LIMSOperationResult
```

The -> sign indicates that he next line must be joined to recreate functional code, as in :

```
Public Function Login(PW As String, Host As String, Port As String) As
LIMSOperationResult
```

Important notes:

- The LIMS Client Server communication is based on network sessions initiated from the client module. Every session is identified by a unique serial number, and if a disconnection occurs, the client should create a brand new session. Trying to communicate using an expired session number will result in a server warning: "Session is invalid 10019"
  To avoid the above mentioned problem, a very important point the programmer must take care of, is the life cycle of the communication object. All created communication objects created MUST be released in the end. See ILIMSConnection 27

- Unless otherwise noted, all LIMS API calls are **synchronous.**
  This means that the commands called will always wait until the command completes before control is returned to the calling program.

- The LIMS server expects to hear a **heartbeat** from a client every ten seconds.
  For debugging purposes, you may want to set a break point and pause the client. To avoid getting logged off:
  1. Open the file C:\Program Files\Roche\LightCycler480\Bin\LIMS.stc
  2. Find and change the line:
     <LIMS_SOCKET_SERVER_HEARTBEAT_TIMEOUT>10</
     LIMS_SOCKET_SERVER_HEARTBEAT_TIMEOUT>
  3. Replace ten with a larger value.

- There is a second level of timeout monitoring: The Server expects some form of client activity during the last 60 Minutes.
  If the client does not do anything during this period, the server will timeout and disconnect the current session. As a lot of experiments last more than 60 Minutes, it is important to keep this in mind and create some client communication to avoid this timeout. A good method is to request the instrument status periodically, or to send an dummy database request.
  Do not overload the system with too many queries, once every 30 seconds more than enough to keep the system active.

- The database being accessed by the LIMS server is the last one that was opened by the regular GUI software.
  When using LIMS, it is a good practice to have only one operational database, otherwise there might be an uncertainty regarding the currently used database.
  This information is stored in the files C:\Program files\Roche\LightCycler480 \bin\HTC1.SXC in the tag `<ActiveServer>`.

  Never change this information manually in the database ! To change the active database, login with the regular LightCycler 480 software, by selecting the desired database, open some experiment or database object, then logout and close the regular GUI.

  To check what database is active, in the HTC1.SXC look for the tag `<ActiveServer>` :

  Example:

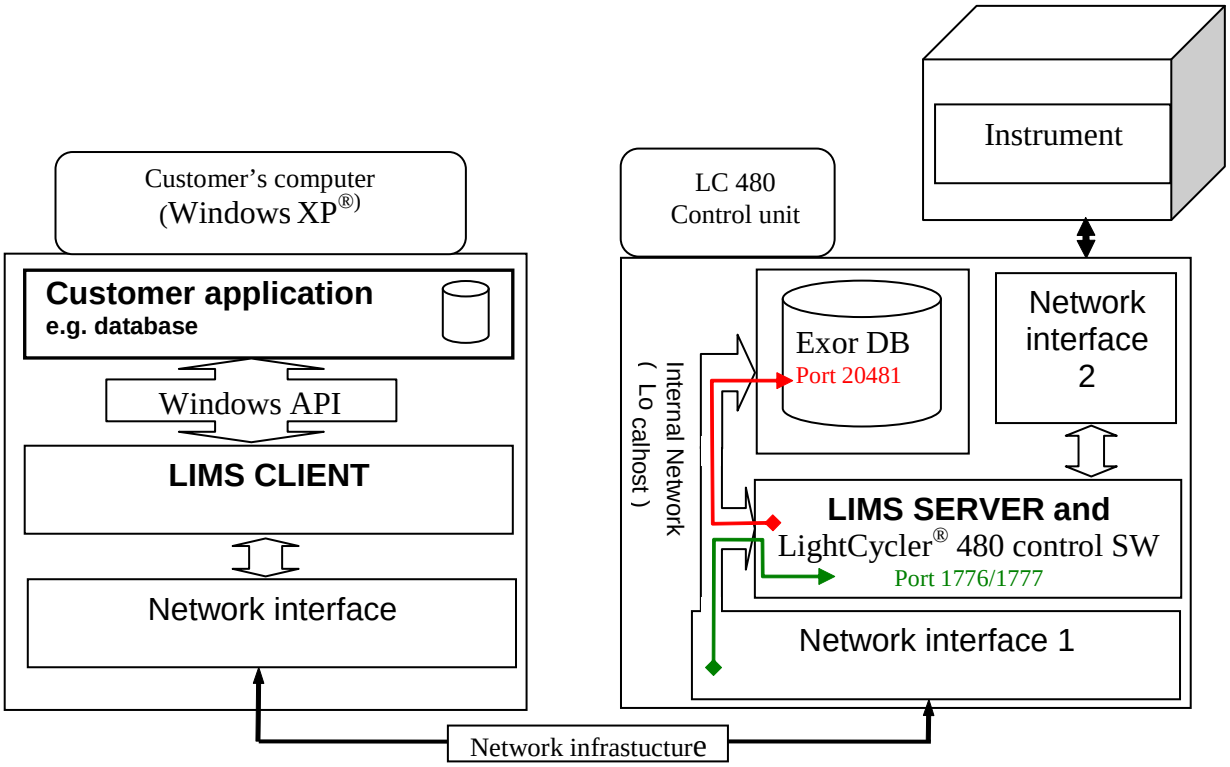
```
<KnownServers>
      <ActiveServer>My Computer</ActiveServer>
      <Server>
            <Name>My Computer</Name>
            <Location>localhost:20481</Location>
      </Server>
      <Server>
            <Name>Remote SERVER</Name>
            <Location>172.20.2.80:20481</Location>
      </Server>
</KnownServers>
```

- **Note regarding critical timing:** The information about the active instrument has to be retrieved from the database.
  Because this record may not be instantly available right after a successful login, it is recommended to have a delay of 15 seconds before attempting to reserve the instrument.
  This can be even more critical if the database is not located in the instrument control unit, but on a remote database server.The delay must be adjusted according to the server and network response time.

- If a firewall or other security device is present between the LIMS client and server, ports 1776 and 1777 must be opened to enable communication.

General architecture of the system:

Note that the Exor DB for the instrument might be installed at another location. e.g. Central server.

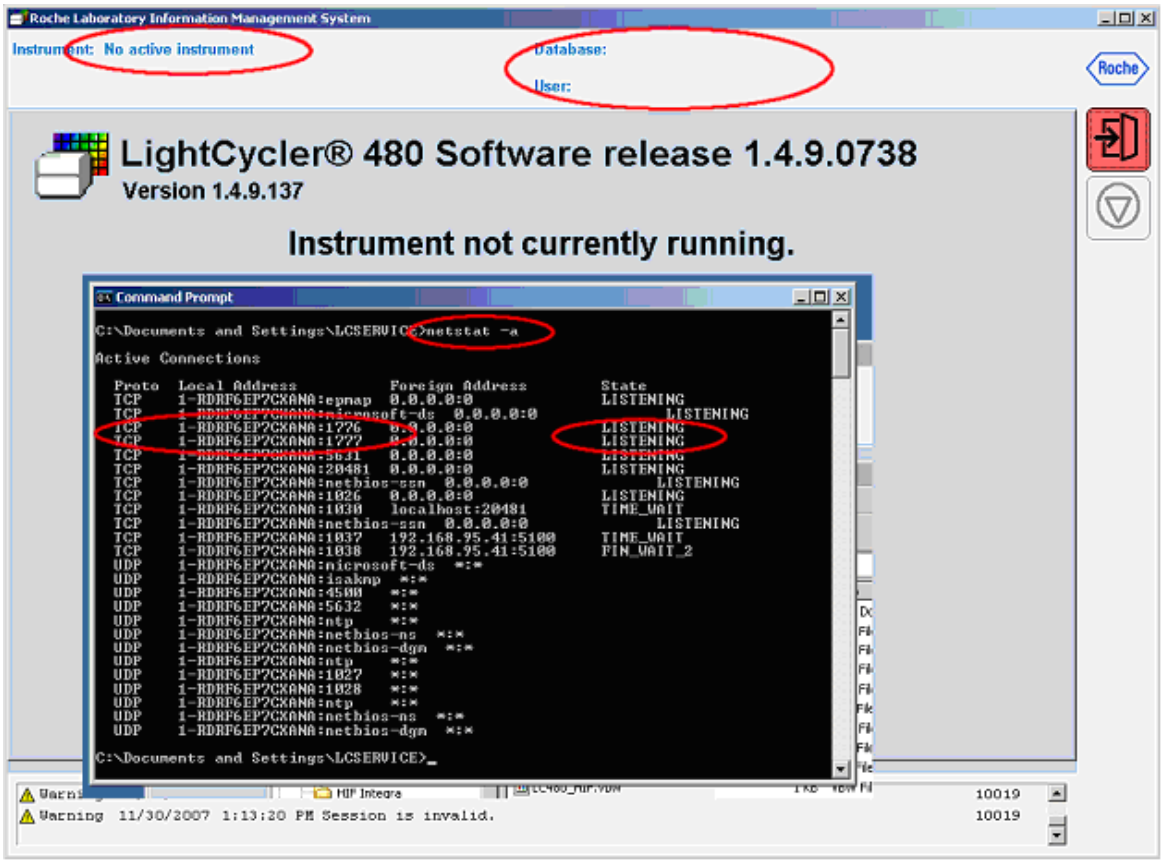


Other debugging tips

1. On the LightCycler control unit:

- Start the LIMS Application (Server)
- Open a command window and type **`netstat -a`**

The list should have two entries from the Control unit computer name with port 1776 and 1777 near the end of the line it should show “Listening”. This indicates that the LIMS server is waiting for a connection.



2. Connect from the application side.

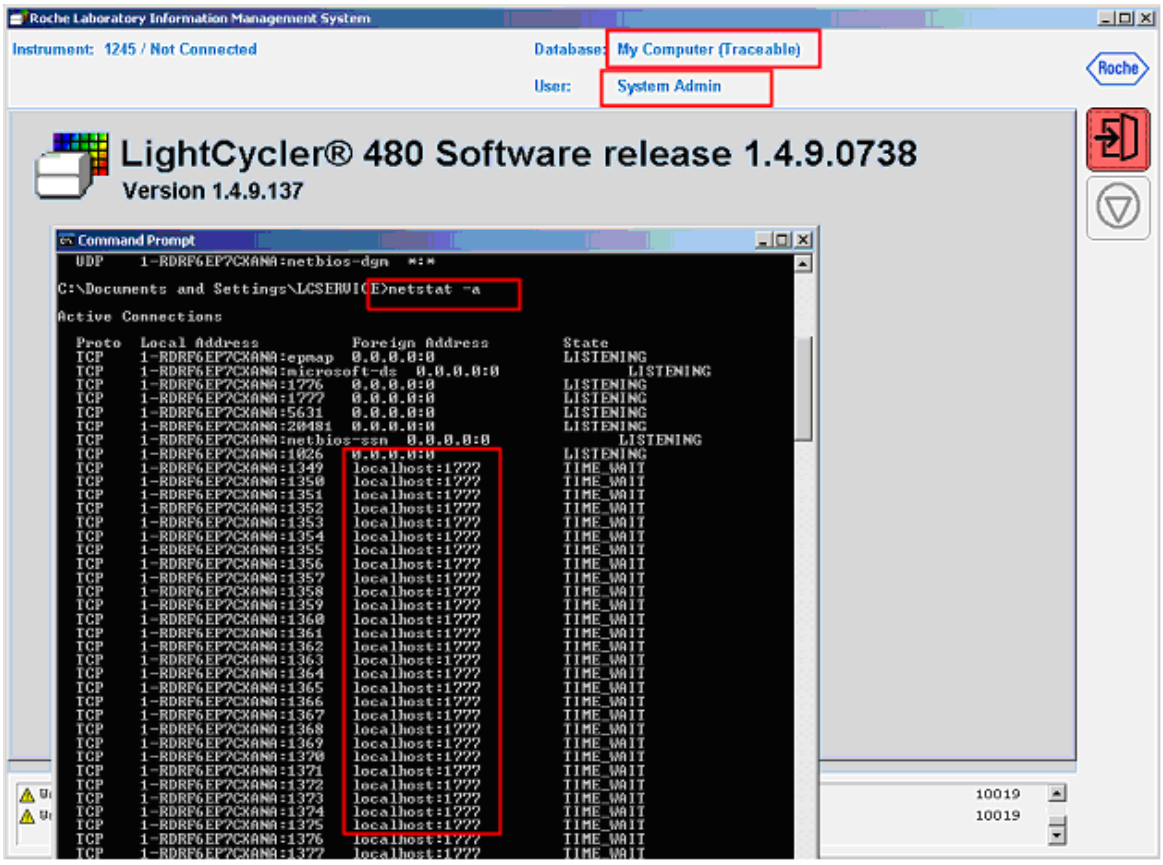
- The application should be on a separate computer running windows
- The LIMS client (Library) must be installed.

Once the application has successfully logged in and connected to the database, the following shows up on the LIMS server side:

- The database name. This is the last database that was successfully opened by the LightCycler application software.
- The user name that was used to connect.

If you retype the netstat –a command in the command window, you should probably see a lot of activity on the port 1777.

Additionally, if an active instrument has been successfully connected, it will show up on the LIMS Server screen with some status (here in the example, the default instrument was not able to connect).



2.1.2 Timing flowchart

This flowchart describes in what order operations must be arranged.
To ensure reliable function of the system, the delays must be adhered to where described.

When the instrument is in stand by, this does not mean that operations are completed. The experiment will still run as data is saved into the database and eventually all programmed analyses are done. This will be indicated by the Experiment status

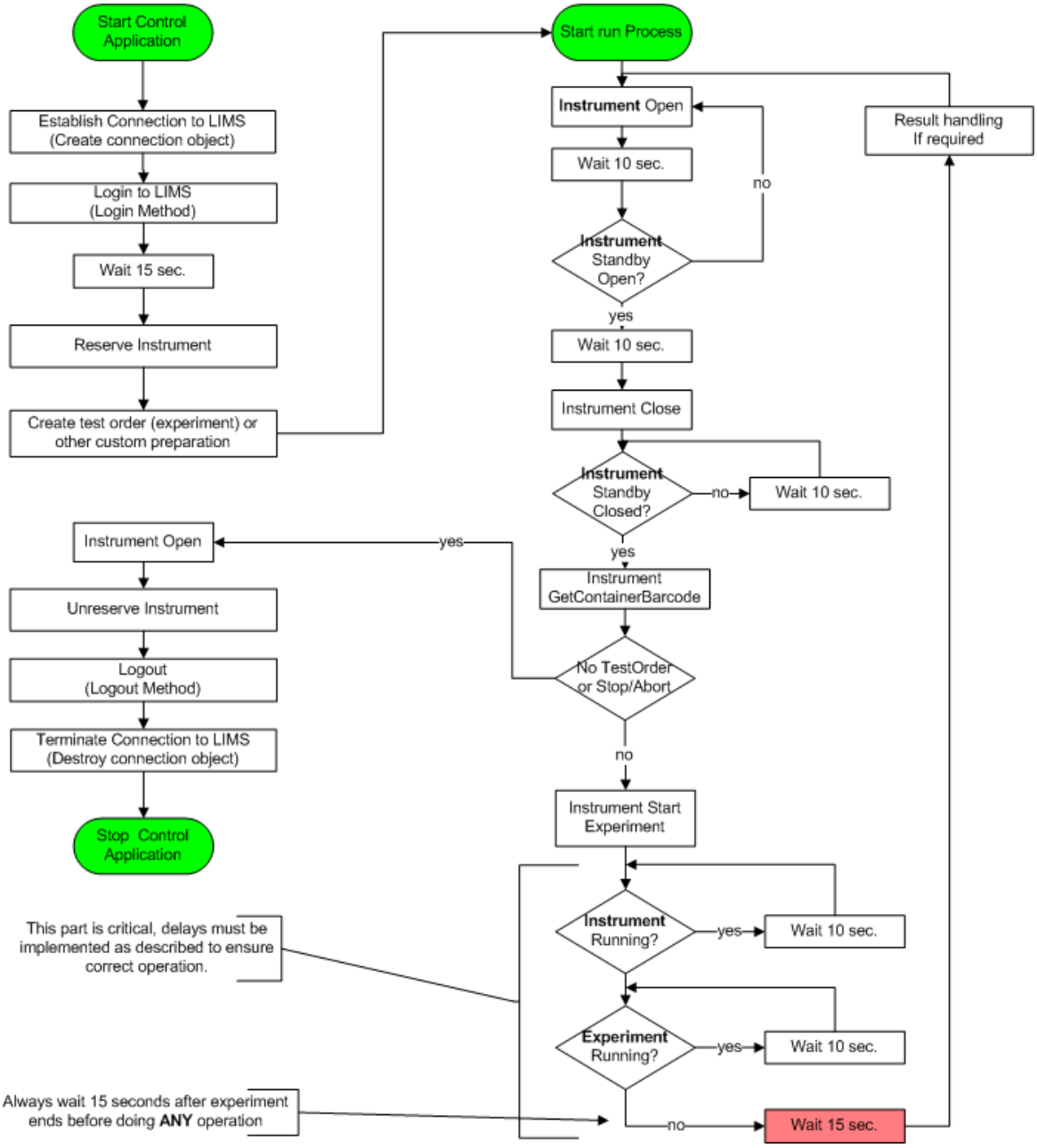
An important point to be followed: When polling the system to find out whether an experiment has been completed, the following operations are needed.

1) Poll the **instrument status.** This must be periodically repeated until the status "stand by, MWP loaded" has been reached.

2) After the instrument goes in stand by, poll the **Experiment status.** When the experiment status changes from "**running**" to "**No analyses**" or "**Has analyses**", wait at least 15seconds before next action.

3) At the end of the run process, when no more actions are required, it is important to properly disconnect the server, otherwise it will eventually timeout 14 after 60 minutes. No communication objects should be left in the background as they will leave "zombie" connections that might interfere if a new connection is created.

Refer to the flow chart next page.


Start Control Application
Establish Connection to LIMS (Create connection object)
Login to LIMS (Login Method)
Wait 15 sec.
Reserve Instrument
Create test order (experiment) or other custom preparation
Start run Process
Instrument Open
Wait 10 sec.
Instrument Standby Open?
no
yes
Wait 10 sec.
Instrument Close
Instrument Standby Closed?
no
Wait 10 sec.
yes
Instrument GetContainerBarcode
No TestOrder or Stop/Abort
yes
no
Instrument Start Experiment
Instrument Running?
yes
Wait 10 sec.
Experiment Running?
yes
Wait 10 sec.
no
Wait 15 sec.
Result handling If required
Instrument Open
Unreserve Instrument
Logout (Logout Method)
Terminate Connection to LIMS (Destroy connection object)
Stop Control Application
This part is critical, delays must be implemented as described to ensure correct operation.
Always wait 15 seconds after experiment ends before doing ANY operation

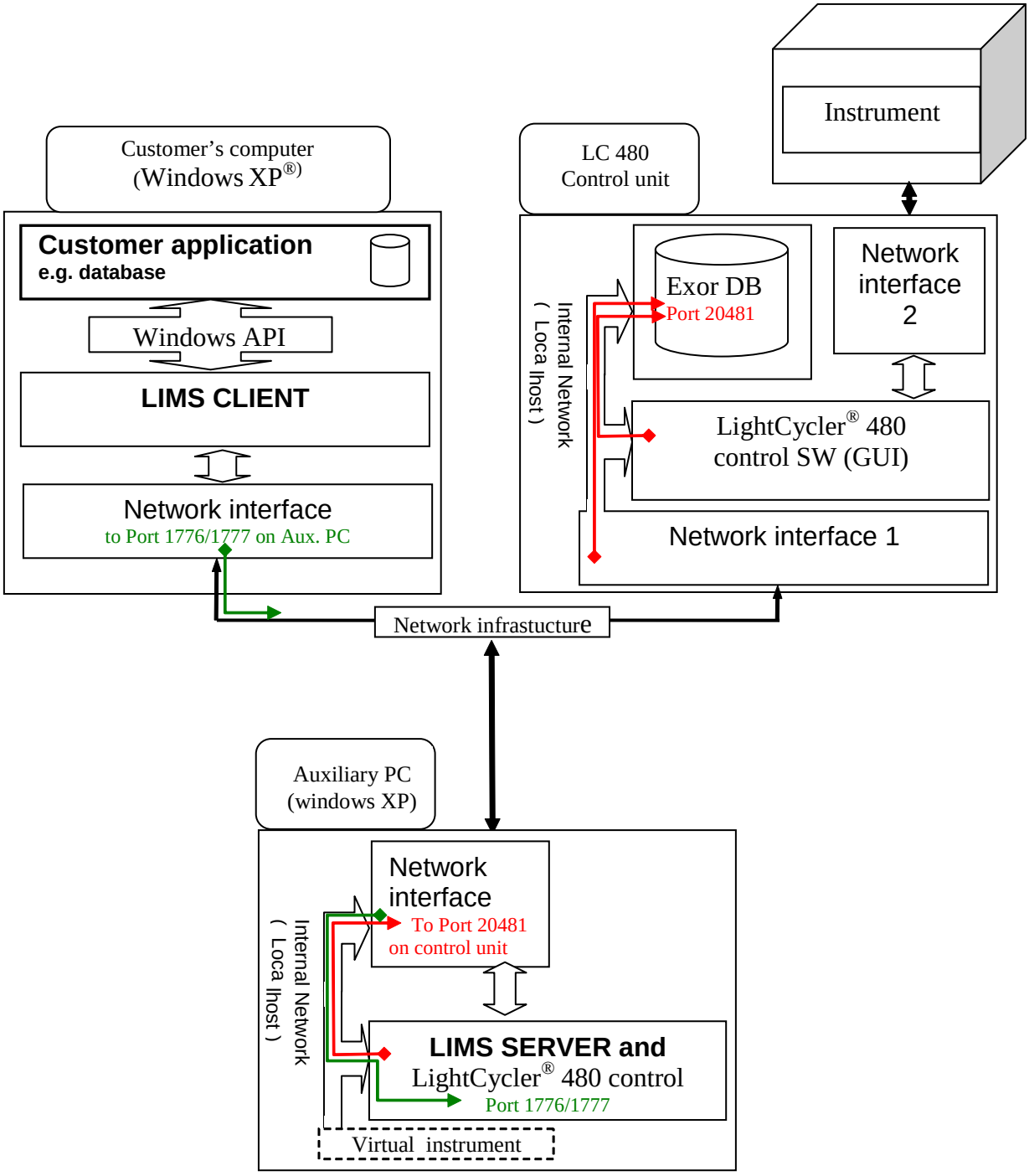
2.1.3 Install LIMS server on another PC

To do read-only LIMS development (no instrument control) without impacting the instrument PC:

1. Install LC480 software and LIMS software on a second computer. (Named Auxiliary PC on the schematic)
2. On the Auxiliary PC start LSC480 software. On the login screen, hit the options button and configure a new database that points to Exor on the instrument PC (Control unit).
3. In LCS 480 software on the Auxiliary PC, set the default instrument to be a virtual instrument.
4. On the Auxiliary PC, log out of the LCS480 software and start the LIMS server
5. Connect a LIMS client to the LIMS server running on the Auxiliary PC

The LIMS client should be able to pull production data from the Exor on the instrument PC without affecting the operation of the instrument.
In other words, lab personnel should be able to run the instrument and informatics personnel should be able to retrieve data at the same time.
Of course, any instrument calls made by the LIMS client will fail, but queries for experiment data should work fine.

See schematic next page


Instrument
Customer's computer
(Windows XP®)
LC 480
Control unit
Customer application
e.g. database
Windows API
LIMS CLIENT
Network interface
to Port 1776/1777 on Aux. PC
Internal Network
(Loca lhost)
Exor DB
Port 20481
Network
interface
2
LightCycler® 480
control SW (GUI)
Network interface 1
Network infrastucture
Auxiliary PC
(windows XP)
Network
interface
To Port 20481
on control unit
Internal Network
(Loca lhost)
LIMS SERVER and
LightCycler® 480 control
Port 1776/1777
Virtual instrument

2.1.4 Multiple LIMS users

Enabling multiple LIMS users on one server.

To enable multiple LIMS users on a single server:

1. Find the file
   C:\Program Files\Roche\LightCycler480\Bin\LIMS.stc
2. In the file, find the line
   <LIMS_SOCKET_SERVER_MAX_SESSIONS>1</
   LIMS_SOCKET_SERVER_MAX_SESSIONS>
3. Change the "1" to the number of user you want enable
   <LIMS_SOCKET_SERVER_MAX_SESSIONS>4</
LIMS_SOCKET_SERVER_MAX_SESSIONS>

Remember, only one **user** can access the instrument at a time.

2.2 Restrictions and known issues

2.2.1 Environment

The LIMS client must be installed on a computer with the following recommended characteristics:

Operating system:

Windows 2000 SP 4
Windows XP SP 2

Languages:

Although some script languages like Python or PHP might work, we have not tested them and do not support them.

VB Script is known to work only partially. Some functions work, while others don't, due to the way VBScript passes the variables to functions or methods (ByRef or ByVal).

2.2.2 Other known issues

- The Database opened by the LIMS server by default is the last database opened by the regular GUI software.
  Caution should be exercised if more than one database exist on the system.
  For this reason it is **recommended** to install **only one database** when using the LIMS module.

- It is recommended to create a separate user for the LIMS access, and not use the System Admin to run the LIMS.
  This LIMS user can be an administrator, and should be used exclusively for this purpose.
  The System Admin user should be reserved for maintenance activities of the database.

- Not releasing connection object creates unexpected problems. One know symptom is the message "Invalid Session" on the LIMS server message window. There might be other unpredictable effects.

- The Replicate Position property is deprecated in the v 1.5. It should not be used anymore.

- LIMS: Invalid floating point operation when regional settings on client PC have been changed to any other setting than English (US).
  Problem appears when starting a macro which includes an Endpoint Genotyping analysis with an external background correction via LIMS.
  After the run has finished and the analysis is being created, an error 'invalid floating point operation' appears (the analysis is not created).

Workaround: Analysis but can be created manually afterwards.

Corrected issues in version 1.5

- Windows Language settings can be other than US. English, except in one special case, see last bullet point above.
  .
- The number of sockets is strictly enforced (see Multiple LIMS users 22)
- Reservation of the instrument enforces exclusive access: If multiple sockets are enabled (see Multiple LIMS users 22) the instrument can be reserved only by one socket at a time, even if the same user credentials are used.
  When a connection has reserved the instrument, subsequent connections give only access to the database.
-

Corrected issues in version 1.5 SP1

Corrections effective in LIMS SP1 of version 1.5:

- Memory leaks on the server side have been almost completely eliminated.
  Although a few minor leaks remain, the memory consumption is low enough that the system can run at least 7 days without need for a reboot.

- Some connection issues have been eliminated, when the Server would not connect properly to the instrument.

- The application time out of 60 minutes no longer leads to continuous "Session Invalid" messages on the LIMS server and subsequent crash of the control unit. The "session invalid" message will still be issued but only for a limited period (< 1 minute).
  In case of a time out, the session will be effectively terminated after a maximum of 1 minute and warning messages will not be issued after the disconnection.
  Further attempts to communicate on the client after the timeout will results in clear error messages on the client side. The Client will be requested to log into a new session.

- Issues with language settings (non US regional settings on the client computer) have been corrected.
  This problem resulted in Run time errors when making queries from client systems with non US regional settings.

2.3 Interfaces

This section contains reference information for the COM interfaces provided by LIMSClientLib. The following interfaces are used with LIMSClientLib.

2.3.1 ILIMSConnection

Root interface for LIMS clients.

General methods and properties.		
property get	LoggedIn 29	Checks to see if you are currently logged in to the LIMS server
property get / put	Host 33	Gets the IP Address or machine name of the computer on which the LIMS server is running
property get / put	Port 34	Gets the TCP port number on which the LIMS server is running (default = 1776)
property get	Instrument 32	Returns an interface for controlling the instrument
property get	ExperimentInfo 31	Returns an interface for accessing the Experiment API
property get	Query 30	Returns an ILIMSQuery interface
method	Login 35	Establishes a session with the LIMS server if not already logged in
method	Logout 36	Terminates the session with the LIMS server

In order to use the LIMS interface, a **connection object** has to be created first. As an example, see the code fragments below:
Very important note: The connection object has to be released before exiting the program or attempting any new connection. Not releasing communication objects will leave "zombie" connections to the server in the background with some unwanted effects. I.e. messages "Invalid session" on the server.

Unused connections will be automatically terminated after a 60 minutes period of inactivity.
Code snipets see next page

[Visual Basic]

```
Public Sub CreateConnection()
    If Not Assigned(gConnection) Then
        Set gConnection = New LIMSClientLib.LIMSConnection
        MsgBox ("Connection created")
    Else
        MsgBox ("Failure TO CREATE CONNECTION")
    End If
End Sub
```

Releasing a connection object:

```
Public Sub FreeConnection()
    If Assigned(gConnection) Then
        Set gConnection = Nothing
        'MsgBox "Connection released"
    Else
        MsgBox "Connection object is not assigned"
    End If
End Sub
```

[C#]

```
private bool CreateConnection()
        {
            if (!Assigned("Object is already assigned"))
            {
                mConnection = new LIMSConnection();
                VerifyMessage(login, "Connection created");
                return true;
            }
            return false;
        }
```


2.3.1.1 LoggedIn Property

Checks to see if you are currently logged in to the LIMS server.

[C/C++]

```
HRESULT   get_LoggedIn( VARIANT_BOOL* Value );
```

[Visual Basic]

```
Public ReadOnly Property  LoggedIn As Boolean
```

Code Snipet

```
Public Function Logout() As LIMSOperationResult
On Error GoTo Error
    If gConnection.LoggedIn Then
         Set Logout = gConnection.Logout
        If Logout.Successful Then
            MsgBox ("Successfully Logged Out")
        Else
            MsgBox ("Failure to logout")
        End If
    End If
Error:
    Exit Function
End Function
```

[C#]

```
public  ref bool   ILIMSConnection.LoggedIn { get; }
```

Parameters

Value

[out,retval]

Please note that there is an application time out[14] of 60 minutes when the application does not request any action from the LIMS server.

2.3.1.2 Query Property

Returns an ILIMSQuery interface.

```
[C/C++]
HRESULT   get_Query( ILIMSQuery** Value );
```

```
[Visual Basic]
Public ReadOnly Property  Query As Object
```

VB code snipet :

```
Public Function GetQueryResult()
    Dim QIndx, Indx As Integer
    Dim OpResult As LIMSOperationResult
    Dim Qresult As LIMSQueryResult
    Dim Qdata As LIMSQueryResultData
    Dim LQuery As LIMSQuery

    If gConnection.LoggedIn = True Then
       gConnection.Query.ObjectType = "Experiment" 'set the filters
                        SEE ALSO ObjectType
        gConnection.Query.Name = "*"
        gConnection.Query.Owner = ""
        Set OpResult = gConnection.Query.ExecuteQuery(Qresult)
            If OpResult.Successful = True Then
                QIndx = Qresult.Count 'Get the number of items returned
                If QIndx > 0 Then
                    Main.ExpList.Clear  ' Delete the list in main frame
                    For Indx = 0 To QIndx - 1
            'Read Individual items and Add them to the list
                        Set Qdata = Qresult.GetResultData(Indx)
                        Main.ExpList.AddItem (Qdata.Name), Indx
                    Next Indx
                End If
            Else
                MsgBox "Exp Query unsucessful .."
            End If
         End If
End Function
```

```
[C#]
public  object   ILIMSConnection.Query { get; }
```

Parameters

Value

[out,retval]

2.3.1.3 ExperimentInfo Property

Returns an interface for accessing the Experiment API.

```
[C/C++]
HRESULT   get_ExperimentInfo( ILIMSExperimentInfo** Value );
```

```
[Visual Basic]
Public ReadOnly Property  ExperimentInfo As Object
```

VB code snipet :

```
Public Function GetExperimentStatus(ExperimentName As String) ->
As LIMSOperationResult
    Dim EStatus As String
    EStatus = ""
    If Assigned(gConnection) Then
Set GetExperimentStatus = gConnection.ExperimentInfo.GetStatus ->
(ExperimentName, EStatus)
        If GetExperimentStatus.Successful Then
            MsgBox ( "Status of experiment : " & EStatus)
        Else
                MsgBox ("No Status found")
        End If
    Else
        MsgBox "Global LIMS Proxy object is not assigned"
    End If
End Function
```

```
[C#]
public  object   ILIMSConnection.ExperimentInfo { get; }
```

Parameters

Value

[out,retval]

2.3.1.4 Instrument Property

Returns an interface for controlling the instrument.

```
[C/C++]
HRESULT   get_Instrument( ILIMSInstrument** Value );
```

```
[Visual Basic]
Public ReadOnly Property  Instrument As Object
```

VB code snipet :

```
Public Function ReserveInstrument() As LIMSOperationResult
 If Assigned(gConnection) Then
        Set ReserveInstrument = gConnection.Instrument.Reserve
        If ReserveInstrument.Successful Then
              MsgBox ("Instrument successfully reserved")
        Else
              MsgBox ("Failure")
        End If
 End If
End Function
```

```
[C#]
public  object   ILIMSConnection.Instrument { get; }
```

Parameters

Value

[out,retval]

2.3.1.5 Host Property

Gets the IP Address or machine name of the computer on which the LIMS server is running.

```
[C/C++]
HRESULT   get_Host( BSTR* Value );
HRESULT   put_Host( BSTR Value );
```

```
[Visual Basic]
Public Overloads Property  Host As String
```

VB code snipet :

```
Public Function Login(User As String, Password As String, Host As String, ->
HostPort As String) As LIMSOperationResult
    gConnection.Host = Host
    gConnection.Port = HostPort
    If Assigned(gConnection) Then
        Set Login = gConnection.Login(User, Password)
        If Login.Successful Then
            MsgBox ("Logged in")
        Else
            MsgBox ("LoginFailure")
        End If
    Else
        MsgBox "Global LIMS Proxy object is not assigned"
    End If
End Function
```

```
[C#]
public  ref string   ILIMSConnection.Host { get; set; }
```

Parameters

Value

[out,retval]

2.3.1.6 Port Property

Gets the TCP port number on which the LIMS server is listening (default = 1776).

```
[C/C++]
HRESULT   get_Port( LONG* Value );
HRESULT   put_Port( LONG Value );
```

```
[Visual Basic]
Public Overloads Property  Port As Long
```

VB code snipet : **See** ILIMSConnection::Host 33

```
[C#]
public  ref int   ILIMSConnection.Port { get; set; }
```

Parameters

Value

[out,retval]

2.3.1.7 Login Method

Establishes a login with the LIMS server if not already logged in.

```
[C/C++]
HRESULT   Login(
    BSTR                    User,
    BSTR                    Password,
    ILIMSOperationResult** Value
);
```

```
[Visual Basic]
object.Login(
   ByVal User     As String,
   ByVal Password As String,
) As Object
```

VB code snipet :

```
Public Function Login(User As String, Password As String, ->
Host As String, HostPort As String) As LIMSOperationResult
    gConnection.Host = Host
    gConnection.Port = HostPort
    If Assigned(gConnection) Then
        Set Login = gConnection.Login(User, Password)
        If Login.Successful Then
            MsgBox ("Logged in")
        Else
            MsgBox ("LoginFailure")
        End If
    Else
        MsgBox "Global LIMS Proxy object is not assigned"
    End If
End Function
```

```
[C#]
void   ILIMSConnection.Login(
    string User,
    string Password,
);
```

Parameters

User

[in]

Password

[in]

Value

[out,retval]

See also note on Reserve Method 44

2.3.1.8 Logout Method

Terminates the login with the LIMS server.

```
[C/C++]
HRESULT   Logout(
    ILIMSOperationResult** Value
);
```

```
[Visual Basic]
object.Logout() As Object
```

VB code snipet :

```
Public Function Logout() As LIMSOperationResult
On Error GoTo Error
    If gConnection.LoggedIn Then
         Set Logout = gConnection.Logout
        If Logout.Successful Then
            MsgBox ("Successfully Logged Out")
        Else
            MsgBox ("Failure to logout")
        End If
    End If
Error:
    Exit Function
End Function
```

```
[C#]
object   ILIMSConnection.Logout();
```

Parameters

Value

[out,retval]

2.3.2 ILIMSExperimentInfo

Interface providing information about the experiment.

General methods and properties.		
method	GetStatus 38	Returns the current state of the specified experiment
method	GetCompletedExperimentSummary 40	Generates and returns summary information for the specified experiment
method	ExportExperiment 42	Exports an experiment to an IXO file

2.3.2.1 GetStatus Method

Returns the current state of the specified experiment.
Important note: read the workflow 18 topic

```
[C/C++]
HRESULT   GetStatus(
    BSTR                    ExperimentName,
    BSTR*                   Status,
    ILIMSOperationResult** Value
);
```

```
[Visual Basic]
object.GetStatus(
   ByVal ExperimentName as string,
   ByRef Status         as string,
) as object
```

VB code snipet :

```
Public Function GetExperimentStatus(ExperimentName As String)->
As LIMSOperationResult
    Dim EStatus As String
    EStatus = ""
    If Assigned(gConnection) Then
        Set GetExperimentStatus = gConnection.ExperimentInfo.->
GetStatus(ExperimentName, EStatus)
        If GetExperimentStatus.Successful Then
            MsgBox ( "Status of experiment : " & EStatus)
        Else
                MsgBox ("No Status found")
        End If
    Else
        MsgBox "Global LIMS Proxy object is not assigned"
    End If
End Function
```

```
[C#]
void   ILIMSExperimentInfo.GetStatus(
    string     ExperimentName,
    ref string Status,
);
```

Parameters
ExperimentName
[in]
Status
[out]
Value
[out,retval]

Remarks
See table next page

The values for experiment status are as follows:

'No analyses':	experiment completed -- no analyses run
'Has analyses':	experiment completed -- at least one analysis run
'Not started':	experiment created, but not started in the instrument
'Running':	experiment running in the instrument
'Aborted':	experiment in the instrument was aborted
'Error':	experiment in the instrument had an error

2.3.2.2 GetCompletedExperimentSummary Method

Generates and returns summary information for the specified experiment.

```
[C/C++]
HRESULT   GetCompletedExperimentSummary(
    BSTR                  ExperimentName,
    BSTR*                 Summary,
    ILIMSOperationResult** Value
);
```

```
[Visual Basic]
object.GetCompletedExperimentSummary(
   ByVal ExperimentName As String,
   ByRef Summary        As String,
) As Object
```

VB code snipet :

```
Public Function GetExperimentSummaryAsXML(ExpName)
    Dim SummaryText As String
    If Assigned(gConnection) Then
        Set GetExperimentSummaryAsXML = gConnection.ExperimentInfo. ->
GetCompletedExperimentSummary(ExpName, SummaryText)
            If GetExperimentSummaryAsXML.Successful Then
                Main.Text1.Text = SummaryText
            ' To display: Use a large text container, the summary
            ' can be > several tenth of MB
            ' In VB a RichTextBox is suitable for such large objects.
                MsgBox ("Size of summary =" & Str(Len(SummaryText)))
             Else
                MsgBox ("Failure to acquire summary")
            End If
    Else
        MsgBox " LIMS Proxy Not Assigned"
    End If
End Function
```

```
[C#]
void   ILIMSExperimentInfo.GetCompletedExperimentSummary(
    string     ExperimentName,
    ref string Summary,
);
```

Parameters

ExperimentName

[in]

Summary

[out]

Value

[out,retval]**See Notice next page**

Notice

The experiment XML has several properties and four subsections, namely run, analysis, subsets, and samples:

```
<Experiment>
    <prop name="name">Standard-384-2</prop>
         more properties ...
    <run>
         run content ...
    </run>
    <analyses>
         analyses content ...
    </analyses>
    <Subsets>
         subset content ...
    <Subsets>
    <Samples>
         samples content ...
    </Samples>
</Experiment>
```

- Each section corresponds roughly to a section in the Experiment summary screen. The most commonly used section will be the analysis section, where analysis results can be obtained.

- For Absolute Quantification, Tm Calling, and Genotyping, each analysis section has a list of properties, followed by a list of samples. The analysis results for each sample are given in the list, and correspond to the columns in the result table of the analysis.

- For Relative Quantification, each analysis section has several properties and a list of pairings. For each pairing, the analysis results are listed as properties, and generally correspond to the columns on the result tab in the relative quantification analysis screen.

2.3.2.3 ExportExperiment Method

Exports an experiment to an IXO file.

```
[C/C++]

HRESULT   ExportExperiment(
    BSTR                  ExperimentName,
    BSTR                  Filename,
    ILIMSOperationResult** Value
);
```

```
[Visual Basic]

object.ExportExperiment(
   ByVal ExperimentName As String,
   ByVal Filename       As String,
) As Object
```

```
[C#]

void   ILIMSExperimentInfo.ExportExperiment(
    string ExperimentName,
    string Filename,
);
```

Parameters

ExperimentName

[in]

Filename

[in]

Value

[out,retval]

2.3.3 ILIMSInstrument

Interface for controlling the instrument.

General methods and properties.		
method	Reserve 44	Attempt to obtain exclusive control of the instrument
method	Unreserve 45	Relinquish exclusive control of the instrument
method	Open 46	Open the loading door on the instrument
method	Close 48	Close the loading door on the instrument
method	StartExperiment 49	Run the specified experiment with the given parameters
method	GetStatus 53	Returns the instrument status as a string
method	AbortExperiment 55	Abort the currently running experiment
method	GetContainerBarcode 56	Get the barcode of the tray currently loaded on the instrument
method	OpenAndWait 57	open the loading door and wait for the status of the instrument
method	CloseAndWait 58	close the loading door and wait for the status of the instrument

Important note: Only a real and active instrument in the database can be reserved. Virtual instruments cannot be reserved or controlled, all the above methods apply only to physical instruments.

2.3.3.1 Reserve Method

Attempt to obtain exclusive control of the instrument

Before attempting to control an instrument, it should be reserved. Please note that exclusive control is only obtained if the relevant setup is correct.
See Multiple LIMS users 22. Otherwise the control may be obtained by multiple clients.

[C/C++]

```
HRESULT   Reserve(
    ILIMSOperationResult** Value
);
```

[Visual Basic]

```
object.Reserve() As Object
```

VB code snipet :

```
Public Function ReserveInstrument() As LIMSOperationResult
    If Assigned(gConnection) Then
        Set ReserveInstrument = gConnection.Instrument.Reserve
           If ReserveInstrument.Successful Then
                MsgBox "Instrument successfully reserved"
           Else
                ErNum = Str(ReserveInstrument.ErrorNumber)
                ErMessage = ReserveInstrument.Message
                MsgBox ( "Reserve failure info:" & ErNum & "/" & ErMessage)
           End If
     Else
        MsgBox "Global LIMS Proxy object is not assigned"
    End If
End Function
```

[C#]

```
object   ILIMSInstrument.Reserve();
```

Parameters

Value

[out,retval]

Important note: The information about the active instrument has to be retrieved from the database.
Because this record may not be instantly available right after a successful login, it is recommended to have a delay of 3 to 5 seconds before attempting to reserve the instrument.
This can be even more critical if the database is not located in the instrument control unit, but on a remote database server.
The delay must be adjusted according to the server and network response time.

2.3.3.2 Unreserve Method

Relinquish exclusive control of the instrument

[C/C++]

```
HRESULT   Unreserve(
    ILIMSOperationResult** Value
);
```

[Visual Basic]

object.Unreserve() As Object

```
Public Function UnreserveInstrument() As LIMSOperationResult
    If Assigned(gConnection) Then
        Set UnreserveInstrument = gConnection.Instrument.Unreserve
        If UnreserveInstrument.Successful Then
            MsgBox ("Instrument successfully released")
        Else
            MsgBox ("Failure to unreserve")
        End If
    Else
        MsgBox "Global LIMS Proxy object is not assigned"
    End If
End Function
```

[C#]

```
object   ILIMSInstrument.Unreserve();
```

Parameters

Value

[out,retval]

2.3.3.3 Open Method

Open the loading drawer on the instrument

Preliminary remarks.

As stated in the General information, the command is synchronous like most of the API methods.

When the command returns control a success condition will most probably occur, regardless of the mechanical movement being completed or not.
The LIMS API cannot assess the position of the plate drawer. The success condition refers only to the fact that the command was accepted and acknowledged by the system.

It is not possible to find directly from the success condition if the open movement was successfully completed or not, however, it is possible to find indirectly, by polling the instrument status after the open command completed "successfully".
If a mechanical failure occurred e.g. a mechanical obstacle prevented the movement of the drawer, the Instrument status will return an error condition
See GetStatus Method 53

A successful Open should result in a **'Standby (no MWP)'** condition for the status testing.

Additionally, an attempt to open or close during the initialization phase of the analyzer will result into a timeout, but this will not be reported to the LIMS client, but rather as a "success".
For this reason, it is a good idea to test for the instrument standby condition before invoking the open function.

[C/C++]

```
HRESULT   Open(
    ILIMSOperationResult** Value
);
```

[Visual Basic]

```
object.Open() As Object
```

VB code snipet :

```
Public Function OpenInstrument() As LIMSOperationResult
Dim TrayStatus As LIMSOperationResult
    If Assigned(gConnection) Then
        Set OpenInstrument = gConnection.Instrument.Open
        If OpenInstrument.Successful Then
                MsgBox ( "Open command accepted")
                ' Take some steps to check if the Instrument Status
                ' is not 'Error'
                ' See Function ILIMSInstrument GetStatus 53
        Else
            MsgBox ("Open Command Failure")
            ErNum = Str(OpenInstrument.ErrorNumber)
            ErMessage = OpenInstrument.Message
            Msg ^Box ("Opening Error:" & ErNum & "/" & ErMessage)
        End If
    Else
        MsgBox "Global LIMS Proxy object is not assigned"
    End If
End Function
```

[C#]

```
object   ILIMSInstrument.Open();
```

Parameters

Value

[out,retval]

2.3.3.4 Close Method

Close the loading drawer on the instrument

Preliminary remarks.
The same general remarks apply as for the Open function.

If a mechanical failure occurred e.g. a mechanical obstacle prevented the movement of the drawer, the Instrument status will return an error condition See GetStatus Method 53

A successful Close should result in a **'Standby (no MWP)' or 'Standby (MWP loaded)'** condition for the status testing.

An attempt to open or close during the initialization phase of the analyzer, or if the drawer is already closed, will result into a timeout, but no error condition will be reported to the LIMS client in such cases.

[C/C++]

```
HRESULT   Close(
    ILIMSOperationResult** Value
);
```

[Visual Basic]

object.Close() As Object

VB code snipet :

```
Public Function CloseInstrument() As LIMSOperationResult
    If Assigned(gConnection) Then
        Set CloseInstrument = gConnection.Instrument.Close
        If CloseInstrument.Successful Then
            MsgBox ("Tray command successfully acknowledged")
        Else
            MsgBox ("Failure to close")
        End If
    Else
        MsgBox "Global LIMS Proxy object is not assigned"
    End If
End Function
```

[C#]

```
object   ILIMSInstrument.Close();
```

Parameters

Value

[out,retval]

2.3.3.5 StartExperiment method

Run the specified experiment with the given parameters
See also the SampleDefinition 67 general Methods and properties.

- If sample definitions are to be used for the run, the collection of samples **must** be created (instantiated) **before** starting the experiment.
- A valid macro name must be passed to be able to start and experiment. The macro must exist and must belong to the logged in user, unless the user is administrator in the database. See here 51 for more explanation on the names.

[C/C++]

```
HRESULT   StartExperiment(
    BSTR                    ExperimentName,
    BSTR                    ContainerBarCode,
    BSTR                    MacroName,
    ILIMSSampleDefinition*  SampleDefinition,
    ILIMSOperationResult**  Value
);
```

[Visual Basic]

```
object.StartExperiment(
   ByVal ExperimentName   As String,
   ByVal ContainerBarCode As String,
   ByVal MacroName        As String,
   ByRef SampleDefinition As ILIMSSampleDefinition,
) As Object
```

VB code snipet :

```
Public Function StartExperiment(aExperimentName As String, aBarCodeContainer ->
As String, aMacroName As String, aSampleDef As LIMSSampleDefinition) ->
As LIMSOperationResult
    Dim RetValue As Integer
    If Assigned(gConnection) Then
        Set StartExperiment = gConnection.Instrument.StartExperiment ->
(aExperimentName, aBarCodeContainer, aMacroName, aSampleDef)
        If StartExperiment.Successful Then
            MsgBox ("Experiment started")
        Else
                ErNum = Str(StartExperiment.ErrorNumber)
                ErMessage = StartExperiment.Message
                RetValue = MsgBox ("Error:" & ErNum & "/" & ErMessage,vbOKOnly &_
                  vbInformation,"Experiment Start Failed")
        End If
    Else
        MsgBox "Global LIMS Proxy object is not assigned"
    End If
End Function
```

[C#]

```
void   ILIMSInstrument.StartExperiment(
    string                   ExperimentName,
    string                   ContainerBarCode,
    string                   MacroName,
    ref ILIMSSampleDefinition SampleDefinition,
);
```

Parameters

ExperimentName

[in]

ContainerBarCode

[in]

MacroName

[in]

SampleDefinition

[in]

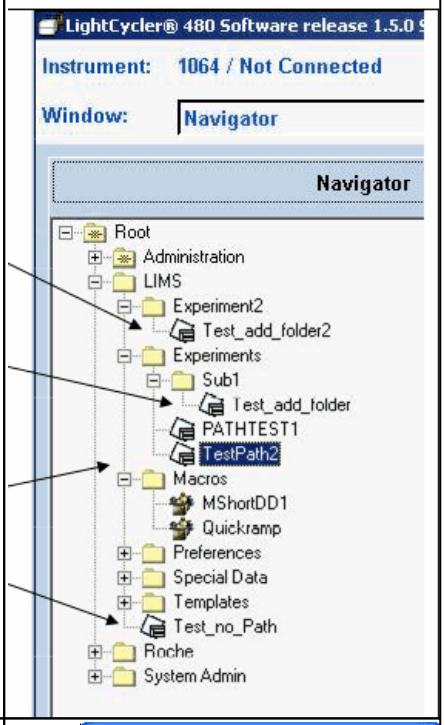
Value

[out,retval]

Macro and Experiment names

Example, If you are logged in as user "LIMS" (User LIMS has Administrator privileges)

Note: It is recommended to create a separate user for the LIMS access, and not use the System Admin to run the LIMS.
This LIMS user can be an administrator, and should be used exclusively for this purpose. If LIMS is an administrator it can also access other user's Macros and store the Experiments into other Folders.

Strings entered for Experiment:	
"LIMS/Experiment2/Test_add_folder2" The folder "Experiment2" was created as it did not exist previously "LIMS/Experiments/Sub1/Test_add_folder" The Sub folder "Sub1" was created as it did not exist previously "LIMS/TestPath2" "Test_no_Path" - No path is defined, the experiment goes to the base path of current user.(LIMS)	
An example of ***incorrect path:*** *Note the leading slash in the string below* "/LIMS/TestPath2" Adding folders to the root is not permitted.	

Syntax for the parameter "Macro"

"LIMS/Macros/MShortDD1"

To obtain the Macro lists, you can either run a query from your application, (See ILIMSQUERY 90) or open the LightCycler software and log in as your proposed LIMS user:

Once logged in, to see the folders, click the Icon for the navigator

Do not forget to close the LightCycler software before starting the LIMS server.
(The LC software will reserve the instrument and lock it if started first, and also lock the write access to the DB).

Additional rules:

- The Experiment name must be unique within a folder.
- The case is relevant. (Also for macros).

2.3.3.6 GetStatus Method

Returns the instrument status as a string

Note1 : This command is also useful to find out if a **close** or **open** command was successfully completed. See Open Method 46 and Close Method 48

Note2: It is not sufficient to test the instrument status for the end of a run. Please refer to the flowchart 18.

[C/C++]

```
HRESULT   GetStatus(
    BSTR*                  Status,
    ILIMSOperationResult** Value
);
```

[Visual Basic]

```
object.GetStatus(
   ByRef Status As String,
) As Object
```

VB code snipet :

```
Public Function GetInstrumentStatus() As LIMSOperationResult
    Dim IStatus As String
    IStatus = ""
    If Assigned(gConnection) Then
        Set GetInstrumentStatus = gConnection.Instrument.GetStatus(IStatus)
        If GetInstrumentStatus.Successful Then
            Select Case IStatus
                Case "Error"
                    ' Action for the ERROR status e.g. Set a Global Flag....
                Case "Initializing"
                     ' Action for the INITIALIZING status
                Case Else
                    ' Action for the OTHER status
            End Select
            MsgBox ("Instrument Status: " & IStatus)
        Else
            MsgBox ("Failure to get instrument STATUS")
        End If
    Else
        MsgBox "Global LIMS Proxy object is not assigned"
    End If
End Function
```

```
[C#]

void   ILIMSInstrument.GetStatus(
    ref string Status,
);
```

Parameters

Status

[out]

Value

[out,retval]

Remarks

The values for instrument status are as follows:

'Running':	The instrument is running an experiment
'Service':	The instrument is being serviced
'Error':	The instrument has an error
'Initializing':	The intrument is initializing
'Standby (MWP loaded)':	The instrument is ready with a plate loaded
'Standby (no MWP)':	The instrument is on standby with no plate loaded

2.3.3.7 AbortExperiment Method

Abort the currently running experiment

[C/C++]

```
HRESULT   AbortExperiment(
    ILIMSOperationResult** Value
);
```

[Visual Basic]

```
object.AbortExperiment() As Object
```

VB code snipet :

```
Public Function AbortExperiment() As LIMSOperationResult
Dim RetValue As Integer
If Assigned(gConnection) Then
    Set AbortExperiment = gConnection.Instrument.AbortExperiment
    If AbortExperiment.Successful Then
            MsgBox ("Abort experiment sucessful")
            Main.CdeAbort.BackColor = &H8000000F
        Else
            ErNum = Str(AbortExperiment.ErrorNumber)
            ErMessage = AbortExperiment.Message
            RetValue = MsgBox (" "Error:" & ErNum & "/" & ErMessage, vbOKOnly &_
                  vbInformation , "Abort experiment Failed")
        End If
End If
```

[C#]

```
object   ILIMSInstrument.AbortExperiment();
```

Parameters

Value

[out,retval]

2.3.3.8 GetContainerBarcode Method

Get the barcode of the tray currently loaded on the instrument

[C/C++]

```
HRESULT   GetContainerBarcode(
    BSTR*                  Barcode,
    ILIMSOperationResult** Value
);
```

[Visual Basic]

```
object.GetContainerBarcode(
   ByRef Barcode As String,
) As Object
```

VB code snipet :

```
Public Function GetContainerBarcode() As LIMSOperationResult
    Dim CBarcode As String
    If Assigned(gConnection) Then
        Set GetContainerBarcode = gConnection.Instrument.->
GetContainerBarcode(CBarcode)
        If GetContainerBarcode.Successful Then
            MsgBox ("Container Barcode: " & CBarcode)

        Else
            MsgBox ("Failure no BC found")
        End If
    Else
        MsgBox "Global LIMS Proxy object is not assigned"
    End If
End Function
```

[C#]

```
void   ILIMSInstrument.GetContainerBarcode(
    ref string Barcode,
);
```

Parameters

Barcode

[out]

Value

[out,retval]

2.3.3.9 OpenAndWait Method

open the loading door and wait for the status of the instrument

Preliminary remark.

When the normal open 46 command returns control to the API a success condition will most probably occur regardless of the mechanical movement being completed or not. The LIMS API cannot assess the position of the plate drawer.

This method provides an alternative to open + get status.
As a successful Open should result in a **'Standby (no MWP)'** condition, this is normally the parameter that should passed as the **StatusToWaitFor**.

```
[C/C++]

HRESULT   OpenAndWait(
    LONG                    TimeoutSeconds,
    BSTR                    StatusToWaitFor,
    ILIMSOperationResult** Value
);
```

```
[Visual Basic]

object.OpenAndWait(
   ByVal TimeoutSeconds  as long,
   ByVal StatusToWaitFor as string,
) as object
```

```
[C#]

void   ILIMSInstrument.OpenAndWait(
    int    TimeoutSeconds,
    string StatusToWaitFor,
);
```

Parameters
TimeoutSeconds
[in]
StatusToWaitFor , normally **'Standby (no MWP)'**
[in]
Value
[out,retval]

2.3.3.10 CloseAndWait Method

close the loading door and wait for the status of the instrument

Preliminary remark.

When the normal close 48 command returns control to the API a success condition will most probably occur regardless of the mechanical movement being completed or not. The LIMS API cannot assess the position of the plate drawer.

This method provides an alternative to close + get status.
As a successful Close should result in a **'Standby (no MWP)' or 'Standby (MWP loaded)'** condition.

Unless the program has also full control on the loading of a plate, e.g. robotic arm, it is difficult to use this function as the correct expected condition is not completely predictable.

```
[C/C++]

HRESULT   CloseAndWait(
    LONG                    TimeoutSeconds,
    BSTR                    StatusToWaitFor,
    ILIMSOperationResult** Value
);
```

```
[Visual Basic]

object.CloseAndWait(
   ByVal TimeoutSeconds  As long,
   ByVal StatusToWaitFor As String,
) As Object
```

```
[C#]

void   ILIMSInstrument.CloseAndWait(
    int    TimeoutSeconds,
    string StatusToWaitFor,
);
```

Parameters

TimeoutSeconds

[in]

StatusToWaitFor

[in]

Value

[out,retval]

2.3.3.11 GetContainerSensor

Get the current setting of the container (plate) sensor

Preliminary remarks.

The plate sensor detects the difference between MWP with either 96 or 384 wells. The sensor can function only with white plates.
When clear plates (transparent plastic) are used, the sensor must be turned OFF.
For the manual use of the instruments, the Administrator can setup general parameters in one of three way:

1. Always ON (White plates only)
2. Always OFF (Clear plates)
3. Mixed (The user decides at the start of each run)

Independent of those settings, the LIMS interface can turn the sensor ON or OFF at any time.
(See also SetContainerSensor 61)

Values returned by the method:
can be "ON"; "OFF"; "failed"

[C/C++]

```
HRESULT   GetContainerSensor(

    BSTR*                  Sensor,
    ILIMSOperationResult** Value
);
```

[Visual Basic]

```
object.GetContainerSensor(
   ByRef Sensor as string,
) as object
```

VB code snipet :

```
Public Function GetContSens_value() As LIMSOperationResult
    Dim ContSensVal As String
    If Assigned(gConnection) Then
        Set GetContSens_value = ->
            gConnection.Instrument.GetContainerSensor(ContSensVal)
        Main.PlateType.Text = ContSensVal ' display result in a text box
    Else
        MsgBox "Global LIMS Proxy object is not assigned"
    End If
End Function
```

[C#]

```
void   ILIMSInstrument.GetContainerSensor(
    ref string Sensor,
);
```

Parameters

Sensor
[out]

[out,retval]

2.3.3.12 SetContainerSensor

Set the Container (plate) Sensor to the specified value

Value can be "ON"; "OFF"

```
[C/C++]
HRESULT   SetContainerSensor(

    BSTR                   Sensor,
    ILIMSOperationResult** Value

);
```

```
[Visual Basic]

object.SetContainerSensor(
   ByVal Sensor As String,
) As Object
```

VB code snipet :

```
Public Function SetContSens_value(SensVal As String)
 ' SensVal can be 'ON' or 'OFF'
    If Assigned(gConnection) Then
        gConnection.Instrument.SetContainerSensor (SensVal)
    Else
        MsgBox "Global LIMS Proxy object is not assigned"
    End If
End Function
```

```
[C#]

void   ILIMSInstrument.SetContainerSensor(
    string Sensor,
);
```

Parameters

Sensor

[in]

Value

[out,retval]

2.3.4 ILIMSOperationResult

Dispatch interface for LIMSOperationResult Object

General methods and properties.		
property get	Successful 63	True if the operation was successful, false otherwise
property get	ServerError 64	True if error occurred on LIMS server rather than client
property get	Message 64	Technical error message
property get	ErrorNumber 65	Error number
property get	DateTime 65	Date and time that the error occurred
property get	UserMessage 66	End user error message

2.3.4.1 Successful Property

True if the operation was successful, false otherwise

```
[C/C++]
HRESULT   get_Successful( VARIANT_BOOL* Value );
```

```
[Visual Basic]
Public ReadOnly Property  Successful As Boolean
```

VB code snipet :

```
......
Set ReserveInstrument = gConnection.Instrument.Reserve
  If ReserveInstrument.Successful Then
       MsgBox "Instrument successfully reserved"
  Else
   If ReserveInstrument.ServerError then        'Detect SERVER errors
            Stamp = ReserveInstrument.DateTime        'Server error Time Stamp
            ErNum = Str(ReserveInstrument.ErrorNumber)'Fetch Error number
            ErMessage = ReserveInstrument.Message     'Fetch Message
            RetValue = MsgBox ( "Err Number : " & ErNum & "/" &_
      ErMessage,vbOKOnly & vbInformation, "Reserve failure info at " &Tstamp )
   End If
  End If
...........
```

```
[C#]
public  ref bool   ILIMSOperationResult.Successful { get; }
```

Parameters

Value

[out,retval]

2.3.4.2 ServerError Property

True if error occurred on LIMS server rather than client

```
[C/C++]
HRESULT   get_ServerError( VARIANT_BOOL* Value );
```

```
[Visual Basic]
Public ReadOnly Property  ServerError As Boolean
```

VB code snipet : see Successful Property 63

```
[C#]
public  ref bool   ILIMSOperationResult.ServerError { get; }
```

Parameters

Value

[out,retval]

2.3.4.3 Message Property

Technical error message

```
[C/C++]
HRESULT   get_Message( BSTR* Value );
```

```
[Visual Basic]
Public ReadOnly Property  Message As String
```

VB code snipet : see Successful Property 63

```
[C#]
public  ref string   ILIMSOperationResult.Message { get; }
```

Parameters

Value

[out,retval]

2.3.4.4 ErrorNumber Property

Error number

```
[C/C++]
HRESULT   get_ErrorNumber( LONG* Value );
```

```
[Visual Basic]
Public ReadOnly Property  ErrorNumber As Long
```

VB code snipet : see Successful Property 63

```
[C#]
public  ref int   ILIMSOperationResult.ErrorNumber { get; }
```

Parameters

Value

[out,retval]

2.3.4.5 DateTime Property

Date and time when the error occurred (according to the LIMS CLIENT clock)

```
[C/C++]
HRESULT   get_DateTime( DATE* Value );
```

```
[Visual Basic]
Public ReadOnly Property  DateTime As Date
```

VB code snipet : see Successful Property 63

```
[C#]
public  ref System.DateTime   ILIMSOperationResult.DateTime { get; }
```

Parameters

Value

[out,retval]

2.3.4.6 UserMessage Property

End user error message

Note : This property does not yield any content in current version.

```
[C/C++]
HRESULT   get_UserMessage( BSTR* Value );
```

```
[Visual Basic]
Public ReadOnly Property  UserMessage As String
```

```
[C#]
public  ref string   ILIMSOperationResult.UserMessage { get; }
```

Parameters

Value

[out,retval]

2.3.5 ILIMSSampleDefinition

Represents a collection of samples

General methods and properties.		
property get	SampleCount 68	returns the number of samples in the collection
property get	ExpDataCount 69	Returns the number of experiment data strings
method	AddSample 70	Adds a sample to the collection
method	Clear 72	Removes all samples from the collection
method	DeleteSample 73	Delete the sample at the given index position
method	GetSample 74	Returns the sample at the given index position
method	AddExpData 76	Add an experiment data string
method	GetExpData 77	Return the experiment data string at the given index
method	DeleteExpData 78	Delete the experiment data string at the given index
method	ClearExpData 79	Delete all experiment data strings

A Sample collection (ILIMSSampledefinition) object must be created before the collection of samples can be manipulated by the program:

```
[Visual Basic]
```

VB code snipet :

```
set gSampleDef = New LIMSSampleDefinition
```

2.3.5.1 SampleCount Property

returns the number of samples in the collection

```
[C/C++]
HRESULT   get_SampleCount( LONG* Value );
```

```
[Visual Basic]
Public ReadOnly Property  SampleCount As Long

' The sample collection gSampleDef must exist prior to calling this routine.
' e.g. as a Global variable. see ILIMSSampleDefinition 67

Public Sub GetSampleCount()
    Dim SampCount As Long
    On Error GoTo Error
    If (Assigned(gConnection) And gConnection.LoggedIn = True) Then
        SampCount = gSampleDef.SampleCount
        RetValue = MsgBox(Str(SampCount) &_
            " Samples found", vbOKOnly, " Samples in collection")
    End If
Exit Sub
```

```
[C#]
public  ref int   ILIMSSampleDefinition.SampleCount { get; }
```

Parameters

Value

[out,retval]

2.3.5.2 ExpDataCount Property

Returns the number of experiment data strings

```
[C/C++]
HRESULT   get_ExpDataCount( LONG* Value );
```

```
[Visual Basic]
Public ReadOnly Property  ExpDataCount As Long
```

```
[C#]
public  ref int   ILIMSSampleDefinition.ExpDataCount { get; }
```

Parameters

Value

[out,retval]

2.3.5.3 AddSample Method

Adds a sample to the collection

Notice: The Replicate Position property will be deprecated in the v 1.5.
It is strongly recommended not to use it to ensure future compatibility

```
[C/C++]
HRESULT   AddSample(
    ILIMSSampleInfo* Sample
);
```

```
[Visual Basic]
object.AddSample(
   ByRef Sample As ILIMSSampleInfo
```

VB code snipet :

```
Public Sub AddSample()
' Creates the sample infos and populates them with the sample data.
' Then adds each of them to the sample definition (this object must have been
' instantiated before  See ILIMSSampleDefinition 67 )

    Dim gSampleInfo As ILIMSSampleInfo
    Dim SampCount As Long
    Dim SamplePos As String
    gSampleDef.Clear             ' First clear the whole collection

      'The sample set must be first instantiated
    Set gSampleInfo = New LIMSSampleInfo
    gSampleInfo.Position = "A1"
    gSampleInfo.ID = "FirstID"
    gSampleInfo.Name = "Sample Name XYZ"
    gSampleInfo.Comment = "Dummy Sample"

    gSampleDef.AddSample gSampleInfo    ' Add the set to the collection.


    ' Set a sample info for a replicate
    Set gSampleInfo = New LIMSSampleInfo

    gSampleInfo.Position = "A3"
        gSampleInfo.ID = "" ' Name,ID,Comment MUST be empty for repl. pos.
        gSampleInfo.Name = ""
        gSampleInfo.Comment = ""
    gSampleInfo.ReplicatePosition = "A1"

    gSampleDef.AddSample gSampleInfo

End Sub
)
```



```
[C#]
void   ILIMSSampleDefinition.AddSample(
    ref ILIMSSampleInfo Sample
);
```

Parameters

Sample

[in]

2.3.5.4 Clear Method

Removes all samples from the collection.

```
[C/C++]
HRESULT   Clear();
```

```
[Visual Basic]
object.Clear()
```

VB code snipet : see AddSample Method 70

```
[C#]
void   ILIMSSampleDefinition.Clear();
```

2.3.5.5 DeleteSample Method

Delete the sample at the given index position

```
[C/C++]
HRESULT   DeleteSample(
    LONG Index
);
```

```
[Visual Basic]
object.DeleteSample(
   ByVal Index As Long
)
```

```
[C#]
void   ILIMSSampleDefinition.DeleteSample(
    int Index
);
```

Parameters

Index

[in]

2.3.5.6 GetSample Method

Returns the sample at the given index position

```
[C/C++]
HRESULT   GetSample(
    LONG              Index,
    ILIMSSampleInfo** Value
);
```

```
[Visual Basic]
object.GetSample(
   ByVal Index As Long,
) As Object

Public Sub SampFback(Scount As Integer) ' Scount = Number of samples found
                                ' in the collection (See SampleCount Property)
    Dim Disp As String
    Dim i, j, X As Integer
    ' SampFB is a RichTextBox control to display the list with tabs
    ' Samp_Feedback is the v.b. form containing the above RichTextBox

    ScaleMode = vbPixels
    Disp = ""
    Samp_Feedback.Show
    SampFB.Text = Disp

    SampFB.SelStart = 0
    SampFB.SelLength = Len(Samp_Feedback.SampFB.Text)
    SampFB.SelTabCount = 4
    SampFB.SelTabs(0) = 50
    SampFB.SelTabs(1) = 200
    SampFB.SelTabs(2) = 300
    SampFB.SelTabs(3) = 600

    Disp = "Position" & vbTab & "ID" & vbTab &_
      "Name" & vbTab & "Comment" & vbCrLf & vbCrLf
    SampFB.Text = Disp

    For i = 0 To Scount - 1
        Set gSampleInfo = gSampleDef.GetSample(i)
        Disp = Disp & gSampleInfo.Position & vbTab & gSampleInfo.ID &_
            vbTab & gSampleInfo.Name & vbTab & gSampleInfo.Comment & vbCrLf
        SampFB.Text = Disp
    Next i
Exit Sub
Error:
RetValue = MsgBox("Error#" & Str(Err.Number) &_
vbCrLf & Err.Description, vbOKOnly, "Error sample data read")
End Sub
```



```
[C#]
void   ILIMSSampleDefinition.GetSample(
    int    Index,
);
```

Parameters

Index

[in]

Value

[out,retval]

2.3.5.7 AddExpData Method

Add an experiment data string

```
[C/C++]
HRESULT   AddExpData(
    BSTR ExpData
);
```

```
[Visual Basic]
object.AddExpData(
   ByVal ExpData As String
)
```

```
[C#]
void   ILIMSSampleDefinition.AddExpData(
    string ExpData
);
```

Parameters

ExpData

Remarks

In LightCycler 480 version 1.5, the experiment data object is used only to set the external experiment name for a relative quantification or PCR Endpoint Genotyping experiments.

Relative Quantification uses the external experiment name as well as the subset and program. The subset name should be the same name that is displayed in subset editor. The program number is a zero-based integer referring to programs as listed in the experiment. The XML string that should be passed to AddExpData is

<RelQuant>
<ReferenceExperimentName>experiment name goes here</ReferenceExperimentName>
<ReferenceExperimentSubsetName>subset name goes here</ReferenceExperimentSubsetName>
<ReferenceExperimentProgramNumber>nprogram number goes here... first program is 0</ReferenceExperimentProgramNumber>
</RelQuant>

Endpoint Genotyping requires only the experiment name. The XML string that should be passed to AddExpData is

<EndptGeno>
<ReferenceExperimentName>name goes here</ReferenceExperimentName>
</EndptGeno>

Note that the XML values (i.e. "name goes here") must be replaced by actual values..

2.3.5.8 GetExpData Method

Return the experiment data string at the given index

```
[C/C++]
HRESULT   GetExpData(
    LONG  Index,
    BSTR* Value
);
```

```
[Visual Basic]
object.GetExpData(
   ByVal Index As Long,
) As String
```

```
[C#]
void   ILIMSSampleDefinition.GetExpData(
    int        Index,
);
```

Parameters

Index

[in]

Value

[out,retval]

2.3.5.9 DeleteExpData Method

Delete the experiment data string at the given index

```
[C/C++]
HRESULT   DeleteExpData(
    LONG Index
);
```

```
[Visual Basic]
object.DeleteExpData(
   ByVal Index As Long
)
```

```
[C#]
void   ILIMSSampleDefinition.DeleteExpData(
    int Index
);
```

Parameters

Index

[in]

2.3.5.10 ClearExpData Method

Delete all experiment data strings

```
[C/C++]
HRESULT   ClearExpData();
```

```
[Visual Basic]
object.ClearExpData()
```

```
[C#]
void   ILIMSSampleDefinition.ClearExpData();
```

2.3.6 ILIMSSampleInfo

Represents a single sample in a SampleDefinition collection

General methods and properties.		
property get / put	Position 81	Get the sample position
property get / put	Name 82	Get the sample name
property get / put	ID 83	Get the sample ID
property get / put	Comment 84	Get the sample comment
property get / put	ReplicatePosition 85	Get the address of the master sample (i.e. A1 or C3) for replicates
property get	ExpDataCount 86	(for future use) Returns the number of experiment data strings for this sample
method	AddExpData 86	(for future use) Add an experiment data string for this sample
method	GetExpData 87	(for future use) Return the experiment data string at the given index
method	DeleteExpData 87	(for future use) Delete the experiment data string at the given index
method	ClearExpData 88	(for future use) Delete all experiment data strings

2.3.6.1 Position Property

Get/Put the sample position.

```
[C/C++]
HRESULT   get_Position( BSTR* Value );
HRESULT   put_Position( BSTR Value );
```

```
[Visual Basic]
Public Overloads Property  Position As String
```

VB code snipet : see AddSample Method 70

```
[C#]
public  ref string   ILIMSSampleInfo.Position { get; set; }
```

Parameters

Value

[out,retval]

2.3.6.2 Name Property

Get the sample name or assign it to the Name property.

Note : when creating a sample position, the sample name **cannot** be blank ("", or empty string).

```
[C/C++]
HRESULT   get_Name( BSTR* Value );
HRESULT   put_Name( BSTR Value );
```

```
[Visual Basic]
Public Overloads Property  Name As String
```

VB code snipet : see AddSample Method 70

```
[C#]
public  ref string   ILIMSSampleInfo.Name { get; set; }
```

Parameters

Value

[out,retval]

2.3.6.3 ID Property

Get the sample ID

```
[C/C++]
HRESULT   get_ID( BSTR* Value );
HRESULT   put_ID( BSTR Value );
```

```
[Visual Basic]
Public Overloads Property  ID As String
```

VB code snipet : see AddSample Method 70

```
[C#]
public  ref string   ILIMSSampleInfo.ID { get; set; }
```

Parameters

Value

[out,retval]

2.3.6.4 Comment Property

Get the sample comment

```
[C/C++]
HRESULT   get_Comment( BSTR* Value );
HRESULT   put_Comment( BSTR Value );
```

```
[Visual Basic]
Public Overloads Property  Comment As String
```

VB code snipet : see AddSample Method 70

```
[C#]
public  ref string   ILIMSSampleInfo.Comment { get; set; }
```

Parameters

Value

[out,retval]

2.3.6.5 ReplicatePosition Property *****

Get the address of the master sample (i.e. A1 or C3) for replicates

Notice: The Replicate Position property will be deprecated in the v 1.5.
It is strongly recommended not to use it to ensure future compatibility

```
[C/C++]
HRESULT   get_ReplicatePosition( BSTR* Value );
HRESULT   put_ReplicatePosition( BSTR Value );
```

```
[Visual Basic]
Public Overloads Property  ReplicatePosition As String
```

VB code snipet : see AddSample Method 70

```
[C#]
public  ref string   ILIMSSampleInfo.ReplicatePosition { get; set; }
```

Parameters

Value

[out,retval]

2.3.6.6 ExpDataCount Property

Returns the number of experiment data strings for this sample

```
[C/C++]
HRESULT   get_ExpDataCount( LONG* Value );
```

```
[Visual Basic]
Public ReadOnly Property  ExpDataCount As Long
```

```
[C#]
public  ref int   ILIMSSampleInfo.ExpDataCount { get; }
```

Parameters

Value

[out,retval]

2.3.6.7 AddExpData Method

Add an experiment data string for this sample

```
[C/C++]
HRESULT   AddExpData(
    BSTR ExpData
);
```

```
[Visual Basic]
object.AddExpData(
   ByVal ExpData As String
)
```

```
[C#]
void   ILIMSSampleInfo.AddExpData(
    string ExpData
);
```

Parameters

ExpData

[in]

2.3.6.8 GetExpData Method

Return the experiment data string at the given index.

```
[C/C++]
HRESULT   GetExpData(
    LONG  Index,
    BSTR* Value
);
```

```
[Visual Basic]
object.GetExpData(
   ByVal Index As Long,
) As String
```

```
[C#]
void   ILIMSSampleInfo.GetExpData(
    int        Index,
);
```

Parameters

Index

[in]

Value

[out,retval]

2.3.6.9 DeleteExpData Method

Delete the experiment data string at the given index.

```
[C/C++]
HRESULT   DeleteExpData(
    LONG Index
);
```

```
[Visual Basic]
object.DeleteExpData(
   ByVal Index As Long
)
```

```
[C#]
void   ILIMSSampleInfo.DeleteExpData(
    int Index
);
```

Parameters

Index

[in]

2.3.6.10 ClearExpData Method

Delete all experiment data strings.

```
[C/C++]
HRESULT   ClearExpData();
```

```
[Visual Basic]
object.ClearExpData()
```

```
[C#]
void   ILIMSSampleInfo.ClearExpData();
```


2.3.7 ILIMSQuery

General methods and properties.		
property get / put	**ObjectType** 90	The type of object to return, or empty string for all object types.
property get / put	**Name** 92	Filter for object name
property get / put	**Owner** 92	Filter for owner of object
property get / put	**FromDate** 93	Begin date search range for object creation or modification date
property get / put	**ToDate** 94	End date search range for object creation or modification date
property get / put	**QueryDateType** 95	Type of date search, either a creation date, modification date, or both. See the enumerated type for LIMSQueryDateType.
method	**ExecuteQuery** 96	Execute the query

See Also ILIMSQueryResultData::Path 103

2.3.7.1 ObjectType Property

The type of object to return, or empty string for all object types.

```
[C/C++]
HRESULT   get_ObjectType( BSTR* Value );
HRESULT   put_ObjectType( BSTR Value );
```

```
[Visual Basic]
Public Overloads Property  ObjectType As String
```

VB code snipet :

```
Public Function GetQueryResult()
    Dim QIndx, Indx As Integer
    Dim OpResult As LIMSOperationResult
    Dim Qresult As LIMSQueryResult
    Dim Qdata As LIMSQueryResultData
    Dim LQuery As LIMSQuery


    If gConnection.LoggedIn = True Then
       gConnection.Query.ObjectType = "Experiment" 'set the filters
                                                    SEE Remarks below
        gConnection.Query.Name = "*"
        gConnection.Query.Owner = ""
        Set OpResult = gConnection.Query.ExecuteQuery(Qresult)
            If OpResult.Successful = True Then
                QIndx = Qresult.Count 'Get the number of items returned
                If QIndx > 0 Then
                    Main.ExpList.Clear  ' Delete the list in main frame
                    For Indx = 0 To QIndx - 1
                        'Read Individual items and Add them to the list
                        Set Qdata = Qresult.GetResultData(Indx)
                        Main.ExpList.AddItem (Qdata.Name), Indx
                    Next Indx
                End If
            Else
                MsgBox "Exp Query unsucessful .."
            End If
         End If
End Function
```

```
[C#]
public  ref string   ILIMSQuery.ObjectType { get; set; }
```

Parameters

Value

[out,retval]

Remarks

Valid object types are as follows: Macro, Experiment, ColorComp, StdCurve, Template

See Also ILIMSQueryResultData::Path 103

2.3.7.2 Name Property

Filter for object name

```
[C/C++]
HRESULT   get_Name( BSTR* Value );
HRESULT   put_Name( BSTR Value );
```

```
[Visual Basic]
Public Overloads Property  Name As String
```

VB code snipet See ILIMSQuery::ObjectType 90
See Also ILIMSQueryResultData::Path 103

```
[C#]
public  ref string   ILIMSQuery.Name { get; set; }
```

Parameters
Value
[out,retval]

2.3.7.3 Owner Property

Filter for owner of object

```
[C/C++]
HRESULT   get_Owner( BSTR* Value );
HRESULT   put_Owner( BSTR Value );
```

```
[Visual Basic]
Public Overloads Property  Owner As String
```

VB code snipet See ILIMSQuery::ObjectType 90

```
[C#]
public  ref string   ILIMSQuery.Owner { get; set; }
```

Parameters
Value
[out,retval]

2.3.7.4 FromDate property

Begin date search range for object creation or modification date

```
[C/C++]
HRESULT   get_FromDate( DATE* Value );
HRESULT   put_FromDate( DATE Value );
```

```
[Visual Basic]
Public Overloads Property  FromDate As Date
```

VB code snipet

```
...
Dim StartDate, EndDate As Date
StartDate = "6/9/2005"
EndDate = "6/10/2005"
...
   gConnection.Query.Name = "*"
   gConnection.Query.Owner = ""
   gConnection.Query.QueryDateType = qdtCreationDateQuery ' Created between
   gConnection.Query.FromDate = StartDate                  ' First modified
   gConnection.Query.Todate = EndDate                      ' Last modified
           Set OpResult = gConnection.Query.ExecuteQuery(Qresult)
                                                                    .....
   gConnection.Query.QueryDateType = qdtAllDateQuery ' Reset the date if you
                                                       need to run further
                                                       queries w/o dates
```

See also ILIMSQuery::ObjectType 90

```
[C#]
public  ref System.DateTime   ILIMSQuery.FromDate { get; set; }
```

Parameters

Value

[out,retval]

2.3.7.5 ToDate Property

End date search range for object creation or modification date

```
[C/C++]
HRESULT   get_ToDate( DATE* Value );
HRESULT   put_ToDate( DATE Value );
```

```
[Visual Basic]
Public Overloads Property  ToDate As Date
```

VB code snipet See ILIMSQuery::FromDate 93

```
[C#]
public  ref System.DateTime   ILIMSQuery.ToDate { get; set; }
```

Parameters

Value

[out,retval]

2.3.7.6 QueryDateType property

Type of date search, either a creation date, modification date, or both. See the enumerated type for LIMSQueryDateType.

```
[C/C++]
HRESULT   get_QueryDateType( LIMSQueryDateType* Value );
HRESULT   put_QueryDateType( LIMSQueryDateType Value );
```

```
[Visual Basic]
Public Overloads Property  QueryDateType As LIMSQueryDateType
```

VB code snipet See ILIMSQuery::FromDate 93

```
[C#]
public  LIMSQueryDateType   ILIMSQuery.QueryDateType { get; set; }
```

Parameters

Value

[out,retval]

Note :

The qdtAllDateQuery value for the QueryDateType means that the date filters FromDate and ToDate are inactive. e.g the query will return all requested objects found.

2.3.7.7 ExecuteQuery Method

Execute the query

```
[C/C++]
HRESULT   ExecuteQuery(
    ILIMSQueryResult**     Result,
    ILIMSOperationResult** Value
);
```

```
[Visual Basic]
object.ExecuteQuery(
   ByRef Result As Object,
) As Object
```

VB code snipet See ILIMSQuery::ObjectType 90

```
[C#]
void   ILIMSQuery.ExecuteQuery(
    object Result,
);
```

Parameters

Result

[out]

Value

[out,retval]

2.3.8 ILIMSQueryResult

General methods and properties.		
property get	Count 98	The number of objects returned by a query.
method	GetResultData 99	method to get the individual results from a query

2.3.8.1 Count property

The number of objects returned by a query.

```
[C/C++]
HRESULT   get_Count( LONG* Value );
```

```
[Visual Basic]
Public ReadOnly Property  Count As Long
```

VB code snipet :

```
Public Function GetQueryResult()
    Dim QIndx, Indx As Integer
    Dim OpResult As LIMSOperationResult
    Dim Qresult As LIMSQueryResult
    Dim Qdata As LIMSQueryResultData
    Dim LQuery As LIMSQuery


    If gConnection.LoggedIn = True Then
       gConnection.Query.ObjectType = "Experiment" 'set the filters
                                                          SEE remarks below
        gConnection.Query.Name = "*"
        gConnection.Query.Owner = ""
        Set OpResult = gConnection.Query.ExecuteQuery(Qresult)
            If OpResult.Successful = True Then
                QIndx = Qresult.Count 'Get the number of items returned
                If QIndx > 0 Then
                    Main.ExpList.Clear  ' Delete the list in main frame
                    For Indx = 0 To QIndx - 1
                        'Read Individual items and Add them to the list
                        Set Qdata = Qresult.GetResultData(Indx)
                        Main.ExpList.AddItem (Qdata.Name), Indx
                    Next Indx
                End If
            Else
                MsgBox "Exp Query unsucessful .."
            End If
         End If
End Function
```

```
[C#]
public  ref int   ILIMSQueryResult.Count { get; }
```

Parameters

Value

[out,retval]

2.3.8.2 GetResultData method

Method to get the individual results from a query.

```
[C/C++]
HRESULT   GetResultData(
    LONG                    Index,
    ILIMSQueryResultData** Value
);
```

```
[Visual Basic]
object.GetResultData(
   ByVal Index As Long,
) As Object
```

VB code snipet : See ILIMSQueryResult::Count 98

```
[C#]
void   ILIMSQueryResult.GetResultData(
    int    Index,
);
```

Parameters

Index

[in]

Value

[out,retval]

2.3.9 ILIMSQueryResultData

General methods and properties.		
property get	**Name** 101	Name of the object
property get	**ObjectType** 101	Type of the object
property get	**CreationDate** 102	Creation date of the object
property get	**ModificationDate** 102	Modification date of the object
property get	**Path** 103	Folder path to the object on the database server

2.3.9.1 Name Property

Name of the object.

```
[C/C++]
HRESULT   get_Name( BSTR* Value );
```

```
[Visual Basic]
Public ReadOnly Property  Name As String
```

```
[C#]
public  ref string   ILIMSQueryResultData.Name { get; }
```

Parameters

Value

[out,retval]

2.3.9.2 ObjectType Property

Type of the object.

```
[C/C++]
HRESULT   get_ObjectType( BSTR* Value );
```

```
[Visual Basic]
Public ReadOnly Property  ObjectType As String
```

```
[C#]
public  ref string   ILIMSQueryResultData.ObjectType { get; }
```

Parameters

Value

[out,retval]

2.3.9.3 CreationDate Property

Creation date of the object.

```
[C/C++]
HRESULT   get_CreationDate( DATE* Value );
```

```
[Visual Basic]
Public ReadOnly Property  CreationDate As Date
```

```
[C#]
public  ref System.DateTime   ILIMSQueryResultData.CreationDate { get; }
```

Parameters

Value

[out,retval]

2.3.9.4 ModificationDate Property

Modification date of the object.

```
[C/C++]
HRESULT   get_ModificationDate( DATE* Value );
```

```
[Visual Basic]
Public ReadOnly Property  ModificationDate As Date
```

```
[C#]
public  ref System.DateTime   ILIMSQueryResultData.ModificationDate { get; }
```

Parameters

Value

[out,retval]

2.3.9.5 Path Property

Folder path to the object on the database server.

```
[C/C++]
HRESULT   get_Path( BSTR* Value );
```

```
[Visual Basic]
Public ReadOnly Property  Path As String
```

VB code snipet :

```
Public Function GetQueryResult()
    Dim QIndx, Indx As Integer
    Dim OpResult As LIMSOperationResult
    Dim Qresult As LIMSQueryResult
    Dim Qdata As LIMSQueryResultData
    Dim LQuery As LIMSQuery
    Dim ExpPath As String

    If gConnection.LoggedIn = True Then
       gConnection.Query.ObjectType = "Experiment" 'set the filters
                                                   SEE ALSO ObjectType 90
        gConnection.Query.Name = "*"
        gConnection.Query.Owner = ""
        Set OpResult = gConnection.Query.ExecuteQuery(Qresult)
            If OpResult.Successful = True Then
                QIndx = Qresult.Count 'Get the number of items returned
                If QIndx > 0 Then
                    Main.ExpList.Clear  ' Delete the list in main frame
                    For Indx = 0 To QIndx - 1
                        'Read Individual items and Add them to the list
                        Set Qdata = Qresult.GetResultData(Indx)
                        ExpPath = Qdata.Path ' Get the path to experiment
                        Main.ExpList.AddItem (Qdata.Name), Indx
                    Next Indx
                End If
            Else
                MsgBox "Exp Query unsucessful .."
            End If
         End If
End Function
```

```
[C#]
public  ref string   ILIMSQueryResultData.Path { get; }
```

Parameters

Value

[out,retval]

2.4 Enumerated Types

This section contains information about the following enumerated types used with LIMSClientLib.

- LIMSQueryDateType 104

2.4.1 LIMSQueryDateType

```
[C/C++]
typedef enum {
    qdtModificationDateQuery = 1,
    qdtCreationDateQuery     = 2,
    qdtAllDateQuery          = 0
} LIMSQueryDateType;

[Visual Basic]
Enum LIMSQueryDateType
    qdtModificationDateQuery  = 1
    qdtCreationDateQuery      = 2
    qdtAllDateQuery           = 0
End Enum

[C#]
enum LIMSQueryDateType {
    qdtModificationDateQuery = 1,
    qdtCreationDateQuery     = 2,
    qdtAllDateQuery          = 0
}
```

Constants

qdtModificationDateQuery
qdtCreationDateQuery
qdtAllDateQuery

Note :
The qdtAllDateQuery value for the QueryDateType means that the date filters FromDate and ToDate are inactive. E.g the query will return all requested objects found.

2.5 CoClasses

This section contains reference information for the COM CoClasses provided by LIMSClientLib. The following CoClasses are available with **LIMSClientLib.**

2.5.1 LIMSConnection

Implements the ILIMSConnection interface
Implemented interfaces
- ILIMSConnection 27

2.5.2 LIMSExperimentInfo

Implements the ILIMSExperimentInfo interface

Implemented interfaces
- ILIMSExperimentInfo 37

2.5.3 LIMSInstrument

Implements the ILIMSInstrument interface

Implemented interfaces
- ILIMSInstrument 43

2.5.4 LIMSOperationResult

Implements the ILIMSOperationResult interface

Implemented interfaces
- ILIMSOperationResult 62

2.5.5 LIMSSampleDefinition

Implements the ILIMSSampleDefinition interface

Implemented interfaces
- ILIMSSampleDefinition 67

2.5.6 LIMSSampleInfo

Implements the ILIMSSampleInfo interface

Implemented interfaces
- ILIMSSampleInfo 80

2.5.7 LIMSQuery

Implemented interfaces
- ILIMSQuery 89

2.5.8 LIMSQueryResult

Implements ILIMSQueryResult interface

Implemented interfaces

- ILIMSQueryResult 97

2.5.9 LIMSQueryResultData

Implements ILIMSQueryResultData

Implemented interfaces

- ILIMSQueryResultData

3 XML output

This chapter describes the complete format of the XML output, as returned by the LIMS interface when an experiment summary is requested.

3.1 Additional details, requirements

specifying the results returned by LIMS

An Abs Quant result will include the following for each sample
- Cp call
- Calculated concentration
- Standard concentration
- Interpolated/Extrapolated flag for concentration
- Certain/uncertain flag on the Cp call
- High/Low/Normal flag for the Cp call

In the Tm module, the results will include a flag to indicate manual editing for each sample.

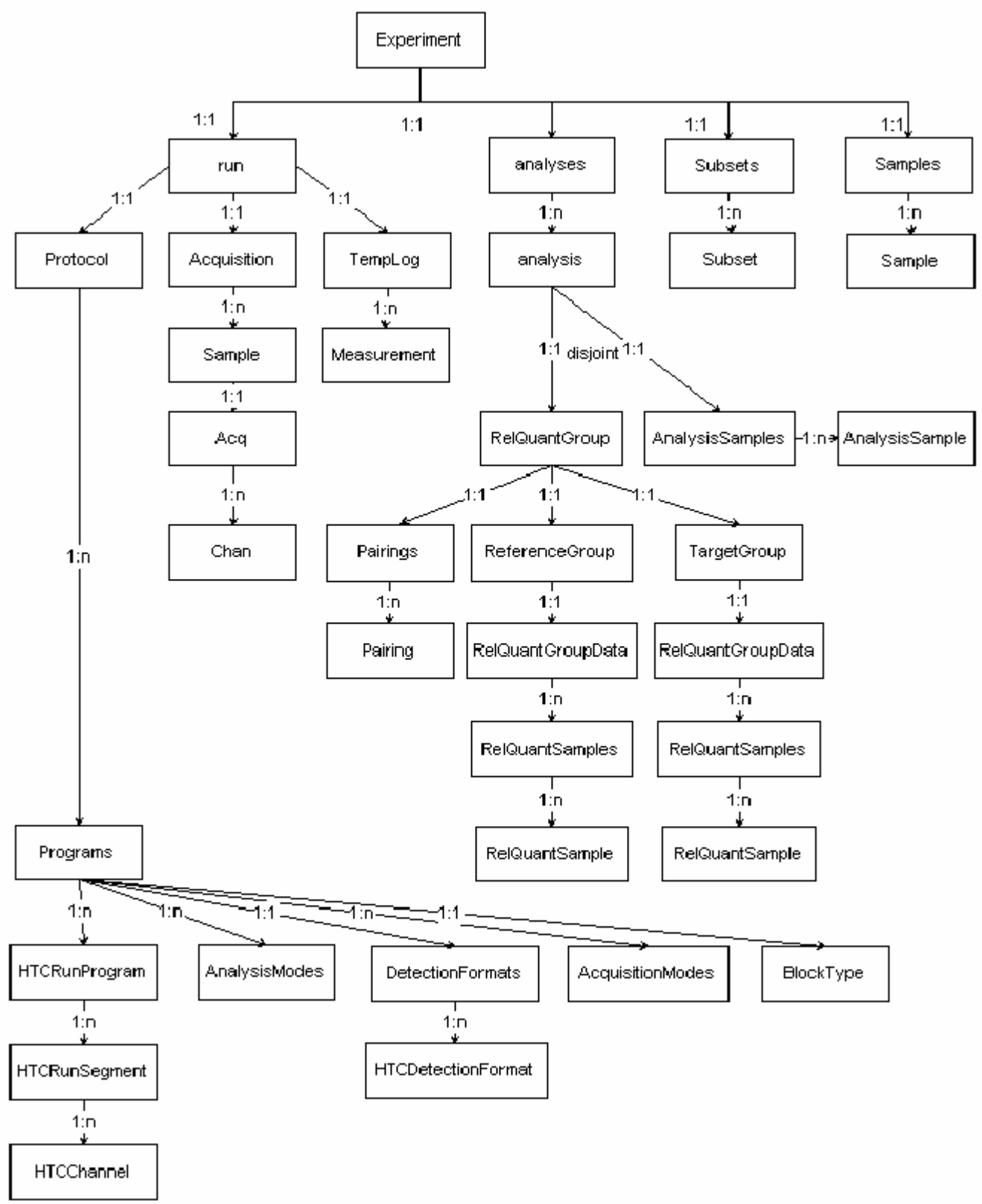
➢ For each sample and for each of the six peaks that may be called in the Tm algorithm, the Tm calling results will include the following:
- Tm
- Area
- Peak
- Width
- Shoulders

➢ Rel Quant results will include the following for result set
- Result set name (this column is empty for calibrators)
- Sample Type, position, and sample name for samples in the results set. Values for sample type are Target Calibrator, Reference Calibrator, Target Unknown, and Reference Unknown.
- Cp call for each sample in the result set and the median Cp for the set.
- Concentration Ratio and error (Concentration ratio of the Calibrators or of the Target and Reference Unknowns in this result set)
- Normalized Ratio and error (The normalized ratio of the Unknowns with the Calibrators for this result set)
- Multiplication/Correction Factor (The multiplication and correction factors for the result set, expressed as a fraction.)

➢Genotyping results from LIMS will include the following for each sample:
- Group Name
- Score
- Resolution
- A flag that indicates manual editing

➢LIMS shall return the following for each sample in a Fit Points analysis:
- Cp call
- Calculated concentration
- Standard concentration
- Interpolated/Extrapolated flag for call and concentration

3.2 Output file schema



3.3 XML-File Elements

3.3.1 Experiment

Experiment is the root element of the XML-file.

- Child Elements:

run
analyses
Subsets
Samples

- Attributes:

Name	String	
Created	Date	yyyy-mm-ddThh:mm:ss.msc
createdByName	String	
LastModified	Date	yyyy-mm-ddThh:mm:ss.msc
LastModifiedByName	String	
SWVersion	String	LCS480 1.0.0.95
RevsComplete	Integer	>0

3.3.1.1 Run

- Child Elements:

Protocol
Acquistion
TempLog

- Attributes:

Name	String	
Created	Date	yyyy-mm-ddThh:mm:ss.msc
CreatedByName	String	
LastModified	Date	yyyy-mm-ddThh:mm:ss.msc
State	String	rscompleted
StartTime	Date	yyyy-mm-ddThh:mm:ss.msc
EndTime	Date	yyyy-mm-ddThh:mm:ss.msc
InstrumentID	Integer	
InstrumentVersion	String	HTC_VER_10

InstrumentName	String	Pilot 517
InstrCalibrationDate	Date	yyyy-mm-dd hh:mm:ss:ms
Technician	String	
Notes	Text	

3.3.1.1.1 Protocol

- Child Elements:

Programs

- Attributes:

Name	Type	Format
class	String	
version	Integer	
created	Date	yyyy-mm-ddThh:mm:ss.msc
last modified	Date	yyyy-mm-ddThh:mm:ss.msc

3.3.1.1.1.1 Programs

- Child Elements:

Emlist	list of HTCRunProgram nodes
AnalysisModes	list of Strings
DetectionFormats	Node with list of HTCDetectionFormat
AcquisitionModes	List of Strings
BlockType	

- Attributes:

class	String	
version	Integer	
ChannelCount	Integer	
InstrumentSubclass	String	
SeekTemp	Integer	
MaxPositionsToSeek	Integer	
SampleVolume	Integer	

re.

- Child Elements:

HTCRunSegment

- Attributes:

version	Integer	
Name	String	
Cycles	Integer	Pre-incubation, Amplification
AnalysisMode	String	Quantification

.

- **Attributes:**

Hold	Integer	
Slope	Float	
StepDelay	Integer	
StepSize	Integer	
Target	Integer	
Target2	Integer	
AcquisitionMode	Integer	
AcqPerDegree	Integer	

- Child Elements:

HTCDetectionFormat	

- Attributes:

class	String	
version	Integer	
DefFormatNdx	Integer	

- Attributes:

class	String
version	Integer
Id	String
RowCount	Integer
ColCount	Integer
OvershootDnDelay	Integer
OvershootUpDelay	Integer
RampRateMaxDn	Float
RampRateMaxUp	Float
ReactionVolMin	Integer
ReactionVolMax	Integer
ReactionVolDefault	Integer
HorVertCrosstalkCoefficient	Float
DiagCrosstalkCoefficent	Float

3.3.1.1.2 Acquisition

3.3.1.1.3 Acquisition

- Child Elements:

Acquisition with child element Sample
Sample with child element Acq
Acq with child element Chan

- Attributes of Chan:

Fluor	float	
Temp	float	
Time	integer	>0

3.3.1.1.4 TempLog

- Child Elements:

Measurement

- Attributes:

Temp	float	
Time	integer	>0

3.3.1.2 Analyses

3.3.1.2.1 Analysis

3.3.1.2.1.1 RelQuantGroup

- Elements:

Pairings	Mapping of ReferenceGroup to TargetGroup
ReferenceGroup	
TargetGroup	

Elements:

RelQuantSamples

.

- Attributes:

name	String	
Position	String	A1, D5
RQSampleIncluded	Boolean	0,1
RQSampleType	String	
RQSampleCrossingPoint	Float	
RQSampleCall	String	pdcPositive
RQSampleConcentration	Float	
Incomplete	integer	
name	String	
Position	String	A1, D5
RQSampleIncluded	Boolean	0,1
RQSampleType	String	

- Elements:

Pairing

- Attributes:

Name	String	
RQResultSetNormalizedRatio	Float	
RQResultSetConcentrationRatio	Float	
RQResultSetCalibratorConcentrationRatio	Float	
RQResultSetState	String	rsPositive
RQResultSetNormalizedRatioError	Float	
RQResultSetConcentrationRatioError	Float	
RQResultSetCalibratorConcentrationRatioError	Float	
RQResultSetMultFactor	Integer	
RQResultSetCorrFactor	Integer	
CalRef	Integer	
CalTarget	Integer	
Ref	Integer	
Target	Integer	
Target Median Cp	Float	
Reference Median Cp	Float	

3.3.1.2.1.2 non RelQuant Analysis

Attributes:

name	String	
Position	String	A1, D5
IsIncluded	Boolean	0,1
CrossingPoint	Float	
CpUncertain	Boolean	0,1
CpState	String	
CalcConc	Float	
StandardConc	Float	
CalcConcUnc	Boolean	0,1

.

- Attributes:

GroupName
Res
Score
ManualGroup

- Attributes:

name	String	
Position	String	A1, D5
IsIncluded	Boolean	0,1
Call	Integer	
TmCount	Integer	
ManualTms	Integer	

- Lists:

Shoulder	Array of float	TM of shoulder if any occur
Tms	Array of float	number of peeks

Amounts	Array of float	value of the area of the peek
Heights	Array of float	height of the peek
Widths	Array of float	width of the peek

Attributes:

name	String	
Position	String	A1, D5
IsIncluded	Boolean	0,1

.

3.3.1.3 Subsets

List of subsets

3.3.1.4 Samples

List of samples

Index

- U -

- V -

- W -

Endnotes

Roche Diagnostics Ltd.
Forrenstrasse
CH-6343 Rotkreuz
Switzerland