Table of Contents

1. OBJECTIVES

1). Send local(Urit equipment) sample result to the remote LIS/HIS Server.

2). Request sample information from the remote LIS/HIS Server.

3). LISHL7Interface.dll can be used for 8030/8060/8021A/8031 and later other series.

2. INTRODUCTION

2.1 Introduction to HL7 version

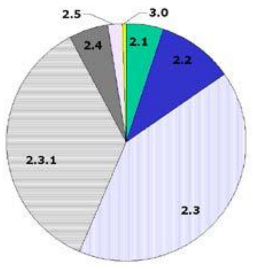


Figure 1: Real-world usage of HL7 messaging standards (approximate). The vast majority of HL7 messaging is done using HL7 2.3 or HL7 2.3.1. Newer releases of HL7 (2.6, 2.7, and 3.0) represent a very small portion of interfaces.

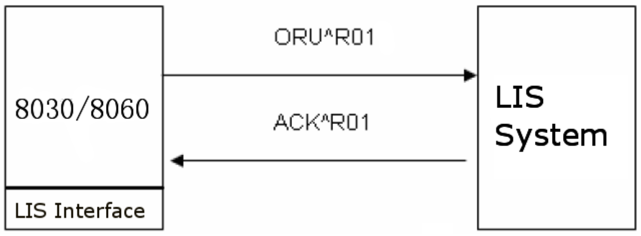
Notes: LISHL7Interface.dll based on HL7 V2.3.1.

2.2 MESSAGE SCOPE

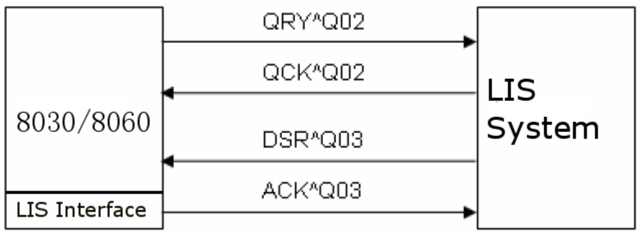
- ✓ ORU
- ✓ ACK
- ✓ QRY
- ✓ QCK
- ✓ DSR

2.3 COMMUNICATION MODEL

Upload:



Download:



2.3.1 Observation Reporting(ORU/ACK)

Used to send sample result to LIS/HIS server.

ORU_R01 - Unsolicited transmission of an observation message

- MSH - MSH - message header segment

- {GROUP} - Patient result
    - [GROUP] - Patient
        - PID - PID - patient identification segment
        - [PD1] - Patient Additional Demographic
        - [{NK1}] - NK1 - next of kin / associated parties segment
        - [{NTE}] - NTE - notes and comments segment
        - [GROUP] - Visit
            - PV1 - PV1 - patient visit segment
            - [PV2] - PV2 - patient visit - additional information segment
    - {GROUP} - Order observation
        - [ORC] - ORC - common order segment
        - OBR - OBR - observation request segment
        - [{NTE}] - NTE - notes and comments segment
        - {GROUP} - Observation
            - [OBX] - OBX - observation/result segment
            - [{NTE}] - NTE - notes and comments segment
        - [{CTI}] - CTI - clinical trial identification segment
- [DSC] - DSC - Continuation pointer segment

ACK - General acknowledgment message

- MSH - MSH - message header segment
- MSA - MSA - message acknowledgment segment
- [ERR] - ERR - error segment

2.3.2 Query(QRY/QCK)

Used to request sample information from the remote LIS/HIS server, request condition can be barcode or time.

QRY_Q02 - Query sent for deferred response

- MSH - MSH - message header segment
- QRD - QRD - original-style query definition segment
- [QRF] - QRF - original style query filter segment
- [DSC] - DSC - Continuation pointer segment

QCK_Q02 - Deferred query

- MSH - MSH - message header segment
- MSA - MSA - message acknowledgment segment
- [ERR] - ERR - error segment
- [QAK] - Query Acknowledgement

2.3.3 Query Response(DSR/ACK)

When the remote server receives the QRY message, the server must acknowledge DSR message to the client. DSR message include sample information, just like barcode, test items and so on.

DSR_Q03 - Deferred response to a query

- MSH - MSH - message header segment
- [MSA] - MSA - message acknowledgment segment
- [ERR] - ERR - error segment
- [QAK] - Query Acknowledgement
- QRD - QRD - original-style query definition segment
- [QRF] - QRF - original style query filter segment
- {DSP} - DSP - display data segment
- [DSC] - DSC - Continuation pointer segment

Notes:

[GROUP]GROUP number can be 0..1;

{GROUP}GROUP number can be 1..unbounded;

[{SFT}]SFT number can be 0..unbounded;

3. Segment

3.1 MSH - Message header segment

Defines the intent, source, destination, and some specifics of the syntax of a message.

Mnemonic	Description	Type	Optionality	Length	Table	Repetition
MSH.1	Field Separator Used to separate fields.	ST	Required	1		No
MSH.2	Encoding Characters Contains component separator, repetition, separator, escape character, and subcomponent separator(^~\&).	ST	Required	4		No
MSH.3	Sending Application Uniquely identifies the sending application(Urit).	HD	Optional	180		No
MSH.4	Sending Facility Uniquely identifies the sending facility(8030/8060/8021A).	HD	Optional	180		No
MSH.5	Receiving Application Uniquely identifies the receiving application. Empty	HD	Optional	180		No
MSH.6	Receiving Facility Uniquely identifies the receiving facility. Empty	HD	Optional	180		No
MSH.7	Date/Time Of Message Date/time when the sending system created the message.	TS	Optional	26		No
MSH.8	Security Used to implement security features.	ST	Optional	40		No

	Empty					
MSH.9	Message Type Contains the message type and trigger event. Used to identify message structure. For example ORU^R01	MSG	Required	7		No
MSH.10	Message Control ID Contains identifier that uniquely identifies the message.	ST	Required	20		No
MSH.11	Processing ID Indicates that message should be processed according to HL7 Application (level 7) Processing rules. Using "P"	PT	Required	3		No
MSH.12	Version ID Identifies message version. Default "2.3.1"	VID	Required	60		No
MSH.13	Sequence Number A non-null value implies that the sequence number protocol is in use. Empty	NM	Optional	15		No
MSH.14	Continuation Pointer Used to define continuations in application-specific ways. Empty	ST	Optional	180		No
MSH.15	Accept Acknowledgment Type Identifies the conditions under which accept acknowledgments are required. Empty	ID	Optional	2	155	No
MSH.16	Application Acknowledgment Type Used in enhanced acknowledgment mode. Using "0"	ID	Optional	2	155	No
MSH.17	Country Code Contains the country of message origin. Empty	ID	Optional	2		No
MSH.18	Character Set Character set used in message. Using ASCII	ID	Optional	16	211	Yes
MSH.19	Principal Language Of Message Principal language used in the message. Empty	CE	Optional	60		No
MSH.20	Alternate Character Set Handling Scheme Alternative character set when special handling required.	ID	Optional	20	356	No

	Empty					

3.2 MSA - Message acknowledgment segment

Contains information sent while acknowledging another message.

Mnemonic	Description	Type	Optionality	Length	Table	Repetition
MSA.1	Acknowledgement Code AA:Accept, AE:Error, AR:Reject	ID	Required	2	8	No
MSA.2	Message Control ID Same as MSH-10	ST	Required	20		No
MSA.3	Text Message Using for error detail, such as ”Message accepted”	ST	Optional	80		No
MSA.4	Expected Sequence Number Empty	NM	Optional	15		No
MSA.5	Delayed Acknowledgment Type Empty	ID	Optional	1	102	No
MSA.6	Error Condition Default “0”	CE	Optional	100		No

3.3 PID - Patient identification segment

Contains patient identifying and demographic information.

Mnemonic	Description	Type	Optionality	Length	Table	Repetition
PID.1	Set ID – PID	SI	Optional	4		No
PID.2	Patient ID sam.InHospitalNo	CX	Optional	20		No
PID.3	Patient Identifier List sam.OutHospitalNo	CX	Required	20		Yes
PID.4	Alternate Patient ID – PID sam.Bed_No	CX	Optional	20		Yes
PID.5	Patient Name sam.Name	XPN	Required	48		Yes
PID.6	Mother s Maiden Name Empty	XPN	Optional	48		Yes
PID.7	Date/Time Of Birth sam.Age	TS	Optional	26		No
PID.8	Sex sam.Sex,M/F/O	IS	Optional	1	1	No
PID.9	Patient Alias Empty	XPN	Optional	48		Yes
PID.10	Race Empty	CE	Optional	80		Yes
PID.11	Patient Address Empty	XAD	Optional	106		Yes
PID.12	County Code, Using as Age unit	IS	Optional	4	289	No

PID.13	Phone Number – Home Empty	XTN	Optional	40		Yes
PID.14	Phone Number – Business Empty	XTN	Optional	40		Yes
PID.15	Primary Language Empty	CE	Optional	60		No
PID.16	Marital Status Empty	CE	Optional	80		No
PID.17	Religion Empty	CE	Optional	80		No
PID.18	Patient Account Number Empty	CX	Optional	20		No
PID.19	SSN Number – Patient Empty	ST	Optional	16		No
PID.20	Driver's License Number – Patient Empty	DLN	Optional	25		No
PID.21	Mother's Identifier Empty	CX	Optional	20		Yes
PID.22	Ethnic Group Empty	CE	Optional	80		Yes
PID.23	Birth Place Empty	ST	Optional	60		No
PID.24	Multiple Birth Indicator Empty	ID	Optional	1	136	No
PID.25	Birth Order Empty	NM	Optional	2		No
PID.26	Citizenship Empty	CE	Optional	80		Yes
PID.27	Veterans Military Status Empty	CE	Optional	60		No
PID.28	Nationality Empty	CE	Optional	80		No
PID.29	Patient Death Date and Time Empty	TS	Optional	26		No
PID.30	Patient Death Indicator Empty	ID	Optional	1	136	No

3.4 OBR - Observation request segment

Used to transmit order information for a diagnostic study or observation.

Mnemonic	**Description**	**Type**	**Optionality**	**Length**	**Table**	**Repetition**
OBR.1	Set ID – OBR	SI	Optional	4		No

OBR.2	Placer Order Number sam.Barcode	EI	Optional	22		No
OBR.3	Filler Order Number sam.ID	EI	Optional	22		No
OBR.4	Universal Service ID Urit^8030/8060/8021A	CE	Required	200		No
OBR.5	Priority-OBR sam.Emergency, Default "N"	ID	Optional	2		No
OBR.6	Requested Date/time Empty	TS	Optional	26		No
OBR.7	Observation Date/Time sam.CheckDate, Check Date	TS	Optional	26		No
OBR.8	Observation End Date/Time Empty	TS	Optional	26		No
OBR.9	Collection Volume * Empty	CQ	Optional	20		No
OBR.10	Collector Identifier * Empty	XCN	Optional	60		Yes
OBR.11	Specimen Action Code * Empty	ID	Optional	1	65	No
OBR.12	Danger Code Empty	CE	Optional	60		No
OBR.13	Relevant Clinical Info. sam.Note	ST	Optional	300		No
OBR.14	Specimen Received Date/Time * Empty	TS	Optional	26		No
OBR.15	Specimen Source sam.SampleType	SPS	Optional	300		No
OBR.16	Ordering Provider sam.Doctor	XCN	Optional	120		Yes
OBR.17	Order Callback Phone Number sam.Deparment	XTN	Optional	40		Yes, less or equal 2times
OBR.18	Placer Field 1 Empty	ST	Optional	60		No
OBR.19	Placer Field 2 Empty	ST	Optional	60		No
OBR.20	Filler Field 1 + Empty	ST	Optional	60		No
OBR.21	Filler Field 2 + Empty	ST	Optional	60		No

OBR.22	Results Rpt/Status Chng - Date/Time + Empty	TS	Optional	26		No
OBR.23	Charge to Practice + Empty	MOC	Optional	40		No
OBR.24	Diagnostic Serv Sect ID Empty	ID	Optional	10	74	No
OBR.25	Result Status + Empty	ID	Optional	1	123	No
OBR.26	Parent Result + Empty	PRL	Optional	200		No
OBR.27	Quantity/Timing Empty	TQ	Optional	200		Yes
OBR.28	Result Copies To Empty	XCN	Optional	150		Yes, less or equal 5times
OBR.29	Parent Number Empty	EIP	Optional	200		No
OBR.30	Transportation Mode Empty	ID	Optional	20	124	No
OBR.31	Reason for Study Empty	CE	Optional	300		Yes
OBR.32	Principal Result Interpreter + Empty	NDL	Optional	200		No
OBR.33	Assistant Result Interpreter + Empty	NDL	Optional	200		Yes
OBR.34	Technician + Empty	NDL	Optional	200		Yes
OBR.35	Transcriptionist + Empty	NDL	Optional	200		Yes
OBR.36	Scheduled Date/Time + Empty	TS	Optional	26		No
OBR.37	Number of Sample Containers * Empty	NM	Optional	4		No
OBR.38	Transport Logistics of Collected Sample * Empty	CE	Optional	60		Yes
OBR.39	Collector s Comment * Empty	CE	Optional	200		Yes
OBR.40	Transport Arrangement Responsibility Empty	CE	Optional	60		No
OBR.41	Transport Arranged Empty	ID	Optional	30	224	No

OBR.42	Escort Required Empty	ID	Optional	1	225	No
OBR.43	Planned Patient Transport Comment Empty	CE	Optional	200		Yes
OBR.44	Procedure Code Empty	CE	Optional	80		No
OBR.45	Procedure Code Modifier Empty	CE	Optional	80		Yes

3.5 OBX - Observation/result segment

Used to transmit observation information or report.

Mnemonic	Description	Type	Optionality	Length	Table	Repetition
OBX.1	Set ID – OBX	SI	Optional	4		No
OBX.2	Value Type Default "NM"	ID	Required	3	125	No
OBX.3	Observation Identifier sam.ItemStru[index].ItemCode	CE	Required	80		No
OBX.4	Observation Sub-ID sam.ItemStru[index].Item	ST	Required	20		No
OBX.5	Observation Value sam.ItemStru[index].Result	VARIES	Optional	65536		Yes
OBX.6	Units sam.ItemStru[index].Unit	CE	Optional	60		No
OBX.7	References Range sam.ItemStru[index].Range	ST	Optional	60		No
OBX.8	Abnormal Flags Default "N"	ID	Optional	5	78	Yes, less or equal 5times
OBX.9	Probability Empty	NM	Optional	5		Yes, less or equal 5times
OBX.10	Nature of Abnormal Test Empty	ID	Optional	2	80	No
OBX.11	Observation Result Status Default "F"	ID	Required	1	85	No
OBX.12	Date Last Obs Normal Values Empty	TS	Optional	26		No
OBX.13	User Defined Access Checks sam.ItemStru[index].Abs	ST	Optional	20		No
OBX.14	Date/Time of the Observation sam.CheckDate	TS	Optional	26		No
OBX.15	Producer's ID Empty	CE	Optional	60		No

OBX.16	Responsible Observer sam.Operator	XCN	Optional	80		Yes
OBX.17	Observation Method Empty	CE	Optional	60		Yes

3.6 QRD - Original-style query definition segment

Mnemonic	Description	Type	Optionality	Length	Table	Repetition
QRD.1	Query Date/Time	TS	Required	26		No
QRD.2	Query Format Code record-oriented format, default "R"	ID	Required	1	106	No
QRD.3	Query Priority Default "D"	ID	Required	1	91	No
QRD.4	Query ID	ST	Required	10		No
QRD.5	Deferred Response Type Empty	ID	Optional	1	107	No
QRD.6	Deferred Response Date/Time Empty	TS	Optional	26		No
QRD.7	Quantity Limited Request Empty	CQ	Required	10		No
QRD.8	Who Subject Filter BarCode	XCN	Required	60		Yes
QRD.9	What Subject Filter Default "OTH"	CE	Required	60		Yes
QRD.10	What Department Data Code Empty	CE	Required	60		Yes
QRD.11	What Data Code Value Qual. Empty	VR	Optional	20		Yes
QRD.12	Query Results Level Default "T"	ID	Optional	1	108	No

3.7 QRF - Original style query filter segment

Mnemonic	Description	Type	Optionality	Length	Table	Repetition
QRF.1	Where Subject Filter 8030/8060/8021A	ST	Required	20		Yes
QRF.2	When Data Start Date/Time QueryStartTime	TS	Optional	26		No
QRF.3	When Data End Date/Time QueryEndTime	TS	Optional	26		No
QRF.4	What User Qualifier Empty	ST	Optional	60		Yes
QRF.5	Other QRY Subject Filter Empty	ST	Optional	60		Yes

QRF.6	Which Date/Time Qualifier Default “RCT”（Specimen receipt date/time, receipt of specimen in filling ancillary (Lab)）	ID	Optional	12	156	Yes
QRF.7	Which Date/Time Status Qualifier Default “COR”（Corrected only (no final with corrections)）	ID	Optional	12	157	Yes
QRF.8	Date/Time Selection Qualifier Default “ALL”（All values within the range）	ID	Optional	12	158	Yes
QRF.9	When Quantity/Timing Qualifier Empty	TQ	Optional	60		No

3.8 ERR - Error segment

Mnemonic	Description	Type	Optionality	Length	Table	Repetition
ERR.1	Error Code and Location Default “0”	ELD	Required	80		Yes

3.9 QAK - Query Acknowledgement

Mnemonic	Description	Type	Optionality	Length	Table	Repetition
QAK.1	Query Tag Default “SR”	ST	Optional	32		No
QAK.2	Query Response Status OK：Data found, no errors NF：No data found, no errors AE：Application error AR：Application reject	ID	Optional	2	208	No

3.10 DSP - Display data segment

Mnemonic	Description	Type	Optionality	Length	Table	Repetition
DSP.1	Set ID – DSP	SI	Optional	4		No
DSP.2	Display Level Empty	SI	Optional	4		No
DSP.3	Data Line Member of HL7SamInfo Data Struct, such as sam.Barcode , sam.ItemStru[i].Item	TX	Required	300		No
DSP.4	Logical Break Point Empty	ST	Optional	2		No
DSP.5	Result ID Empty	TX	Optional	20		No

3.11 DSC - Continuation pointer segment

DSC mark the DSR^Q03 message index, DSC|-1| means that it’s the last DSR^Q03 message.

Mnemonic	Description	Type	Optionality	Length	Table	Repetition
DSC.1	Continuation Pointer	ST	Optional	180		No

4. Urit HL7 LIS APP Dev environment

VC++, thirdparty library(tinyxml.lib, LISHL7Interface.lib)

5. Application DataStruct

```
typedef   struct _HL7SamInfo
{
        char        ID[31];                //Sample ID;
        char        Barcode[31];           //Barcode
        char        SampleType[21];        //Sample Type
        char        Name[50];              //Patient Name
        char        Sex[10];               //Sex
        int         Age;                   //Age
        int         Age_unit;              //Age Unit
        char        InHospitalNo[50];      //InHospital No.
        char        OutHospitalNo[50];     //OutHospital NO.
        char        Bed_No[50];            //Bed No.
        char        Deparment[50];         //Department
        char        Doctor[30];            //Doctor
        char        Operator[30];          //Operator
        char        Note[50];              // Clinical expression
        int            TotalItemNum;       //Total Item Number
        HL7ItemInfo ItemStru[50];          //Item Result
        char   CheckDate[20];              //Check Date
        char   Emergency[2];               //STAT or not
} HL7SamInfo;

typedef struct _HL7ItemInfo
{
        char        ItemCode[20];        //Item Index
        char        Item[50];            //Item Name
        char        Abs[10];             //abs
        char        Result[13];          //Result
        char        Unit[20];            //Result Unit
        char   Range[24];                //Item Result Range
}HL7ItemInfo;
```

6. Minimal lower layout protocol(MLLP)

Minimal Lower Layer Protocol (MLLP) defines the leading and trailing delimiters for an HL7 message. These delimiters help the receiving application to determine the start and end of an HL7 message that uses Internet Protocol network as transport.

Message format as follow:

<SB> *ddddd* <EB><CR>

<SB> = Start Block character (1 byte), char 0x0B

ddddd = Data (variable number of bytes)

<EB> = End Block character (1 byte), char 0x1C

<CR> = Carriage Return (1 byte), char 0x0D

7. Main API of LISHL7Interface.dll

7.1 ORU_R01SAMmsgHL7

const char* ORU_R01SAMmsgHL7(HL7SamInfo sam, int SetID);

ORU^R01 message include segment-MSH,PID,OBR,OBX(multi).

Param:

sam—Sample Struct

SetID—Set ID

Return:

ORU_R01 fromat message.

Example:

```
CHL7XML *pHL7 = new CHL7XML;
char *msg = (char *)( pHL7 ->ORU_R01SAMmsgHL7(sam, 1));
pSocketHL7->SendData(msg);
pHL7 ->HL7XMLDelete(msg);
```

```
<SB>MSH|^~\&|urit|8030|||20120830103931||ORU^R01|1|P|2.3.1||||0||ASCII|||<CR>
PID|1||||null||0|M|||||||||||||||||||||||<CR>
OBR|1|null|201208290001|urit^8030|N||2012-08-29||||||||other0|||||||||||||||||||||||||||||||<CR>
OBX|1|NM|1|ALB|11.8|g/l|35.0-55.0|N|||F||0.3279|2012-08-29||Server||<CR>
OBX|2|NM|2|APOA_1|1.43|g/L|0.73-1.69|N|||F||0.3767|2012-08-29||Server||<CR>
OBX|3|NM|3|LDL_C|4.47|mmol/L|2.07-3.10|N|||F||0.7833|2012-08-29||Server||<CR>
OBX|4|NM|4|GGT|7939|U/L|0-50|N|||F||-7.0474|2012-08-29||Server||<CR>
<EB><CR>
```

7.2 ACKmsgHL7

```
const char* ACKmsgHL7(const char *TriggerEvent,
                      char *MessageControlID,
                      char *StatusConfirmCode,
                      char *StatusText,
                      int ErrorCode);
```

ACK^Q03, ACK^R01 include segment-MSH, MSA.

Param:

TriggerEvent—Such as Q03;

MessageControlID—Set as Sample ID;

StatusConfirmCode—acknowledgement status;

StatusText—acknowledgement text;

ErrorCode—default 0;

Return:

TriggerEvent Message;

Example:

```
CHL7XML *pHL7 = new CHL7XML;
char *ack = (char *)(pHL7->ACKmsgHL7("Q03", sam->ID, MSA_AA, "Message accepted", 0));
pSocketHL7->SendData(ack);
pHL7->HL7XMLDelete(ack);
```

```
<SB>MSH|^~\&|urit|8030|||20120830105821||ACK^Q03|1|P|2.3.1||||0||ASCII|||<CR>
MSA|AA|2|Message accepted|||0|<CR>
<EB><CR>
```

```
<SB>MSH|^~\&|urit|8030|||20120830103931||ACK^R01|1|P|2.3.1||||0||ASCII|||<CR>
MSA|AA|2|Message accepted|||0|<CR>
<EB><CR>
```

7.3 QRY_Q02msgHL7

```
const char*    QRY_Q02msgHL7(int iQueryID,
                             char *BarCode,
                             const char *QueryStartTime,
                             const char *QueryEndTime,
                             const char *MessageControlID);
```

QRY^Q02 message include segment-MSH, QRD, QRF.

Param:

iQueryID—Query index ID;

BarCode—Query barcode

QueryStartTime—Query Start Time

QueryEndTime—Query End Time

MessageControlID—Set as Curtime;

Return:

QRY_Q02 message

Example:

```
CHL7XML *pHL7 = new CHL7XML;
char *msg = (char *) pHL7 ->QRY_Q02msgHL7(iQueryID++, (LPTSTR)(LPCTSTR)strHL7BarCode, starTime,
strEndTime, curtime);
TryToSendMsg(msg);
pHL7 ->HL7XMLDelete(msg);
```

```
<SB>MSH|^~\&|urit|8030|||20120830104844||QRY^Q02|1|P|2.3.1||||0||ASCII|||<CR>
QRD|20120830104844|R|D|14|||RD||OTH|||T|<CR>
QRF|8030|20120821000000|20120821235959|||RCT|COR|ALL||<CR>
<EB><CR>
```

7.4 QCK_Q02msgHL7

```
const char* QCK_Q02msgHL7(const char *TriggerEvent,
                          const char *MessageControlID,
                                    const char *QakStatus);
```

QCK^Q02 message include segment-MSH, MSA, ERR, QAK.

Param:

TriggerEvent—Q02;

MessageControlID—Message ID

QakStatus—Query Status:

```
"OK"        //Get Data, no error
"NF"        //No Data, No Error
"AE"        //Application Error
"AR"        //Application Reject
```

Return:

QCK_Q02 message

Example:

```
CHL7XML *pHL7 = new CHL7XML;
pHL7->GetQRYQ02HL7(msg,QRYControlID,QryBarCode,QryStartTime,QryEndTime);
const char *Msg = pHL7->QCK_Q02msgHL7("Q02", QRYControlID.c_str(), QAK_OK);
SendData(Msg);

<SB>MSH|^~\&|urit|8030|||20120830104844||QCK^Q02|1|P|2.3.1||||0||ASCII|||<CR>
MSA|AA|2|Message accepted|||0|<CR>
ERR|0|<CR>
QAK|SR|OK|<CR>
<EB><CR>
```

7.5 DSR_Q03msgHL7

```
const char* DSR_Q03msgHL7(int iDSRindex,
                                    const char *QakStatus,
                                    char *BarCode,
                                    HL7SamInfo sam,
                                    const char *QueryStartTime,
                                    const char *QueryEndTime);
```

DSR^Q03 message include segment-MSH, MSA, ERR, QAK, QRD, QRF, DSP(multi), DSC. Each sample occupy a DSR^Q03 message. When the server send multi sample once a time, we use *"DSC|index|<CR>"* to represent different sample.

"DSC|-1|<CR>" means the last message.

Param:

iDSRindex—different DSR message

QakStatus—query result status

BarCode—sample barcode

Sam—sample datastruct

QueryStartTime—query start time

QueryEndTime—query end time

Return:

DSR^Q03 message

Example:

```
CHL7XML *pHL7 = new CHL7XML;
const char *Msg = pHL7->DSR_Q03msgHL7(-1, QAK_OK, m_CurIndexT.Barcode, m_CurIndexT,
"20120821000000", "20120821235959");
```

```
<SB>MSH|^~\&|urit|8030|||20120830104845||DSR^Q03|1|P|2.3.1||||0||ASCII|||<CR>
MSA|AA|2|Message accepted|||0|<CR>
ERR|0|<CR>
QAK|SR|OK|<CR>
QRD|20120830104845|R|D|11|||RD|1111|OTH|||T|<CR>
QRF|8030|20120821000000|20120821235959|||RCT|COR|ALL||<CR>
DSP|1||201208210001|||<CR>
DSP|2||1111|||<CR>
DSP|3||other0|||<CR>
DSP|4|||||<CR>
DSP|5|||||<CR>
DSP|6||0|||<CR>
DSP|7||0|||<CR>
DSP|8|||||<CR>
DSP|9|||||<CR>
DSP|10|||||<CR>
DSP|11||Laboratory|||<CR>
DSP|12||Server|||<CR>
DSP|13||123|||<CR>
DSP|14|||||<CR>
DSP|15||2012-08-21|||<CR>
DSP|16||N|||<CR>
DSP|17||7|||<CR>
DSP|18||1^ALB^-0.2779^10.0^g/l^35.0-55.0|||<CR>
DSP|19||2^TP^0.3927^44.5^g/l^60.0-85.0|||<CR>
DSP|20||3^GLU^0.0340^0.54^mmol/L^3.90-6.10|||<CR>
DSP|21||4^GGT^-5.7496^6477^U/L^0-50|||<CR>
DSP|22||5^LDH^-8.6731^45504^UL/L^114-240|||<CR>
DSP|23||6^A/G^^0.29^^0.00-10.00|||<CR>
DSP|24||7^GLB^^34.5^g/L^0.0-45.0|||<CR>
DSC|-1|<CR>
<EB><CR>
```

DSP segment definition:

```
DSP|1||Sam ID|||<CR>
DSP|2||Sam Barcode|||<CR>
DSP|3||Sam Type|||<CR>
```

DSP|4||Sam Name|||*<CR>*

DSP|5||Sam Sex|||*<CR>*

DSP|6||Sam Age|||*<CR>*

DSP|7||Sam Age Unit|||*<CR>*

DSP|8||Sam In Hospital No.||| *<CR>*

DSP|9||Sam Out Hospital No.||| *<CR>*

DSP|10||Sam Bed No.||| *<CR>*

DSP|1|1|Sam Department|||*<CR>*

DSP|12||Sam Doctor|||*<CR>*

DSP|13||Sam Operator|||*<CR>*

DSP|14||Sam Clinical Diagnosis|||*<CR>*

DSP|15||Sam Check Date|||*<CR>*

DSP|16||Sam Is Stat Or Not|||*<CR>*

DSP|17||Sam Total Item Number|||*<CR>*

DSP|18||ItemIndex1^ItemName1^ItemAbs1^ItemResult1^ItemResultUnit1^ItemRange1|||<CR>

DSP|19||ItemIndex2^ItemName2^ItemAbs2^ItemResult2^ItemResultUnit2^ItemRange2|||<CR>

7.6 ParseHL7

```
bool ParseHL7(const char *msg,
                HL7SamInfo &sam,
                int &iDsrEnd);
```

Parse ORU^R01 or DSR^Q03 message to HL7SamInfo struct. It is the inverse process of function -*ORU_R01SAMmsgHL7* or *DSR_Q03msgHL7*.

Param:

msg—Just for ORU^R01 and DSR^Q03 message
sam—Sample info after parsing the message
iDsrEnd—Only used in DSR message, when iDsrEnd = -1 means we had met the end of DSR message.

Return:

If it is not DSR^Q03 message, return false;

If it is DSR^Q03 message and parses success, then return true;

7.7 GetQRYQ02HL7

```
void GetQRYQ02HL7(char *Msg,
                std::string &MessageControlID,
                std::string &BarCode,
                std::string &QueryStartTime,
                std::string &QueryEndTime);
```

Param:

Msg—QRY^Q02 Message;
BarCode—Query Barcode, Used to determine the query mode.

QueryStartTime—Query Starttime

QueryEndTime—Query Endtime

8. Introduction to LISHL7Interface.dll Algorithm

8.1 Server side

Used the follow interface:

const char* **GetHL7MessageType**(const char *msg);

GetQRYQ02HL7(char *Msg, std::string &MessageControlID, std::string &BarCode, std::string &QueryStartTime,

std::string &QueryEndTime);

QCK_Q02msgHL7(const char *TriggerEvent, const char *MessageControlID, const char *QakStatus);

DSR_Q03msgHL7(int iDSRindex, const char *QakStatus, char *BarCode, HL7SamInfo sam, const char *QueryStartTime, const char *QueryEndTime);

Pseudo code:

```
Messagetype = GetHL7MessageType(g_HL7Msg.GetBuffer(0));
if (messagetype is ORU^R01)
{
    Acknowledgement for ORU^R01
    ACKmsgHL7("R01", sam->ID, MSA_AA, "Message accepted", 0))
    ParseHL7((char *)(LPCSTR)g_HL7Msg, *sam, iDsc);
    //After ParseHL7, we got the sample info-“sam”. Server side can save the sample data to centre database.
}
else if (messagetype is QRY^Q02)
{
    Acknowledgement for DSR^Q03
    if (QryBarCode.length() == 0 && QryStartTime.length() > 0)
    {
        //query by time.
        //The LIS/HIS server may query the centre database by time.
        If (query is ok)
        {
            QCK_Q02msgHL7("Q02", QRYControlID.c_str(), QAK_OK);
            SendMessage(DSR^Q03, query result-HL7Saminfo);
        }
        else
        {
            QCK_Q02msgHL7("Q02", QRYControlID.c_str(), QAK_NF);
        }
    }
    else if (QryBarCode.length() > 0 && QryStartTime.length() == 0)
    {
        //query by barcode.
        //The LIS/HIS server may query the centre database by barcode.
        If (query is ok)
        {
            QCK_Q02msgHL7("Q02", QRYControlID.c_str(), QAK_OK);
            SendMessage(DSR^Q03, query result-HL7Saminfo);
        }
        else
        {
            QCK_Q02msgHL7("Q02", QRYControlID.c_str(), QAK_NF);
        }
        }
        else
        {
                //query by barcode and time.
                //The LIS/HIS server may query centre database by barcode and time.
                If (query is ok)
                {
                        QCK_Q02msgHL7("Q02", QRYControlID.c_str(), QAK_OK);
                        SendMessage(DSR^Q03, query result-HL7Saminfo);
                }
                else
                {
                        QCK_Q02msgHL7("Q02", QRYControlID.c_str(), QAK_NF);
                }
        }
}
```

8.2 Client side

Used the follow interface:

```
const char*   QRY_Q02msgHL7(int iQueryID, char *BarCode, const char *QueryStartTime, const char *QueryEndTime, const char *MessageControlID);
const char* ORU_R01SAMmsgHL7(HL7SamInfo sam, int SetID);
const char* GetHL7MessageType(const char *msg);
void GetACKHL7(char *Msg, char *MessageControlID, char *StatusConfirmCode, char *StatusText, int &ErrorCode);    //ACK^R01
void GetQAKHL7(char *Msg, std::string &MessageControlID, std::string &QakStatus);    //QCK^Q02
```

Pseudo code:

Query message:

```
OnBnClickedBtnTaskLoad()
{
        if (query mode is on way)
        {
                //one way can not exec query.
                Return;
        }
        else If (query mode is two way)
        {
                If (system is run, but not get sample info in realtime)
                {
                        Return;
                }
        }
        Query by barcode, time, barcode&&time
        QRY_Q02msgHL7(iQueryID++, (LPTSTR)(LPCTSTR)strHL7BarCode, starTime, strEndTime, curtime);
}
```

Send message:

Get HL7SampleInfo

ORU_R01SAMmsgHL7(*sam, index))

Receive message:

```
Messagetype = GetHL7MessageType(pSocketHL7->m_ReceiveMsg.GetBuffer(0))
if (messagetype is ACK^R01)
{
    GetACKHL7(char *Msg, char *MessageControlID, char *StatusConfirmCode, char *StatusText, int
&ErrorCode);
    Mark the sample that had been sent.
}

If (query mode is two way)
{
    If (system is run, but not get sample info in realtime)
    {
        Return;
    }
    if (messagetype is QCK^Q02)
    {
        GetQAKHL7(m_ReceiveMsg, MessageControlID, status);
        //get query status, success or not.
    }
    if (messagetype is DSR^Q03)
    {
        ParseHL7(m_ReceiveMsg.GetBuffer(0), *sam, iDsc);
        if (iDsc == -1)    //last dsr message
        {
            //add the sample list to cache list(grid).
        }
        ACKmsgHL7("Q03", sam->ID, MSA_AA, "Message accepted", 0));
    }
}
```