# C-DAC's Medical Informatics SDK Suite v3.1
# SDK for HL7 v2.8.2 Tutorials

## Introduction to SDK for HL7

C-DAC's Medical Informatics Standards Software Development Kit for HL7 is a toolkit that provides APIs for applications/ medical devices to comply with ANSI/HL7 V2.8.2-2015 Standard. It is a rapid application development tool which provides high return on investment through cost effective implementation of the standard.

HL7 is an ANSI accredited standard which was introduced in 1987 by Health Level Seven organization to overcome the interoperability issues in telemedicine communication. HL7 defines structure for messages and auxiliary protocols to support telemedicine communication in the form of event notification and query/response.

Achieving compliance to HL7 standard, for healthcare application, includes overheads because first it requires a complete knowledge of protocol specifications. Building such a capability from scratch is time consuming since implementer first need to understand the complexities of standard.

HL7 SDK supports all messages and segments belonging to different health care systems which are defined in HL7 v2.8.2 for e.g. Patient Admin System, Financial System etc. Along with different systems HL7 SDK supports different auxiliary protocols which are defined in standard like Batch Protocol, Message Continuation, and Query Cancellation etc. All components of HL7 SDK are available with proper customization and can be enhanced as per requirements.

Back to top

## Features of SDK for HL7

- Easy to use Object-Oriented implementation of ANSI approved HL7 v2.8.2 standard
- Allows customization or extension by implementing provided interfaces
- Comprehensive Error/Warning Logging capability to assist debugging
- Supports all messages and segments belonging to different systems defined by HL7 v2.8.2. Systems supported by HL7 SDK are listed below:
    - Patient Administration
    - Financial Management
    - Observation Reporting
    - Master Files
    - Medical Records
    - Scheduling
    - Patient Referral
    - Patient Care
    - Clinical Laboratory Automation
    - Application Management
    - Personal Management
    - Order Entry
    - Query

- Claims and Reimbursement
- Material Management

- Supports different auxiliary and special protocols defined by HL7 v2.8.2
  - Local Extension for Z-Message and Z-Segment
  - Batch Protocol
  - Message Continuation Protocol
  - Query Cancellation Protocol
  - Query Interactive Continuation Protocol
  - Publish Subscribe Protocol
  - Sequence Number Protocol

- Set of API Documentation
- Sample test codes along with the sample test data to demonstrate the capabilities of SDK
- Ready to run Command Line Utilities
- Source Code of SDK and also for Utilities
- Readme and Help document to give assistance to the user while using SDK

Back to top

## How to enable logging?

Log can be generated in HL7 SDK for different events occurred while processing.
The HL7 SDK describes two levels of logging:

SEVERE - At this level of logging any exception's stack trace is completely logged into a log file. Along with the stack trace the data that are failed are also logged.

INFO - The failed data elements with status WARNING are logged in the log file.

The logging is done only when the logging mode in ON. Logging can be enabled by calling enableLogging () method available in *HL7Config class*. This logging configuration is required only once in SDK life time. Once logging is enabled, log will be generated for all events which are occurred after this configuration. Ideally this should be the first statement before using HL7 SDK.

For e.g.:

```
// log files will be created in 'C:\HL7_Logs' directory.
HL7Config.createInstance ().enableLogging ("C:\\HL7_Logs");

// If no directory location is defined by user then user's
//temp directory will be taken as default.
HL7Config. createInstance ().enableLogging ();
```

Back to top

## HL7 Buffer Stream

*HL7BufferStream* is to perform reading/writing operations on the byte array. Abstract class *HL7Stream* is parent of all stream classes used in HL7 SDK. Applications that need to define the subclass of *HL7Stream* must set either the input stream or the output stream before start using the class.

For e.g.:

Initializing HL7BufferStream for reading operation:

```
HL7BufferStream objHL7BufferStream = new HL7BufferStream ();
ByteArrayInputStream bis = new ByteArrayInputStream (buffer);
objHL7BufferStream.setInputStream (bis);
```

Where, buffer is the byte array.

Initializing HL7BufferStream for writing operation:

```
HL7BufferStream objHL7BufferStream = new HL7BufferStream ();
ByteArrayOutputStream bos = new ByteArrayOutputStream ();
objHL7BufferStream.setOutputStream (bos);
```

Back to top

## HL7 File Stream

*HL7FileStream* is to perform reading/writing operations on the file. Abstract class *HL7Stream* is parent of all stream classes used in HL7 SDK. Applications that need to define the subclass of HL7Stream must set either the input stream or the output stream before start using the class.

Initializing HL7FileStream for reading from file:

```
String strFilePath = "C:\ABC.HL7";
HL7FileStream hfs = new HL7FileStream ();
FileInputStream fis = new FileInputStream(strFilePath);
hfs.setInputStream(fis);
Initializing HL7FileStream for writing to file:
String strFilePath = "C:\ABC.HL7";
HL7FileStream hfs = new HL7FileStream ();
FileOutputStream fos = new FileOutputStream(strFilePath);
hfs.setOutputStream(fos);
```

Back to top

## Different operation with Streams

Once *HL7BufferStream* or *HL7FileStream* is initialized for reading or writing, all operations can be performed on it in the similar way, as it is available with basic streams.

For e.g.:

```
HL7BufferStream objHL7BufferStream = new HL7BufferStream ();
ByteArrayInputStream bis = new ByteArrayInputStream (buffer);
objHL7BufferStream.setInputStream (bis);

//To check availability
objHL7BufferStream.available();

//To close the stream
objHL7BufferStream.close();
```

Refer API docs to check different operations available with *HL7Stream*.

Back to top

## Data Types defined by HL7 v2.8.2

HL7 defines data types which work as basic building block to construct or restrict the contents of data fields of a segment. HL7 SDK supports all data types which are defined by the HL7 v2.8.2 standard. The library contains object-oriented classes for these data types which facilitate easy creation, validation, read – write mechanisms for these data types. HL7 standard categorizes data types in primitive and composite forms. HL7 SDK supports all primitive and composite data types defined by HL7 v2.8.2.

Back to top

## Primitive data types

Primitive data types defines a specific value and does not include component or sub component. Primitive data type values do not share state with other primitive data type values. Primitive data types supported by the HL7 SDK are: DT, DTM, FT, GTS, ID, IS, NM, SI, SNM, ST, TM, TX and UN.

Primitive data types can be created through provided data type class according to object oriented programming manner.

For e.g.:

```
//Initialization of DT data type
DT objDT = new DT(iMinLength, iMaxLength, strCLength, iTableNo);
```

Note: - iMinLength defines minimum length allowed for this data type as per defined in standard, iMaxLength defines maximum length allowed for this data type as per defined in standard, strCLength defines Conformance length allowed for this data type as per defined in standard. iTableNo defines Table number to which the value for the component is specified.

```
//Setting value to data type
objDT.setValue (strvalue);
```

Note: - strValue defines data type value.

Similarly other primitive data types can be used. Refer API docs of specific data type as per requirements.

Back to top

## Composite data types

Composite data types are data types which includes primitive data types and other composite types. These data types includes components and sub components. The act of constructing a composite type is known as composition. HL7 SDK supports all composite data types defined by HL7 v2.8.2 standard. For e.g. AD, CWE, CNE etc.

Composite data types can be created through provided data type class according to object oriented programming manner.

For e.g.:

```
//Initializing HD data type
HD objHD= new HD (iMinLength, iMaxLength, strCLength, iComponentType);
```

Note: - iMinLength defines minimum length allowed for this data type as per defined in standard, iMaxLength defines maximum length allowed for this data type as per defined in standard, strCLength defines Conformance length allowed for this data type as per defined in standard. iComponentType defines whether this data type is at component level or sub component level. For component and sub component level constants has been defined in HL7Constants class.
HL7Constants.HL7_COMPONENT;
HL7Constants.HL7_SUBCOMPONENT;
These constants can be used to define iComponentType.

```
//Setting value to data type
objHD.setNamespaceId(strNamespaceId);
objHD.setUniversalId(strUniversalId);
objHD.setUniversalIdType(strUniversalIdType);
```

Note: - strNamespaceId specifies the HL7 identifier for the user-defined table of values for this component. strUniversalId specifies a string formatted according to the scheme defined by the component. strUniversalIdType governs the interpretation of the strUniversalId. HD data type is constructed through three primitive data types.

Similarly other data types can be created. Refer API docs of specific data type as per requirements.

Back to top

## Length Constraints

Data type in HL7 contains length constraints. HL7 SDK follows these constraints as defined in HL7 v2.8.2 standard. Length of a data type varies in different segments.

**Normative Length** –

For some fields or components, the value domain of the content leads to clearly established boundaries for minimum and/or maximum length of the content. In these cases, these known limits are specified for the item. Normative lengths are only specified for primitive data types.

**Length & Persistent Data Stores –**

For many fields or components, the value domain of the content does not lead to clearly established boundaries for minimum and/or maximum length of the content. In many cases, systems store the information of these value domains using data storage mechanisms that have fixed lengths, such as relational databases, and must impose a limitation on the amount of information that may be stored. Though this does not directly impact on the length of the item in the instance, nevertheless the storage length has great significance for establishing interoperability.

**Truncation Pattern –**

For technical and/or architectural reasons, many applications must define a limit to the length that they will store for a particular item. This creates a need for the length of an element to be defined somewhere and raises the question of what should happen if a real world value is longer than the acceptable value. It can be handeled in two ways, First, either the message cannot be constructed

or must be rejected completely , and Second, for some data items such as names and addresses, this is generally unwelcome information – the system can still function to some degree in the presence of truncated data. However truncation of data may have later consequences. For this reason, when values are truncated because they are too long, the value should be truncated at N-1, where N is the length limit, and the final character replaced with a special truncation character. This means that whenever that value is subsequently processed later, either by the system, a different system, or a human user, the fact that the value has been truncated has been preserved, and the information can be handled accordingly.

The truncation character is not fixed; applications may use any character. The truncation character used in the message is defined in MSH-2. The default truncation character in a message is # (23), because the character must come from the narrow range of allowed characters in an instance. The truncation character only represents truncation when it appears as the last character of a truncatable field. It SHALL be escaped if the last character of the data that is the maximum allowable size for the component is the truncation character.

Example:

For a field with a conformance length of 5 where the content is |1234#| the truncation character is not representing truncation, it is the actual data.

**Conformance Length –**

If populated, the conformance length column specifies the minimum length that applications must be able to store. Conformant applications SHALL NOT truncate a value that is shorter than the length specified. The conformance length is also the minimum value that maybe assigned to maximum length in an implementation profile.

In addition, the conformance length may be followed by a "=" or a "#". The "=" denotes the value may never be truncated, and the "#" denotes that the truncation behaviour defined for the data type applies.

Consider the following *AD* data type, which is a composite data type, this data type specifies the address of a person, place or organization.

AD – Address:

| SEQ | LEN | C.LEN | DT | OPT | TBL# | COMPONENT NAME | COMMENTS |
|---|---|---|---|---|---|---|---|
| 1 | | 120# | ST | O | | Street Address | |
| 2 | | 120# | ST | O | | Other Designation | |
| 3 | | 50# | ST | O | | City | |
| 4 | | 50# | ST | O | | State or Province | |
| 5 | | 12= | ST | O | | Zip or Postal Code | |
| 6 | 3..3 | | ID | O | 0399 | Country | |
| 7 | 1..3 | | ID | O | 0190 | Address Type | |
| 8 | | 50# | ST | O | | Other Geographic Designation | |

This Data Type specifies the Street Address has 120# Conformance Length, and the Address Type can have Minimum Length 1 and Maximum Length 3. For the valid values of the data types, see the above data type table, “Address Type” component name has its own defined HL7 value table. While populate it, follow the value specified in the value table.

Back to top

## Parsing and serialization of data types

Parsing and serialization of a data type refers to the reading and writing process respectively. Reading and writing of data types can be done by passing appropriate streams and delimiters definition.

For e.g.:

```
//Reading of data type
DT objDT = new DT(iMinLength, iMaxLength, strCLength, iTableNo);
objDT.read (objIHL7Stream, objIDelimiter);

//Writing of data type
DT objDT = new DT(iMinLength, iMaxLength, strCLength, iTableNo);
objDT.setValue (strvalue);
objDT.write (objIHL7Stream, objIDelimiter);
```

Note: - objIHL7Stream refers to the valid input stream for reading and objIDelimiter defines details of delimiters which can be retrieved during parsing of message header.

Back to top

## DataType Map and Component Item

Each DataType in HL7 is defined by a structure containing a sequence of data fields. HL7 SDK represents this structure in form of *DataTypeMap*.

DataType Map represents structure of a DataType as per defined in HL7 standard. DataType Map contains a list of *ComponentItem*. HL7 SDK represents different data fields of a DataType in form of ComponentItem. Each data field of a Data Type contains few properties like sequence no. , minimum length, maximum length, conformance length, data type, optionality, repeatability, value table number, component name, as it is shown in example below.

AD – Address:

| SEQ | LEN | C.LEN | DT | OPT | TBL# | COMPONENT NAME | COMMENTS |
|---|---|---|---|---|---|---|---|
| 1 | | 120# | ST | O | | Street Address | |
| 2 | | 120# | ST | O | | Other Designation | |
| 3 | | 50# | ST | O | | City | |
| 4 | | 50# | ST | O | | State or Province | |
| 5 | | 12= | ST | O | | Zip or Postal Code | |
| 6 | 3..3 | | ID | O | 0399 | Country | |
| 7 | 1..3 | | ID | O | 0190 | Address Type | |
| 8 | | 50# | ST | O | | Other Geographic Designation | |

Similarly ComponentItem can be populated for all data fields of a DataType. Once ComponentItem of a Data Type is populated, this can be set on DataType.

For e.g.:

```
// populating ComponentItem for first attribute of MSA segment.

DataTypeMap objDataTypeMap = new DataTypeMap();
ComponentItem objComponentItem = new ComponentItem();
objComponentItem.setSequenceNo(7);
objComponentItem.setDataType("ID");
objComponentItem.setFieldName("Address Type");
objComponentItem.setOptional(true);
objComponentItem. setMinimumLength(1);
objComponentItem. setMaximumLength(3);

int [] table = new int[1];
table [0] = 190;
objComponentItem.setTableNo(table);

objDataTypeMap.addComponentItem(objComponentItem);
```

Similarly other attributes can be populated and add on DataTypeMap.

Note: - DataType which are defined by HL7 v2.8.2 need not to be populated using above mechanism because HL7 SDK internally performs this process. This process is required only for locally extended DataType.

For locally extended datatype refer "*Local Extension Protocol*"

Back to top

## Segments defined by HL7 v2.8.2

A segment is a logical grouping of data fields. Segments of a message may be required or optional. They may occur only once in a message or they may be allowed to repeat. Each segment is given a name. For example, the ADT message may contain the following segments: Message Header (MSH), Event Type (EVN), Patient ID (PID), and Patient Visit (PV1). Each segment is identified by a unique three-character code known as the Segment ID. All segment ID codes beginning with the letter Z are reserved for locally defined segments.

For locally defined segments refer "*Local Extension Protocol*".

HL7 SDK supports all segments defined by HL7 v2.8.2.

Back to top

## Segment Map and Attribute Item

Each Segment in HL7 is defined by a structure containing a sequence of data fields. HL7 SDK represents this segment structure in form of *SegmentMap*.

SegmentMap represents structure of a segment as per defined in HL7 standard. SegmentMap contains a list of *AttributeItem*. HL7 SDK represents different data fields of a segment in form of AttributeItem. Each data field of a segment contains few properties like sequence no. , length , data type , optionality , repeatability , value table number , item id and element name , as it is shown in example below.

HL7 Attribute Table - MSA - Message Acknowledgment

| SEQ | LEN | C.LEN | DT | OPT | RP/# | TBL# | ITEM # | ELEMENT NAME |
|---|---|---|---|---|---|---|---|---|
| 1 | 2..2 | | ID | R | | 0008 | 00018 | Acknowledgment Code |
| 2 | 1..199 | 199= | ST | R | | | 00010 | Message Control ID |
| 3 | | | | W | | | 00020 | Text Message |
| 4 | | | NM | O | | | 00021 | Expected Sequence Number |
| 5 | | | | W | | | 00022 | Delayed Acknowledgment Type |
| 6 | | | | W | | | 00023 | Error Condition |
| 7 | | | NM | O | | | 01827 | Message Waiting Number |
| 8 | 1..1 | | ID | O | | 0520 | 01828 | Message Waiting Priority |

Similarly AttributeItem can be populated for all data fields of a segment. Once AttributeItem of a segment is populated, this can be set on SegmentMap.

For e.g.:

```
// populating AttributeItem for second attribute of MSA segment.

SegmentMap objSegmentMap = new SegmentMap ();
AttributeItem objAttributeItem = new AttributeItem ();
objAttributeItem.setSequenceNo(2);
objAttributeItem.setFieldName("Message Control ID");
objAttributeItem.setItemID(10);
objAttributeItem. setMinimumLength(1);
objAttributeItem. setMaximumLength(199);
objAttributeItem. setConformanceLength("199=");
objAttributeItem.setDataType("ST");
objAttributeItem.setRepeatable(false);
objAttributeItem.setOptional(false);
```

Similarly other attributes can be populated and add on SegmentMap.

Note: - Segments which are defined by HL7 v2.8.2 need not to be populated using above mechanism because HL7 SDK internally performs this process. This process is required only for locally extended segments.

For locally extended segments refer "*Local Extension Protocol*".

Back to top

## Segment creation through SDK

All segments, which are defined by HL7 v2.8.2, are supported by HL7 SDK. Initialization of segment can be done through two ways.

```
MSA objMSA = new MSA ();
Or
MSA objMSA = (MSA) SegmentFactory.createSegment("MSA");
```

Segment Factory:-
Segment Factory represents an instance of a real world Information Object. Segment Factory can be used to create object of HL7 v2.8.2 defined segments by providing name of segment.

For e.g.:

```
ISegment objSegment = SegmentFactory.createSegment("MSA");
MSA objMSA = (MSA)objSegment;
```

Note: - For Z-Segment and unknown segment it returns NULL;

Back to top

## Population of a Segment

Population of a segment can be done either by parsing through stream or manually.

**Segment Population through manual process -**

Create object of segment and set all attributes in the form of data type attributes which are defined by HL7 v2.8.2 standard.

```
MSA objMSA = new MSA ();
objMSA.setAcknowledgmentCode(objID);
objMSA.setMessageControlID(objST);
objMSA.setExpectedSequenceNumber(objNM);
```

Note: - objID, objST and objNM represents data type fields which are defined by MSA segment structure. For primitive data fields user can use available overloaded methods also. Refer API docs.

**Segment population through parsing -**

Create object of segment and populate segment by passing populated input stream and delimiter definition.

```
MSA objMSA = new MSA ();
objMSA.parse (objIHL7Stream, objIDelimiter);
```

Note: - objIHL7Stream defines populated stream and objIDelimiter defines delimiter definition which can be retrieved from populated MSA segment.

Back to top

## Accessing attributes of a Segment

Different fields of a segment can be retrieved by calling appropriate getter method on segment object. Refer API docs for available methods on a specific segment.

```
//writing contents of a segment
objMSA.serialize(objIHL7Stream ,objIDelimiter);
```

Note: - objIHL7Stream defines output stream and objIDelimiter defines delimiter definition.

Back to top

## Validation of a segment

HL7 defines a structure for each segment in the form of description for different attribute fields. Each attribute field consists of properties defining optionality and repeatability constraints. HL7 SDK applies validation rules on a segment based on these constraints. Each segment defines its own validation process which can be used by calling validate() method on Segment.

For e.g.:

```
objMSA.validate();
```

Validation call on segment returns status as Boolean value or throws exception if required. Validation process works as validation modes defined on SDK.

Back to top

## Groups defined by HL7 v2.8.2

Groups are logical collection of segments. Collectively these segments represent meaningful information. For example, Patient Visit Group consists of segments that represent information regarding a patient's visit to a healthcare facility. One group can be specified in any number of messages depending upon the structure of the message.

Back to top

## Group creation through SDK

In order to create a Group instance SDK provides a single generic class named Group which consists of a constructor that accepts the name of the group. Following is a sample code to create a PATIENT VISIT GROUP.

```
IMap objPatientVisitGroupMap = objMessageMap.getGroupMap(EnumSegments.
HL7_GROUP_PATIENT_VISIT);

Group objPatientVisitGroup = new
Group(EnumSegments.HL7_GROUP_PATIENT_VISIT, objPatientVisitGroupMap);
```

This creates an empty Group instance.

Back to top

## Population of a Group

In order to populate a group instance we can either manually add each segment onto the Group instance or read the multiple segments from stream. For understanding segments and its creation, population mechanisms, please refer *HL7 Segments*.

- Population of a Group manually

```
//Create a Segment instance and populate it
PV1 objSegmentPV1 = new PV1 ();
objSegmentPV1.setAdmissionType(objCWE);

//Create a Group instance and Add a Segment instance onto it
objPatientVisitGroup.addSegment(EnumSegments.HL7_SEG_PV1, objSegmentPV1);
```

- Population of a Group from stream

  In order to populate a Group through stream we need a serialized stream of HL7 Segments. The stream can be a File or a ByteArray stream. Following is an example to show parsing of Patient Visit Group.

```
objPatientVisitGroup.parse(objIHL7Stream, objIDelimiter);
```

Note: - objIHL7Stream defines populated stream and objIDelimiter defines delimiter definition which can be retrieved from populated MSH segment

Back to top

## Accessing members of a Group

In order to access different segments of a Group, then there are two ways, either retrieve a collection of all segments of the Group or retrieve only a particular Segment by using a SegmentKey.

- Retrieve a collection of Segments from the Group

```
ISingleCollection<ISegment> objSegmentCollection =
objPatientVisitGroup.getAllSegments();
```

- Retrieve a single segment from the Group

```
PV1 objSegmentPV1 = (PV1)
objPatientVisitGroup.getBySegmentID(EnumSegments.HL7_SEG_PV1);
```

In order to retrieve any other member of the Group instance, please refer to the API Docs for Group Class.

Back to top

## Validation of a Group

HL7 defines a structure of a Group that contains a sequence of segments represented by its conditionality of presence or repeatability. Below is a sample code to validate a Group.

- Retrieve all segments from the Group

```
ISingleCollection <ISegment> objSegmentCollection =
objPatientVisitGroup.getAllSegments();
```

- Validate individual segments

```
for(ISegment objSegment : objSegmentCollection)
{
objSegment.validate();
}
```

Back to top

## Segment - Set

*SegmentSet* represents a list of Segments. It provides required method to operate upon different segments which collectively represents a HL7 Message. It has a list of segment and methods to provide the segment from the list.

Back to top

## HL7 Parser

The *HL7Parser* parses the HL7Message and provides a *SegmentSet*. SegmentSet contains all segments of a HL7 Message in the same order as they appeared.

To parse a HL7 Message, there are two way to parse, first, provide HL7Stream which gives ISegmentSet.

```
String strFilePath = "C:/ABC.HL7";
FileInputStream objFileInputStream = new FileInputStream(strFilePath);
HL7FileStream objHL7FileStream = new HL7FileStream();
objHL7FileStream.setInputStream(objFileInputStream);
ISegmentSet objSegmentSet = parse(objHL7FileStream);
Or provide File path which also gives ISegmentSet.
String strFilePath = "C:/ABC.HL7";
ISegmentSet objSegmentSet = parse(strFilePath);
```

Back to top

## HL7 Serializer

*HL7Serializer* serializes the *SegmentSet* to the stream. SegmentSet is a list of segments which represents a HL7 message.

To Serialize a SegmentSet, there are two way to serialize, first Serializes the SegmentSet on given HL7Stream.

```
String strFilePath = "C:/ABC.HL7";
FileOutputStream objFileOutputStream = new FileOutputStream(strFilePath);
HL7FileStream objHL7FileStream = new HL7FileStream();
objHL7FileStream.setOutputStream(objFileOutputStream);

ISegmentSet objSegmentSet = objMessage.getSegmentSet();
HL7Serializer objHL7Serializer = new HL7Serializer();

objHL7Serializer.serialize(objSegmentSet, objHL7FileStream);
```

Or Serializes the SegmentSet on given file location.

```
String strFilePath = "C:/ABC.HL7";
ISegmentSet objSegmentSet = objMessage.getSegmentSet();
HL7Serializer objHL7Serializer = new HL7Serializer();

objHL7Serializer.serialize(objSegmentSet, objHL7FileStream);
```

Back to top

## Messages defined by HL7 v2.8.2

A *Message* is the atomic unit of data transferred between systems. It is comprised of a group of segments in a defined sequence. Each message has a message type that defines its purpose. For example, the ADT Message type is used to transmit portions of a patient's Patient Administration (ADT) data from one system to another. A three-character code contained within each message identifies its type. The real-world event is called the trigger event. These codes represent values such as a patient is admitted or an order event occurred.

Back to top

## Message Map and Segment Item

*Message Map* class represents a map for Segment Items represent structure of a Message. It implements IMessageMap interface, which provides Structure for a HL7 Message.

*SegmentItem* class represents structure of a Segment. It provides the setters and getters to specify structure for a segment.

```
IMessageMapReader objIMessageMapReader =
MessageMapReader.createInstance(EnumHL7System.HL7_SYSTEM_APPLICATIONM
ANAGEMENT);


IMessageMap obj = objIMessageMapReader.getMessageMap(EnumMessageCode.ADT,
EnumTriggerEvent.A01);
```

Back to top

## Message creation through SDK

```
IMessageSource objQuerySource = new QrySource();
Message objMessage = objQuerySource.createMessage(EnumMessageCode.QBP,
EnumTriggerEvent.Q11);
QBP_Q11 objQBP_Q11 = (QBP_Q11)objMessage;
```

Back to top

## Message Factory

MessageFactory class provides support for creation and population of messages. It creates and populates supported message according to the available list of Source and Recipient. We can also create HL7 messages through Message Factory.

```
MessageFactory objIMessageFactory = MessageFactory.createInstance();
Message objMessage = objIMessageFactory.createSendSupportedMessage("QBP",
"Q11");
QBP_Q11 objQBP_Q11 = (QBP_Q11)objMessage;
```

Back to top

## Population of a Message

User wants to populate a HL7 message, first *create* the Message from specific source or recipient. Then add the populated segment on it.

```
IMessageSource objQuerySource = new QrySource();
Message objMessage = objQuerySource.createMessage(EnumMessageCode.QBP,
EnumTriggerEvent.Q11);
QBP_Q11 objQBP_Q11 = (QBP_Q11)objMessage;
objQBP_Q11.setHeader(objMSH);
objQBP_Q11.addSoftwareSegment(objSFT);
```

Where, objMSH is the MSH segment and objSFT is the SFT segment. Similarly user can add more segments on the message.

Back to top

## Accessing members of a Message

If user wants to access the members of Message, then user have two choices, first user can specify the segment name and get the collection of specified segment,

```
ISingleCollection <ISegment> objCollection =
objQBP_Q11.getSegment(EnumSegments.HL7_SEG_SFT);
```

Or user can get specified segment, if user knows the which method gives specific segment, like,

```
objQBP_Q11.getSoftwareSegments();
```

Here, getSoftwareSegments method gives the SFT segment from the message.

Back to top

## Validation of a Message

If user wants to validate the message, then simply calls validate method upon Message.

```
objQBP_Q11.validate();
```

Back to top

## What is HL7 Source

*HL7 Source* provides the capability to create the HL7 Messages, with respective to their HL7 defined systems.

Back to top

## What is HL7 Recipient

*HL7 Recipient* provides the capability to create the HL7 Messages, with respective to their HL7 defined systems.

Back to top

## Capability of Source and Recipient

HL7 Source and HL7 Recipient can send and receive HL7 messages, but there are some constraints, Source and recipient are not fully capable to send and receive all HL7 messages. Following is the difference,

HL7 Source
Can send – Query and Acknowledgement Can receive – Event, Response and Acknowledgement
HL7 Recipient
Can send – Event, Response and Acknowledgement Can receive – Query and Acknowledgement

Back to top

## HL7 Systems:-

## Patient Admin System

The Patient Administration transaction set provides for the transmission of new or updated demographic and visit information about patients. Any system attached to the network requires information about patients; the Patient Administration transaction set is one of the most commonly used. In general communication, information is entered into a Patient Administration System and passed to the nursing, ancillary and financial systems either in the form of an unsolicited update or a response to a record-oriented query.

Back to top

## Financial Management System

The Finance chapter describes patient accounting transactions. Financial transactions can be sent between applications either in batches or online. The patient accounting message set provides for the entry and manipulation of information on billing accounts, charges, payments, adjustments, insurance, and other related patient billing and accounts receivable information.

Back to top

## Claims and Reimbursement System

Claims and Reimbursement System contains the HL7 messaging specifications to support Claims and Reimbursement (CR) for the electronic exchange of health invoice (claim) data. This system is intended for use by benefit group vendors, Third Party Administrators (TPA) and Payers who wish to develop software that is compliant with an international standard for the electronic exchange of claim data.

Back to top

## Clinical Laboratory System

Clinical laboratory automation involves the integration or interfacing of automated or robotic transport systems, analytical instruments, and pre- or post-analytical process equipment such as automated centrifuges and aliquoters, decappers, recappers, sorters, and specimen storage and retrieval systems. The types of information communicated between these systems include process control and status information for each device or analyzer, each specimen, specimen container, and container carrier, information and detailed data related to patients, orders, and results, and information related to specimen flow algorithms and automated decision making.

Back to top

## Application Management System

This system had previously been entitled Network Management, and has been renamed to more accurately describe the purpose of the messages described. This system does not specify a protocol for managing networks, like TCP/IP SNMP. Rather, its messages provide a means to manage HL7-supporting applications over a network. Because this chapter was originally named "Network Management," the messages and segments have labels beginning with the letter "N." These labels are retained for backward compatibility.

Back to top

## Master Files System

In an open-architecture healthcare environment there often exists a set of common reference files used by one or more application systems. Such files are called master files. These common reference files need to be synchronized across the various applications at a given site. The Master Files Notification message provides a way of maintaining this synchronization by specifying a standard for the transmission of this data between applications.

Back to top

## Materials Management System

This Materials Management system defines abstract messages for the purpose of communicating various events related to the transactions derived from supply chain management within a healthcare facility. There are two basic types of messages defined in this chapter: inventory item master file updates, and supply item sterilization messages. The goal of the Inventory Item Master File Update message specifications is to facilitate the communication of inventory item master catalog and lot information between applications. Sterilization and decontamination messages in this system are exchanged between a sterilizer or washer and an Instrument-tracking System. The main focus of the sterilization and decontamination process is a load or grouping of supply items.

Back to top

## Medical Records System

This system defines the Medical Document Management (MDM) transaction set. It is also intended to support the data exchange needs of applications supporting other medical record functions, including chart location and tracking, deficiency analysis, consents, and release of information. The main purpose of the medical record is to produce an accurate, legal, and legible document that serves as a comprehensive account of healthcare services provided to a patient. It supports transmission of new or updated documents or information about their status. The trigger events and messages may be divided into two broad categories. One which describes the status of a document only and the other that describes the status and contains the document content itself.

Back to top

## Observation Reporting System

This system describes the transaction set required for sending structured patient-oriented clinical data from one computer system to another. Common use of these transaction sets will be to transmit observations and results of diagnostic studies from the producing system (e.g., clinical laboratory system, EKG system) (the filler), to the ordering system (e.g., HIS order entry, physician's office system) (the placer). Observations can be sent from producing systems to clinical information systems (not necessarily the order placer) and from such systems to other systems that were not part of the ordering loop. This system also provides mechanisms for registering clinical trials and methods for linking orders and results to clinical trials and for reporting experiences with drugs and devices. If the observation being reported meets one or more of the following criteria, then the content would qualify as a medical document management message (MDM) rather than an observation message (ORU).

Back to top

## Order Entry System

Order Entry System includes the transmission of orders or information about orders between applications that capture the order, by those that fulfill the order, and other applications as needed. An order is a request for material or services, usually for a specific patient. Most orders are associated with a particular patient. The Standard also allows a department to order from another ancillary department without regard to a patient (e.g., floor stock), as well as orders originating in an ancillary department (i.e., any application may be the placer of an order or the filler of an order).

Back to top

## Patient Care System

Patient Care System supports the communication of problem-oriented records, including clinical problems, goals, and pathway information between computer systems. This system describes healthcare messages that need to be communicated between clinical applications for a given individual. These message transactions can be sent in either batch or online mode.

Back to top

## Patient Referral System

The Patient Referral chapter defines the message set used in patient referral communications between mutually exclusive healthcare entities. These referral transactions frequently occur between entities with different methods and systems of capturing and storing data. Such transactions frequently traverse a path connecting primary care providers, specialists, payers, government agencies, hospitals, labs, and other healthcare entities. The availability, completeness, and currency of information for a given patient will vary greatly across such a spectrum.

Back to top

## Personnel Management System

The Personnel Management transaction set provides for the transmission of new or updated administration information about individual healthcare practitioners and supporting staff members.

Back to top

## Scheduling System

Scheduling System defines messages for the purpose of communicating various events related to the scheduling of appointments for services or for the use of resources.

Back to top

## Query System

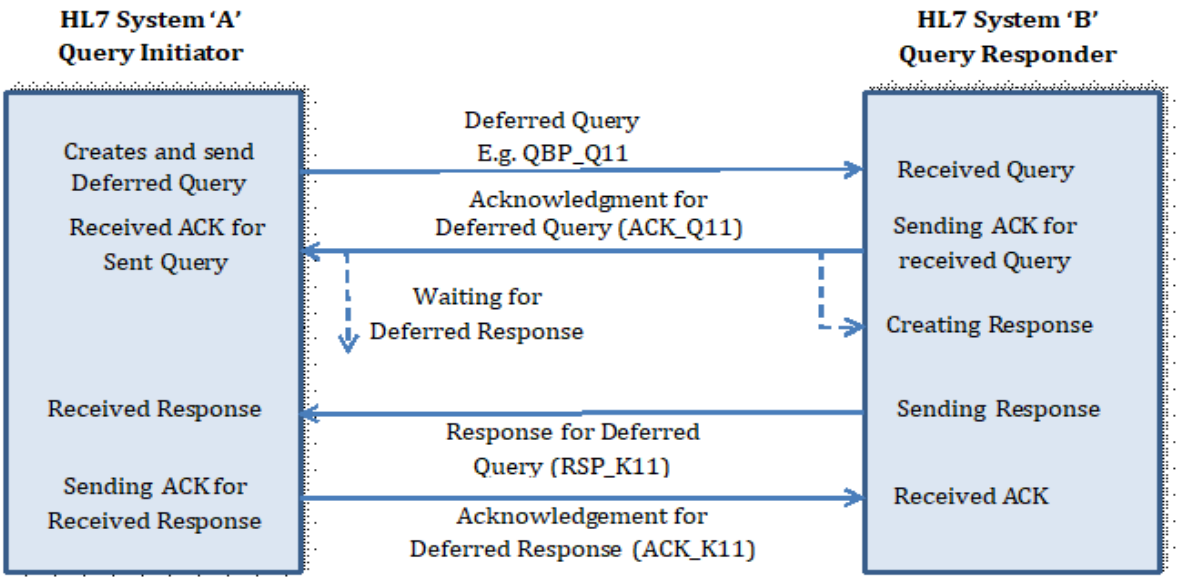
Query System defines the rules that apply to queries and to their responses. It also defines the unsolicited display messages because their message syntax is query-like in nature. The variety of potential queries is almost unlimited.

Back to top

## What is HL7 Communication?

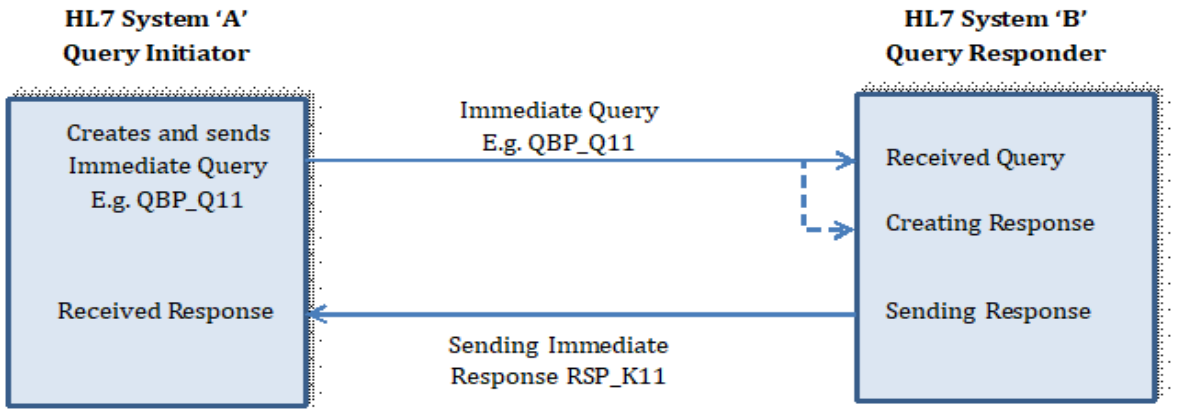
HL7 communication is process by which user can send different HL7 Solicited and Unsolicited messages and receiving of acknowledgment and response for the sent messages.

For successful HL7 Communication between two entities it is mandatory that they should be HL7 compliant. It means that they should be capable of creating message and to send to other entity.



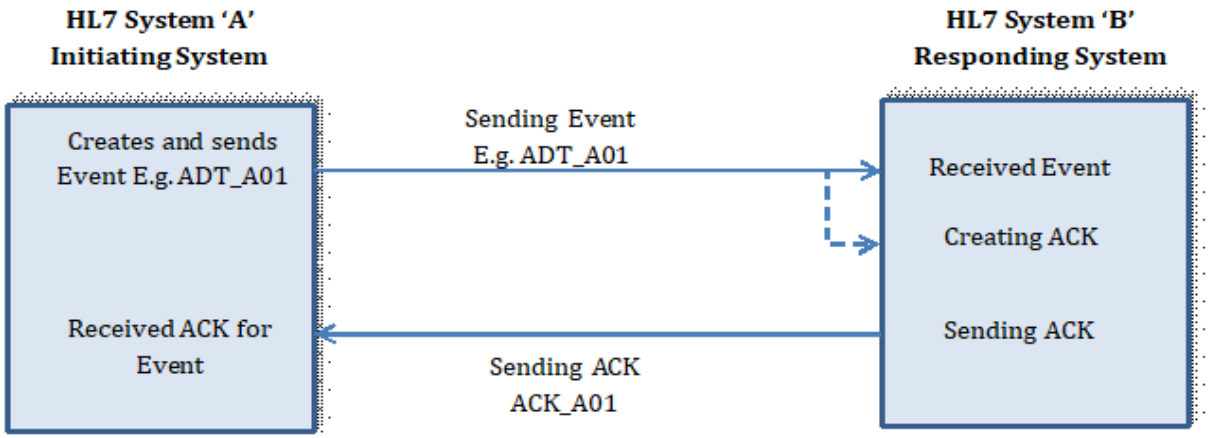

Diagrammatic representation of Query Response model for Deferred Query in HL7 System.

Diagram: (A)




Diagrammatic representation of Query Response model for Immediate Query in HL7 System.

Diagram: (B)




Diagrammatic representation of Query Response model for Event in HL7 System.

Diagram: (C)

Use case Scenarios:

Diagram A:

This describes the scenario where initiating side sends Deferred Query and responding system generates the Acknowledgment for this query and sends it, after time interval responding system sends the response to initiating system which gives notification about successful reception of response by sending ACK to responding System.

Diagram B:

Initiating side sends immediate query and responding side generates the immediate response and sends it to initiating side.

Diagram C:

Initiating side generates event and sends to responding side in response to that responding system sends the ACK to initiating side.

Back to top

## User Session

User Session is entity which takes initiative for sending Solicited and Unsolicited Messages. This entity is present at initiator side on which different entities are set like Message Factory which is having capability of creating and populating of messages. Different interfaces which will redirect the response received to User Test Code directly. All agents like for Batch handling it requires Batch agent, for fragmentation of Messages and Segment it requires Fragment Agent.

For initiating communication first create User Session.

```
UserSession objUserSession = new UserSession();
```

Back to top

## Configuration of User Session

Configuration of User Session for different HL7 Messages,

- For sending of Messages creation and population *MessageFactory* will be created which is loaded with source and recipients.

  To refer more about Message Factory check for *MessageFactory* above. This loaded Message Factory will be set on the User Session.

  Message Factory can be set on User Session like this:

```
objUserSession.setMessageFactory(objMessageFactory);
```

- For Batch send by client receives response in form of batch then in this case it will be redirected to *IBatchReceiver* which is implemented by user test code. By implementing this class will be capable to receive batch response. User can provide his own implementation by implementing interface IBatchReceiver. IBatchReceiver can be set on User Session like this:

```
objUserSession.setIBatchReceiver (IBatchReceiver obj IBatchReceiver);
```

- For Cancellation query send by client received canceled response will be redirected to *ICanceledResponseReceiver* which is implemented by User Test Code. User can provide his own implementation by implementing interface ICanceledResponseReceiver.

```
objUserSession.setICanceledResponseReceiver (ICanceledResponseReceiver obj ICanceledResponseReceiver);
```

- For Interactive query send by client received interactive response will be redirected to *IInteractiveResponseReciever* which is implemented by User Test Code. User can provide his own implementation by implementing interface IInteractiveResponseReciever.

*objUserSession.setIInteractiveResponseReciever (IInteractiveResponseReciever objIInteractiveResponseReciever);*

- As the protocol suggests, after sending the message (event or original mode query) certain waiting period can be defined for the response/ acknowledgement. If this time elapses, sender assumes that the destination entity is not reachable and stops further communication with the destination entity. This can be done as

*objUserSession.setWaitPeriod (waiting time);*

- Sets Socket read timeout of this session, in milliseconds. With this option set to a non-zero timeout, Socket will block for only this amount of time if the timeout expires, a session is closed, and the timeout of 0 will be set as infinite.

objUserSession.setTimeOut (itimeout);

- User can set the read and write length for Message of size being send which will be used at time of fragmentation, According to Write length set it will make fragment of that size. In case of fragment agent is not set then it is mandatory that write length should be greater than that of Message being send. These read and write length can be set as:

objUserSession.setReadWriteLength (int iReadLength, int iWriteLength);

- After this initialization process, session can be started by provided destination entity's IP address and port on which that entity is listening.

*objUserSession.start (strIP, iPort);*

This will connect your client to server with IP address strIP and port as iPort.

Back to top

## Server Session

This server session entity acts as responder for various received messages send by initiating system. This entity also manages the communication with the multiple destination entities. This will acts as responding system for the various Clients by creating separate client session. Various interfaces are registered with Server session for receiving special messages. Server session can be created as,

*ServerSession objServerSession = new ServerSession ();*

Back to top

## Configuration of Server Session

Configuration of Server Session for different Session,

- To receive messages sent by the initiating system and if exception occurs during communication this interface has to be implemented by User. On Registering this interface this will be set on Client Session. User can provide his own implementation by providing implementation to the IProcessHandler. It can be set on Server session like this:

```
objServerSession.setIProcessHandler (IProcessHandler obj IProcessHandler);
```

- As the protocol suggests, after sending the message (event or Original mode query) certain waiting period can be defined for the acknowledgement. If this time elapses, sender assumes that the destination entity is not reachable and stops further communication with the destination entity.

```
objServerSession.setWaitPeriod (waiting time);
```

- User can set the read and write length for Message of size being send which will be used at time of fragmentation, According to Write length set it will make fragment of that size. In case of fragment agent is not set then it is mandatory that write length should be greater than that of Message being send. These read and write length can be set as:

```
objServerSession.setReadWriteLength (int iReadLength, int iWriteLength);
```

- Sets Socket read timeout of this session, in milliseconds. With this option set to a non-zero timeout, Socket will block for only this amount of time if the timeout expires, a session is closed, and the timeout of 0 will be set as infinite.

```
objServerSession.setTimeOut (int itimeout);
```

Back to top

## Client Session

This is entity which represents client at server side. It will be created each time when server connected to client. For each client connection separate data will be kept by using this entity by setting required agents on it.

Object of client Session can be received on method onConnect method by using which user can set different agents on it.

```
public void OnConnect(ClientSession objClientSession)
{
//Set agent on Client Session.
objClientSession.setAgent (Agent);
}
```

Note: - Here setAgent can be setPublishAgent , setSequenceAgent, setServerSideQCNAgent, setServerSideQICAgent.

Back to top

## Configuration of Client Session

Configuration of Client Session for different Session,

- For Sending and receiving capability of messages it requires source and recipient is added to Message Factory. This has to load message factory and set the agents as per requirement. All auxiliary protocols and transaction handling which keeps track for the incoming and receiving messages will be set by User when it receives Client Session. This can be set as:

```
objClientSession.setMessageFactory(objMessageFactory);
```

- For Cancellation query send by client will be redirected to *ICanceledQueryReceiver* which is implemented by User Test Code. User can provide his own implementation by implementing interface ICanceledQueryReceiver.

```
objClientSession. setICanceledQueryReceiver (ICanceledQueryReceiver
objICanceledQueryReceiver);
```

- For Interactive query send by client it will be redirected to *IInteractiveQueryReciever* which is implemented by User Test Code. User can provide his own implementation by implementing interface IInteractiveQueryReciever.

```
objClientSession. setIInteractiveQueryReciever (IInteractiveQueryReciever obj
IInteractiveQueryReciever);
```

- For subscription message sent by client it will be redirected to *ISubscriptionListener* which is implemented by User Test Code. User can provide his own implementation by implementing interface ISubscriptionListener.

```
objClientSession. setISubscriptionListener (ISubscriptionListener
objISubscriptionListener);
```

Back to top

## Establish connection with responding entity

For establishing connection with responding entity first it has to configure initiating side i.e. User Session.
Refer to *configure User Session*. For sending message from initiating side it has to connect with responding system so this can be done by providing IP address and port at which Responding system is present. This can be done as follows:

```
Objusersession.connect(String strip, int iPort);
```

Back to top

## Sending HL7 message(event/query) to responder entity

By the initiating side different HL7 messages can be sent. It may be query, Event, Batch and ACK. For this user has to create message by source and recipient present.

Refer *HL7 Messages*.

This can be done as:

```
objUserSession.sendMessage(Message objMessage);
Collection of different message in the form of batch file can be send as follows:
objUserSession.sendBatchFile(IHl7BatchFile objIHl7BatchFile);
```

Back to top

## Establish connection with requestor entity

For establishing connection with requestor entity first it has to Configure Responding side i.e. Server Session.

Refer link to *configure Server Session*. For sending message from responding side it has to connect with initiating system. So, responding system starts to listen on specific port and particular IP address.

This can be done as follows:

```
objServerSession.start(String strIP, int iPort);
```

Where strIP is IP address and iPort is port at which it is listening.

After successful connection with initiating side it creates Client session for that requester entity. Responding side can set desired configuration on client session like setting different agents and handling of query by configuring it.

Please refer link *configure Client Session*

Back to top

## Process received HL7 message(query/event)

For processing the received event, batch or Query requester side has to implement the interface *IProcessHandler*.

This interface has to register on Server Session and implementing this interface requester entity will be capable to receive Messages on method,

```
OnMessageReceive(ClientSession objClientSession);
```

Same way Batch file can be receive on method

```
onBatchFileReceive(ClientSession objClientSession);
```

Requester side can process the incoming messages and batch file and it can produce the desired response or ACK

Back to top

## Generate response/acknowledgement

Requester entity processes the incoming message and generates response/acknowledgement as per the type of HL7 Message. For Event it generates the ACK, for deferred query first it sends the ACK and then its response. For Immediate Query it directly generates the Response and sends it. For batch file as per the messages in it will generate ACK and response it sends it in batch or as single message. ACK can be generated as follows:

For generating ACK it requires two things
1. Message Map
2. Trigger event of message received

Message Map can get by Message Map Reader by creating it as like this:

```
IMessageMapReader objMessageMapReader = MessageMapReader.createInstance();
IMessageMap objMessageMap =
objMessageMapReader.getMessageMap(EnumMessageCode.ACK, null);
```

Trigger Event can get from message as like this:

```
String strTriggerEvent = objReceivedMessage.getTriggerEvent();
```

Pass this value for creation of ACK.

```
ACK ackMessage = new ACK(strTriggerEvent, objMessageMap);
```

Different Segments in it like MSH, SFT, MSA, ERR. Can be populated by meaningful data and send it to Requester entity.

The overall code looks like:

```
public void OnMessageReceive(ClientSession cSession)
{
Message message = cSession.getMessage();

IMessageMapReader objMessageMapReader = MessageMapReader.createInstance();
IMessageMap objMessageMap =
objMessageMapReader.getMessageMap(EnumMessageCode.ACK, null);

String strTriggerEvent = objReceivedMessage.getTriggerEvent();

//Passing this value for creation of ACK.
ACK ackMessage = new ACK(strTriggerEvent, objMessageMap);
ackMessage. setHeader(MSH objMSH);
ackMessage.addSoftwareSegment(objSFT);
ackMessage.setMessageAcknowledgmentSegment(MSA objMSA);
ackMessage.addErrorSegment(objERR);

cSession.sendMessage(ack);
cSession.close();
}
```

Response can be generated as follows:

Response can be of two types, Immediate and Deferred it can be decided by Segments QRD and RCP in which QueryPriority attribute. “D” value denotes for Deferred and “I” denotes as Immediate.

Response can be generated for Query e.g. for Query QBP_Q11 its Response can be send by RSP_K11.

So create message RSP_K11 by QryRecipient by providing Message Code and Trigger event as follows.

```
RSP_K11obj RSP_K11 = (RSP_K11) new
QryRecepient().createMessage(EnumMessageCode.RSP, EnumTriggerEvent.K11);
```

Add different Segments in it like MSH, SFT, MSA, ERR, QAK, QRD, QRF, DSP with meaningful data and set all this messages one by one on Message RSP_K11.

E.g.

*objRSP_K11.setHeader(MSH objMSH);*
*objRSP_K11.setQueryDefinitionSegment(QRD objQRD);*
*And send this response to requester entity.*

Back to top

## Minimal Lower Layer Protocol (MLLP)

## What is MLLP?

The Minimal Lower Layer Protocol (MLLP) is the most common mechanism for exchanging the HL7 data. MLLP uses the TCP/IP protocol to transfer the data in continuous stream of bytes. MLLP delimiters are used to recognize the start and the end of message.

MLLP is how you wrap an HL7 message with a start and end to insure you knows where a message starts, where a message stops, and where the next message starts.

The typical structure of an HL7 message being sent via MLLP is described in the table below. It contains four parts:

The header is a vertical tab character <VT> its hex value is 0×0b. The trailer is a field separator character <FS> (hex 0×1c) immediately followed by a carriage return <CR> (hex 0×0d)

| <VT> (hex 0×0b) | HL7 Message goes here | <FS> (hex 0×1c) | <CR> (hex 0×0d) |
|---|---|---|---|

These headers and trailers are usually non-printable characters that would not typically be in the content of HL7 messages.

The structure of an MLLP message is given below
<SB> + <Message> + <EB> + <CR>

<SB> = Start Block. Messages are prefixed with start byte
<Message> = HL7 Message
<EB> = End Block. Messages are terminated with end byte
<CR> = Carriage Return

Default hexadecimal values of MLLP delimiters
<SB> = 0x0B (VT)

<EB> = 0x1C (FS)
<CR> = 0×0D (CR)

Configuration -

If user wants to send the HL7 message with MLLP component, then user have to enable MLLP on UserSession.

```
//Enable MLLP on User Session.
objUserSession.enableMLLP();
```

And for receiving the HL7 message with MLLP component, then user have to set MLLP on ClientSession.

```
//Set MLLP on Client Session.
IMLLP objIMLLP = new MLLP();
objClientSession.setMLLPProcessor(objIMLLP);
```

How it works?

When user tries to send HL7 Message with MLLP component, MLLP Agent adds its entire component on the HL7 Message with the help of its agent.

The structure of an MLLP message is given below
<SB> + <Message> + <EB> + <CR>

<SB> = Start Block. Messages are prefixed with start byte
<Message> = HL7 Message
<EB> = End Block. Messages are terminated with end byte
<CR> = Carriage Return

When this message reached on the receiving side, MLLP agent removes its entire component.

If user not set the MLLP at the receiving side, then it will throw exception.

Back to top

## Message/Segments Continuation Configuration

Sometimes, implementation limitations require that large messages or segments be broken into manageable chunks for ease of transmission of Data so here in SDK this term is called as "fragmentation". This describes how a logical message is broken into one or more separate HL7 messages.

- First, a single segment may be too large. HL7 uses the "ADD" segment to handle breaking a single segment into several smaller segments.

- Second, a single HL7 message may be too large. HL7 uses the DSC segment and the continuation protocol to handle message fragmentation.

Note: HL7 does not define what "too large" means. Acceptable values are subject to site negotiations.

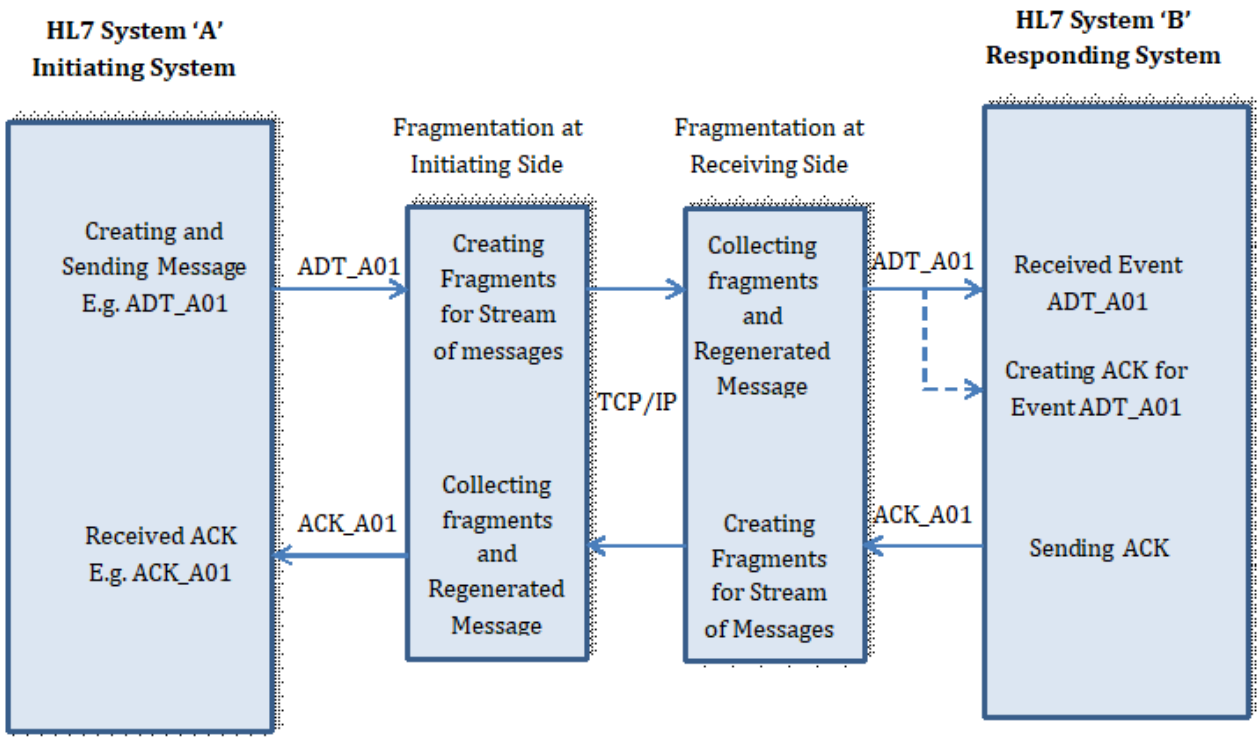



Diagram A: Diagrammatic representation of Fragmentation in HL7 Communication.

Use case Scenarios:

Diagram A:
-Initiating side generates event and sends to responding side but while sending this message it is broken down in to small chunks if user wants.
-These chunks are received at Server Side and collected all fragments and regenerated message with help of this chunks.
-Same step is repeated while sending it from server side.

At initiating side:
The default implementation is given by SDK library for IFragmentProcessor in form of FragmentAgent however user can implement his own behavior by implementing IFragmentProcessor.

Instantiating IFragmentProcessor which handles the fragmentation and de-fragmentation for larger messages.

```
IFragmentProcessor objIFragmentProcessor = new FragmentAgent ();
```

This fragment agent who will does the function of fragmentation for the sending message on network which forms the packet size as specified by user. When it sets the value of Read and Write length on User Session.

This fragment processor can be set on User Session like this:

```
objUserSession.setFragmentProcessor (IFragmentProcessor objIFragmentProcessor);
```

At receiving side:
The default implementation is given by SDK library for IFragmentProcessor in form of FragmentAgent however user can implement his own behavior by implementing IFragmentProcessor.

Instantiating IFragmentProcessor which handles the fragmentation and de fragmentation for larger messages.

```
IFragmentProcessor objIFragmentProcessor = new FragmentAgent ();
```

This fragment agent who will does the function of fragmentation for the sending message on network which forms the packet size as specified by user. When it sets the value of Read and Write length on Server Session.

This fragment processor can be set on Server Session like this:

```
objUserSession.setFragmentProcessor (IFragmentProcessor objIFragmentProcessor);
```

Back to top

## Batch Configuration

Batch protocol specifies the structure of batch. HL7 Batch transfer is shown as follows.

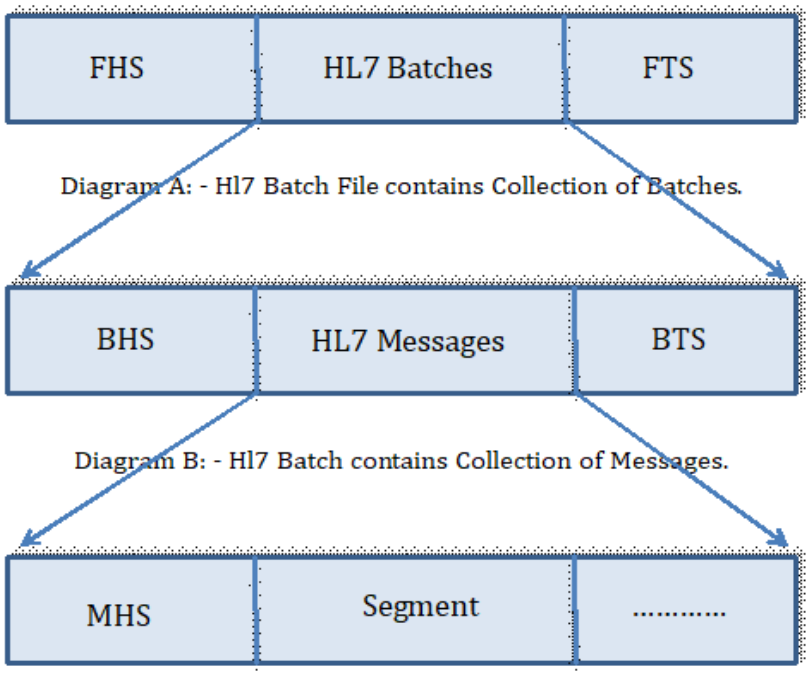



Diagram C: - Hl7 Message contains Different Segments.

Use case Scenarios:

Diagram A:
As shown in diagram HL7Batch File is contains HL7 batch inside with two segments FHS and FTS which is file header and File trailer segment.

Diagram B:
As shown in diagram HL7Batch contains HL7 Messages with two segments BHS and BTS which is Batch header and Batch trailer segment.

Diagram C:
As shown in diagram HL7 Message contains Different segments like MSH, DSC which is Message header and Continuation pointer segment.

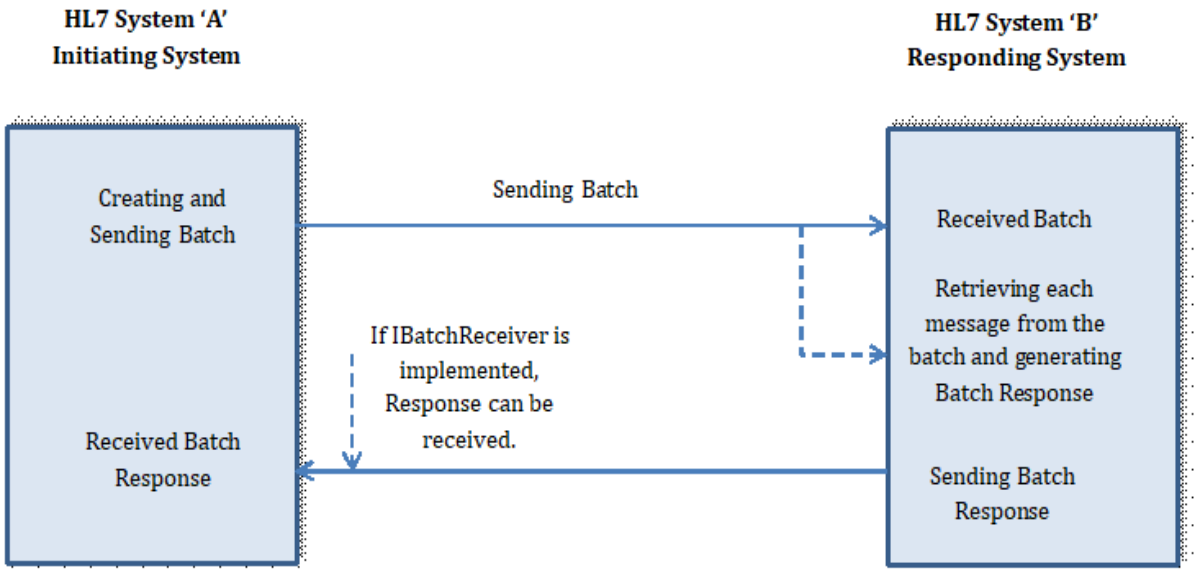



Diagram A: Diagrammatic representation of Batch protocol in HL7 Communication.

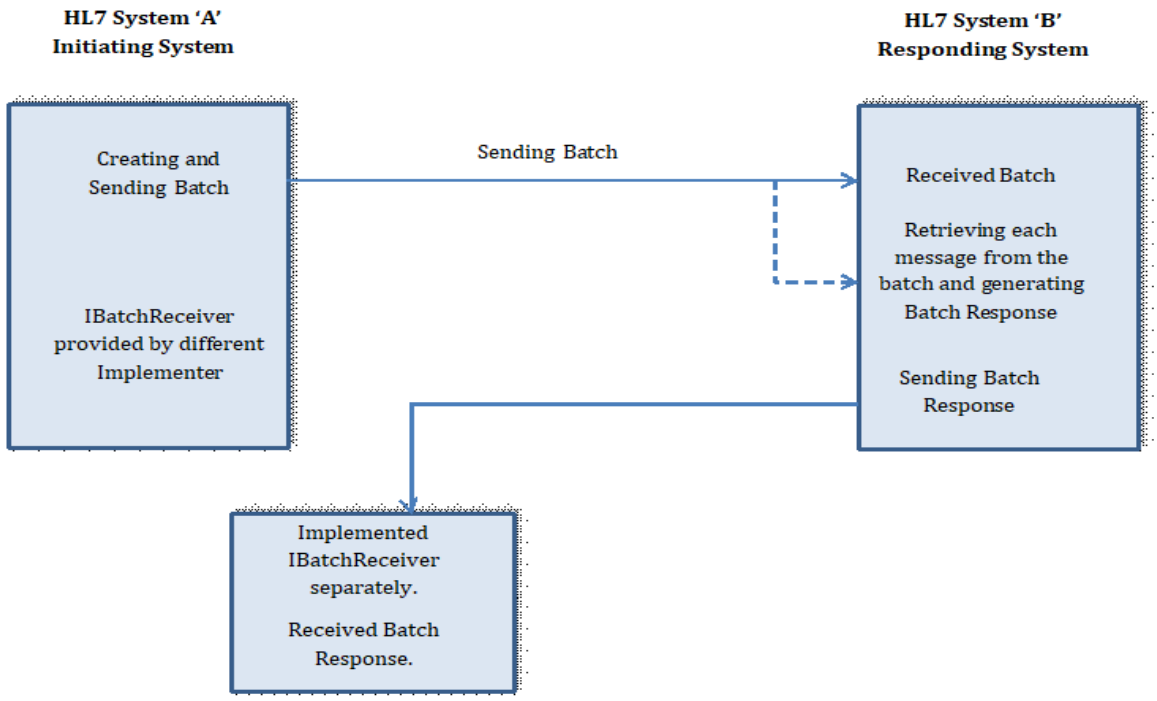



Diagram A: Diagrammatic representation of Batch protocol giving separate Implementation for IBatchReceiver in HL7 Communication.

Use case Scenarios:

Diagram A:
As shown in diagram initiating system creates the Batch file and sends it to Responding system which will be analyzed by responding system. Each message will be verified and generates Response for it which will be added in batch and this is given as batch response to initiating system. If initiating system has implemented IBatchReceiver so received response on method OnBatchFileReceive(ClientSession objClientSession).

Diagram B:
User can provide different implementation for IBatchReceiver now all the batch response will be redirected to implementer of IBatchReceiver on method onBatchReceive(ClientSession objClientSession).

Creation of Batch: To create batch follow the below steps:

1. Create and populate BHS segment.
2. Create and populate BTS segment.
3. Create a collection of Messages to be placed in HL7 Message Batch.

Please refer link HL7 Messages for creation and population of it.

```
BHS objBatchHeader = new BHS();
objBatchHeader.setBatchFieldSeparator('|');
objBatchHeader.setBatchEncodingChars('^','&','~','\\');
objBatchHeader.setBatchControlID("Batch123456");

//code to populate BTS.
BTS objBatchTrailer = new BTS();
objBatchTrailer.addBatchTotals("2");

//create collection of messages.
ISingleCollection <Message> objMessageCollection = new SingleCollection
<Message> ();
objMessageCollection.add(Message1);
objMessageCollection.add(Message2);

IHL7Batch objBatch = new
HL7Batch(objBatchHeader,objBatchTrailer,objMessageCollection);
```

In this way the HL7 message batch contains Message1 and Message 2.

Creation of Batch File: To create batch follow the below steps:

1. Create and populate FHS segment
2. Create and populate FTS segment.
3. Create HL7 Message Batches to be serialized in Batch File. As just now shown.
4. Create HL7 Batch File using HL7BatchFile class.

```
//Code to populate FHS
FHS objBatchFileHeader = new FHS();
objBatchFileHeader.setFieldSeparator('|');
objBatchFileHeader.setEncodingChars('^','&','~','\\');
objBatchFileHeader.setSubComponentSeparator('!');
objBatchFileHeader.setFileControlID("BatchFile1234");

//code to populate FTS.
FTS objBatchFileTrailer = new FTS();
objBatchFileTrailer.setFileBatchCount("1");

//Created Batch added in Batch File.
objIHL7BatchCollection.add(batch1);

IHL7BatchFile objBatchFile = new HL7BatchFile (objBatchFileHeader,
objBatchFileTrailer, objIHL7BatchCollection);
```

Configuration:

The default implementation is given by SDK where user can give his own implementation by implementing interfaces IBatchProcessor for initiating system

at initiating side:

User can use the default implementation of IBatchProcessor which is BatchAgent. First instantiate the BatchAgent and set it on User Session.

```
IBatchProcessor objIBatchProcessor = new BatchAgent ();
```

Configure the User session refer Configure User Session.

Set this agent on User Session like this:

```
objServerSession.setBatchProcessor (objIBatchProcessor);
```

Now initiating side can send batch file with help of User session like this. However user just wants to send batch then in this case it has to make it in form of batch file and then only it can send this batch.

```
objUserSession.sendBatchFile (IHL7BatchFile objIHL7BatchFile);
```

After sending batch response received in batch can be get by implementing IBatchReceiver. This implementer has to set on User Session. In case of batch received then in this case it will be redirected to onBatchReceive(IHL7BatchFile objHL7BatchFile); on this method. IBatchReceiver can be set on User session like this.

```
objUserSession.setIBatchReceiver(IBatchReceiver objIBatchReceiver);
```

If any case IBatchReceiver is not set it on Session then in this case last response batch will be added in collection. User can get the latest batch response by this collection.

```
ISingleCollection <IHL7BatchFile> objACKBatchCollectionForFile =
objUserSession.getResponseBatches();
```

Note that only one and last response batch will be stored in collection.

At receiving side:

Receiving side has implemented the interface IProcessHandler. Which has method OnBatchFileReceive(ClientSession objClientSession) on which user can gets the Batch File. Receiving side retrieves the batch file and processes the each messages in it and generates the ACK and Response according to type of Message. Please refer link Generate response/acknowledgement for processing of Message to generate ACK/Response. The overall code looks like this:

```
public void OnBatchFileReceive (ClientSession objClientSession)
{
IHL7BatchFile objHL7BatchFile = objClientSession.getBatchFile();
//objHL7Batch now contains batch file received from client system
}
```

E.g.

1.Receiving side gets Event, Deferred Query then it will generate Batch file in which ACK for event and Deferred Query will be added and send. Deferred Response will send in form of batch if message wants or simple response will be created and send.

2. In case of Immediate Query and Event is received then in this case Batch with ACK for Event and Response for Query will be created and added to batch which will be send.

Back to top

## Interactive Continuation Configuration

The Interactive Continuation Protocol defines the methodology for the intentional transmission of a large query-response payload over multiple HL7 messages. Without this protocol, the response would be returned in a single large logical message.

The protocol is called interactive because there is an ongoing dialog between the Client and the Server. The dialog commences when the Client issues a query for a potentially large amount of data, but specifies in the RCP-2-Quantity limited request, that only a limited amount of data is to be returned in each continued response. The Server then returns one response message containing

data up to the requested quantity. The Client may continue to ask for further subsets of the data until the entire set is exhausted or may choose to cancel the query.

This use of the term continuation responses in queries should not be confused with its use in continuing an unsolicited fragmented message. In the case of continuing a response to query the control is on the side of the querying application and there is an explicit cancellation event. In the case of continuation of an unsolicited message, the control is on the part of the sending application and there is no concept of canceling the message transmission.

The default implementation is given by SDK, where user can give his own implementation by implementing interfaces IClientInteractiveContinuation for initiating system and IServerInteractiveContinuation for responding system.

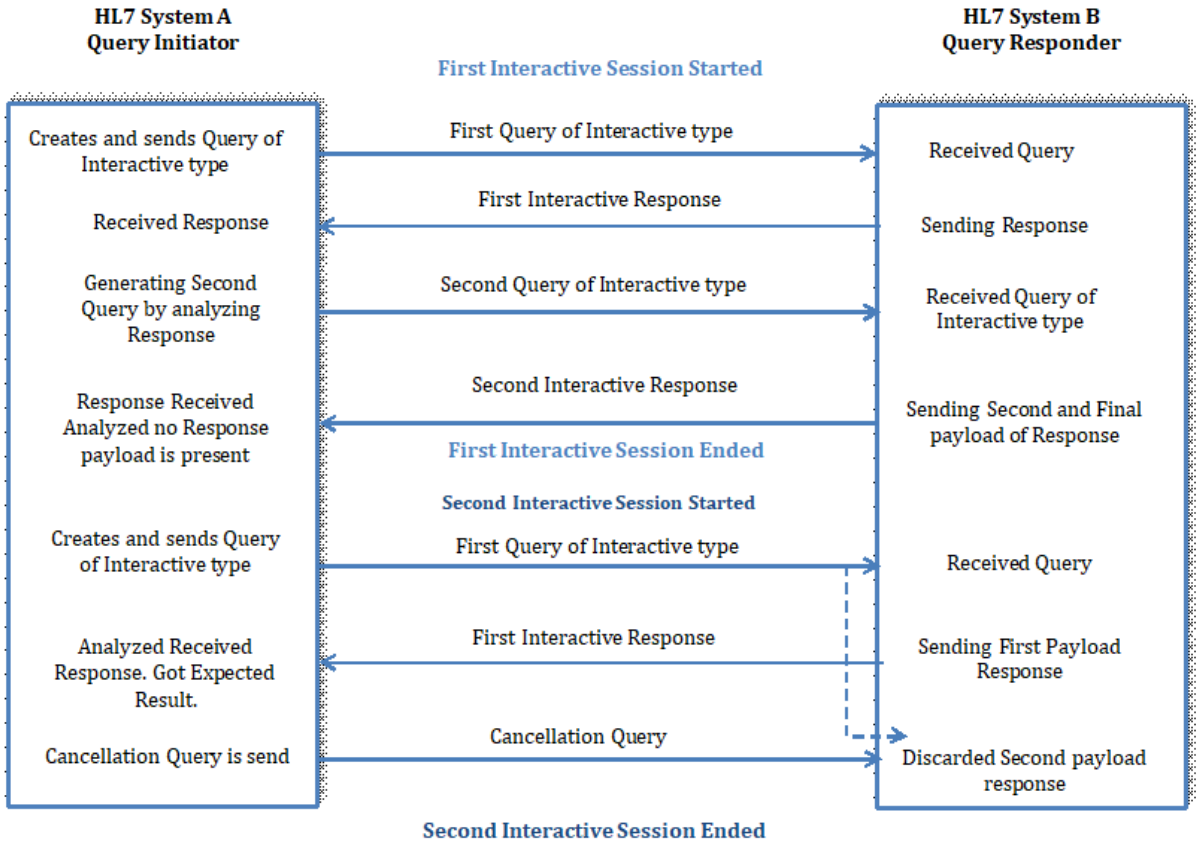



Diagram A: Diagrammatic representation of Two HL7 Systems Sending Query of Interactive Type.

Use case Scenarios:

Diagram A:

-As shown in diagram initiating system sends the query of interactive type and sends it to responding system. With respect to that it sends the ACK and generated the response for this

query. First payload response is sent to initiating system and in DSC field it is set that more response payloads are still remained. Initiating system analyzes the response and generates query for more payload and responding system sends the Response with setting that no more response payload is now presenting this way first interactive session ended.

-In one of scenario of Second interactive session for first interactive query when send by initiating system server sends first payload and set DSC field which says that more response payloads are present. After analyzing the response initiating system got the required data so it sends cancellation query for server which closes the interactive session.

At initiating side:
User can use the default implementation of IClientInteractiveContinuation which is ClientSideQICAgent.

First instantiate the ClientSideQICAgent and set it on User Session.

```
IClientInteractiveContinuation  objClientSideQICAgent = new ClientSideQICAgent();
```

Configure the User session refer Configure User Session.

Set this agent on User Session like this:

```
objUserSession.setClientSideQICAgent(objClientSideQICAgent);
```

1. After receiving response from the Responding system if received subset of data of the message is terminated with a DSC segment with the DSC-1-Continuation pointer set to the appropriate pointer value and the DSC-2 -Continuation type set to "I"
2. If the Client wishes to receive the next installment, the query is sent again with a DSC segment following the RCP. The DSC-1-Continuation pointer echoes the value sent by the Server.
3. The Server continues to send installments in response to the Client's request until there is no more data. The end of data is signified by the absence of the DSC segment OR an empty value in DSC-1-Continuation pointer.
4. If the Client wishes to cancel the query before the end of the data is reached, a Cancel query is sent.

However user can use the different utility methods provided on Interactive agent to generate interactive query and cancellation query by Query id of previously send query.

```
ISegmentSet objSegmentSet =
objClientSideQICAgent.generateInteractiveQuery(String strQueryID);
```

By using this segment set user can generate its own interactive Query.

Similarly for cancellation of query user can get the Segment Set from agent providing Query id for Query which has to cancel.

```
ISegmentSet objSegmentSet = objClientSideQICAgent. generateCancellationQuery
(String strQueryID);
```

User can handle the Interactive Response on client side by implementing interface IInteractiveResponseReciever and Set this interface on user session. Whenever for any interactive response will be received by Client then in this case it will be redirected to method given on this interface onInteractiveResponseReceive(Message objMessage).

The overall code looks like this:

```
objUserSession.setIInteractiveResponseReciever(IInteractiveResponseReciever
objIInteractiveResponseReciever);

public void onInteractiveResponseReceive(Message  Response)
{
//Received interactive Response.
}
```

At receiving side:

User can use the default implementation of IServerInteractiveContinuation which is ServerSideQICAgent.

First instantiate the ServerSideQICAgent and set it on Client Session.

```
IServerInteractiveContinuation objServerSideQICAgent = new ServerSideQICAgent ();
```

Configure the User session refer Configure Client Session.

Set this agent on Client Session like this:

```
public void OnConnect(ClientSession objClientSession)
{
objClientSession.setServerSideQICAgent(ServerSideQICAgent
objServerSideQICAgent);
}
```

Receiving side interactive queries can be handled separately by implementing interface by receiver called IInteractiveQueryReciever.

Those queries which are interactive will be redirected to method

```
onInteractiveQueryReceive(ClientSession objClientSession);
```

User can process the message and generates the response and ACK for it.

Please refer link Generate response/acknowledgement. If for this query if multiple response payload are present then it add the DSC segment set the DSC-1-Continuation pointer to the appropriate pointer value and the DSC-2 -Continuation type to “I”.

If initiating side is interested for more response payload messages then in this case it will send another interactive query to responder.

Things to remember:
1. It is mandatory that when library is in STRICT mode then while sending response payload if QAK Segment is present then it must echo back the value of Message Control ID field received from the MSH segment of received Message.
2. Query id in case of interactive query is same throughout the session.

Back to top

## Query Cancellation Configuration

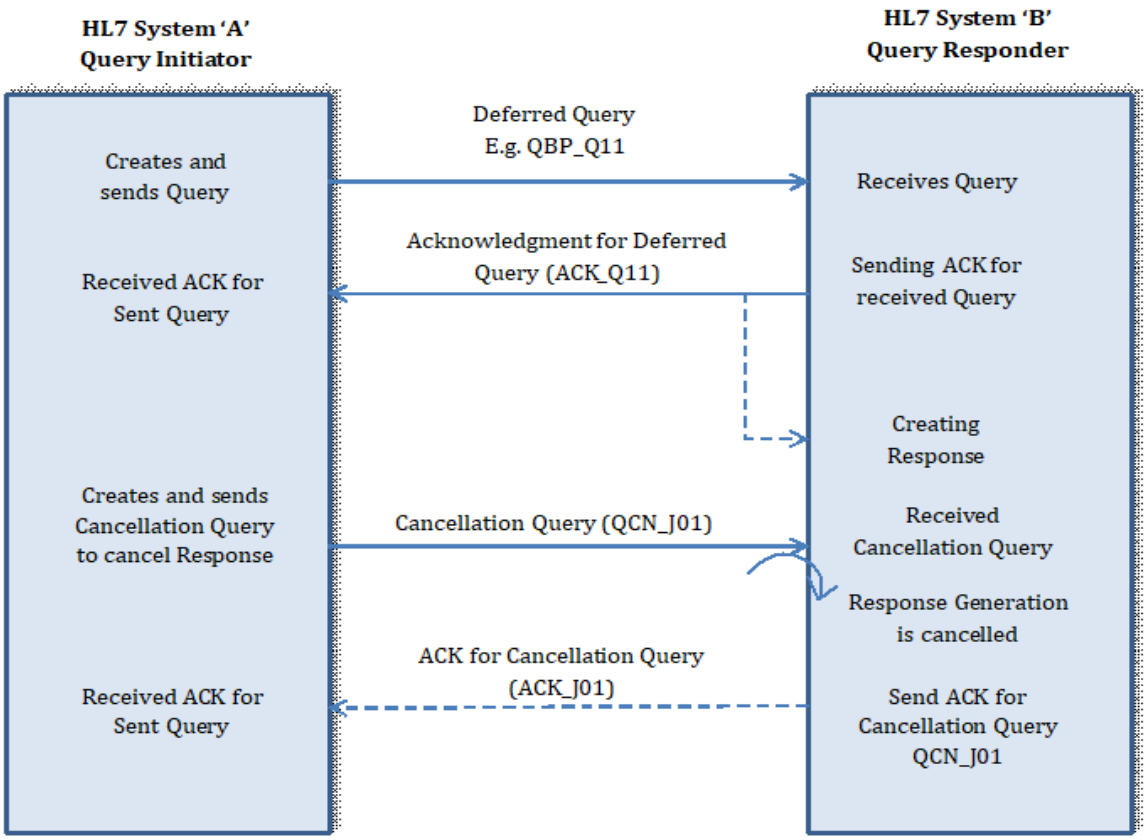



Diagram A: Diagrammatic representation of Query Cancellation process between two HL7 Systems.

Use case Scenario:

Diagram A:

-As shown in diagram initiating system sends the query to responding system for which it sends ACK for that. During mean time initiating system sends out the cancellation query which cancels the response.
-Responding system can send ACK for the query QCN_J01.

Configuration:

Query cancellation is the canceling of send request in the order by sending specific query to responding entity for which response may already on its way. However client will not be interested in received response for a query that is cancelled.

The default implementation is given by SDK, where user can give his own implementation by implementing interfaces IClientQueryCancellation for initiating system and IServerQueryCancellation for responding system.

At initiating side:

User can use the default implementation of IClientQueryCancellation which is ClientSideQCNAgent.

First instantiate the ClientSideQCNAgent and set it on User Session.

```
IClientQueryCancellation objClientSideQCNAgent = new ClientSideQCNAgent();
```

Configure the User session refer Configure User Session.

At set this agent on User Session like this:

```
objUserSession.setClientSideQCNAgent (objClientSideQCNAgent);
```

At responding side:

User can use the default implementation of IServerQueryCancellation which is ServerSideQCNAgent.

First instantiate the ServerSideQCNAgent and set it on User Session.

```
IServerQueryCancellation objServerSideQCNAgent = new ServerSideQCNAgent ();
```

Configure the Client session refer Configure Client Session.

Set this agent on client Session like this:

```
public void OnConnect(ClientSession objClientSession)
{
objClientSession.setServerSideQCNAgent (objServerSideQCNAgent);
}
```

Back to top

## Query Cancellation Process

At initiating side:

Cancellation query for canceling response for sent query can be done by special query called QCN. This query can be generated by Query source.

e.g. QCN_J01 query is to be sent for canceling response of Deferred Query QBP_Q11.

Instantiating QrySource like:

```
QrySource objQrySource = new QrySource ();
```

Create message with Message Code QCN and Trigger event J01.

```
QCN_J01 objQCN_J01 = (QCN_J01) objQrySource.createMessage
(EnumMessageCode.QCN, EnumTriggerEvent.J01);
```

Different Segments can be added in it like MSH, SFT and QID by populating with meaningful data. Important segment in this message is QID which tells about which query is to cancel.

Where in QID segment in attribute Query Tag set the value of Query id for query whose response is to be canceled.

This can be shown as:

```
QID objQID = new QID ();
objQID.setQueryTag (String strQueryIdToCancel);
objQCN_J01.setQueryIdentificationSegment (objQID);
```

While generation of Segment Set for making cancellation query you can use the utility method provided by ClientSideQCNAgent which will return the segment set for it. For that you have to just pass the Query id for which cancellation query is to send.

This can be shown as:

```
ISegmentSet objISegmentSet =
objClientSideQCNAgent.generateCancellationQuery (String strQueryIDtoCancel);
```

By this Segment Set you can generate the Cancellation query. For cancellation response if received at Initiating side then this data will be discarded if intersected in it then User implement interface ICanceledResponseReceiver and set this on userSession like this:

```
objUserSession.setICanceledResponseReceiver(ICanceledResponseReceiver
objICanceledResponseReceiver);
```

After receiving cancelled response this will be redirected to method,

```
onResponseCancellationReceive (Message objMessage);
```

At receiving side:

For handling cancellation query separately responding side can implement the interface ICanceledQueryReceiver.

Set this interface on Client session like this:

```
public void OnConnect(ClientSession objClientSession)
{
//This will set the ICanceledQueryReceiver on Server Session.
objClientSession.setICanceledQueryReceiver(ICanceledQueryReceiver obj
ICanceledQueryReceiver);
}
```

Now all valid cancellation queries will be redirected to method of class implemented by this interface onQueryCancellationReceive(ClientSession objClientSession).

If user has not registered it then in this case it will be given on method OnMessageReceive(ClientSession objClientSession).

For cancellation query server can send acknowledgment for query.

Please refer link Generate response/acknowledgement.

Back to top

## Sequence Number Configuration

For certain types of data transactions between systems the issue of keeping databases synchronized is critical. An example is an ancillary system such as lab, which needs to know the locations of all inpatients to route stat results correctly. If the lab receives an ADT transaction out of sequence, the census/location information may be incorrect. Although it is true that a simple one-to-one acknowledgment scheme can prevent out-of-sequence transactions between any two systems, only the use of sequence numbers can prevent duplicate transactions.

1. The sequence number is a positive (non-zero) integer; and it is incremented by one for each successive transaction.

2. The system receiving the data stream is expected to store the sequence number of the most recently accepted transaction in a secure fashion before acknowledging that transaction. This stored sequence number allows comparison with the next transaction's sequence number.
3. The initiating system keeps a queue of outgoing transactions indexed by the sequence number. The length of this queue must be negotiated as part of the design process for a given link. By default the minimum length for this queue is one.

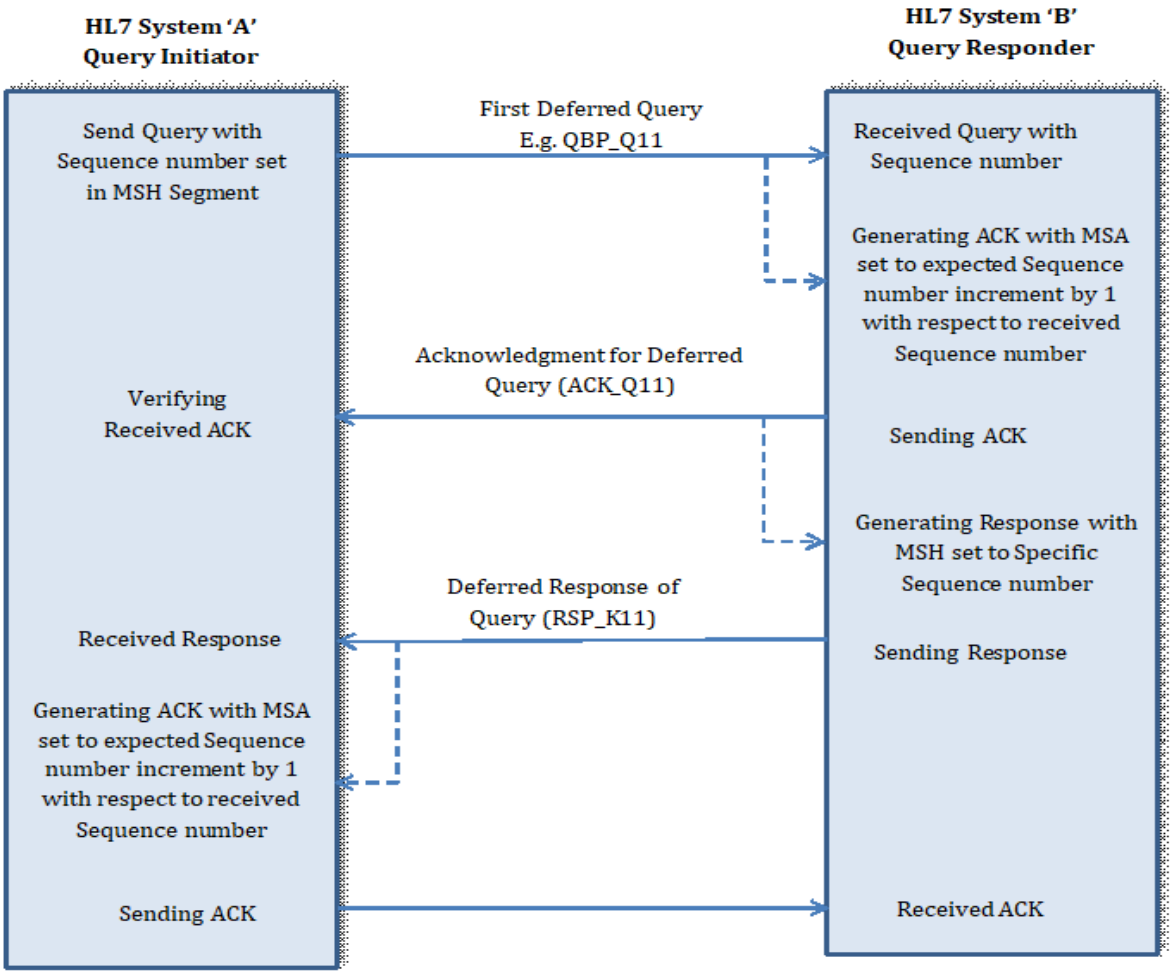



Diagram A: Diagrammatic representation of two HL7 Systems performing Sequence numbering of HL7 messages.

Use case Scenarios:

Diagram A:

-As shown in diagram initiating system sends message with particular Sequence number to responding system. Now responding system echo backs by incrementing it by 1 and set it to MSA-14 field.
-While sending response from responding system it starts the sequence number from start and

sends it to initiating system which will be increment by 1 and set it to MSA-14 field.

Configuration:

The default implementation of sequence number is given by SDK. Where user can give his/her own implementation by implementing interfaces ISequenceAgent.

At initiating side:

User can use the default implementation of ISequenceAgent which is InitiatingSequenceAgent.

Instantiate the InitiatingSequenceAgent like this:

```
ISequenceAgent objInitiatingSequenceAgent = InitiatingSequenceAgent.createInstance();
```

So set the queue length like this:

```
objInitiatingSequenceAgent.setQueueLength(int iLength);
```

Set the Initial Queue Value like this:

```
objInitiatingSequenceAgent.setInitialQueueValue(int iValue);
```

If user wants the Sequence numbering should be applied for all incoming messages and sending message then start the functionality of this agent.

```
objInitiatingSequenceAgent.setStartSequencing(true);
```

Now set this InitiatingSequenceAgent on user session.

```
objUserSession.setSequenceAgent(objInitiatingSequenceAgnet);
```

When initiating system (client side) sent any message with non-zero or positive integer, it keeps the Sequence number (with the MSH in MSH-13-sequence number) in queue, length of this queue must be negotiated as part of the design process for a given link. On the next time sequence number received by receiving system will be incremented by one (by the initiating system) for each successive transaction by the initiating system.

At receiving side:

User can use the default implementation of ISequenceAgent which is InitiatingSequenceAgent.

Instantiate the InitiatingSequenceAgent like this:

```
ISequenceAgent objReceivingSequenceAgent = InitiatingSequenceAgent.createInstance();
```

Set the queue length like this:

```
objReceivingSequenceAgent.setQueueLength(int iLength);
```

Set the Initial Queue Value like this:

```
objInitiatingSequenceAgent.setInitialQueueValue(int iValue);
```

If user wants the Sequence numbering should be applied for all incoming messages and sending message then start the functionality of this agent.

```
objReceivingSequenceAgent.setStartSequencing(true);
```

Now set this InitiatingSequenceAgent on Client session.

```
public void OnConnect(ClientSession objClientSession)
{
//This will set the SequenceAgent on Client Session.
objClientSession.setSequenceAgent(objReceivingSequenceAgent);
}
```

Receiving system (server side) stores the sequence number of the most recently accepted transaction in a secure fashion before acknowledging that transaction. This stored sequence number allows comparison with the next transaction's sequence number.

When initiating system (client side) sent any message with 0 (zero) in the sequence number field, receiving system should respond with response whose MSA contains a sequence number one greater than the sequence number of the last transaction it accepted in the Expected Sequence Number field. If this value does not exist, the MSA should contain a sequence number of -1, meaning that the initial system will use the positive, non-zero sequence number of the first transaction it accepts as its initial sequence number.

When initiating system (client side) sent any message with -1 in the sequence number field, receiving system should respond with response contains a sequence number -1 in the expected sequence number field. The initiating system then resets its sequence number, using the non-zero positive sequence number for the next transaction it accepts initially.

Back to top

## Use Case Scenarios for Publish and Subscriber

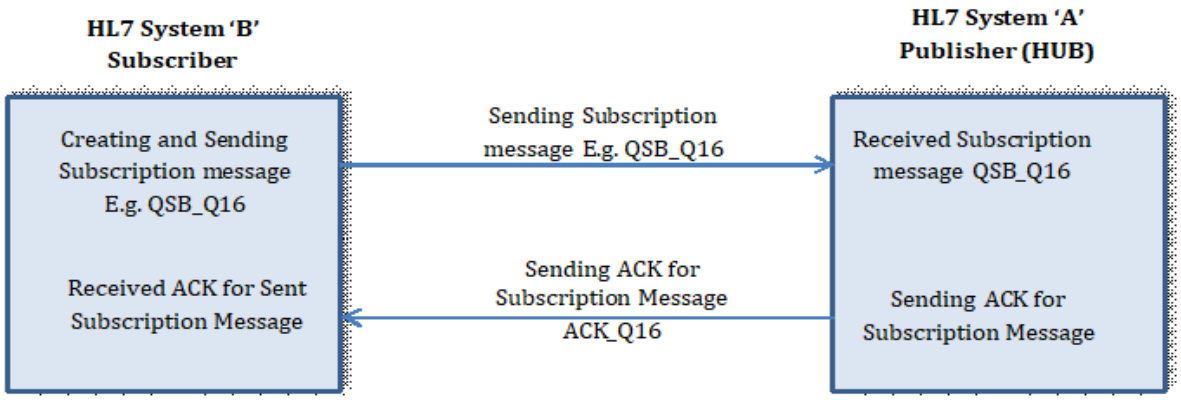



Diagram A: Sending Subscription Message to Publisher for Specific Data

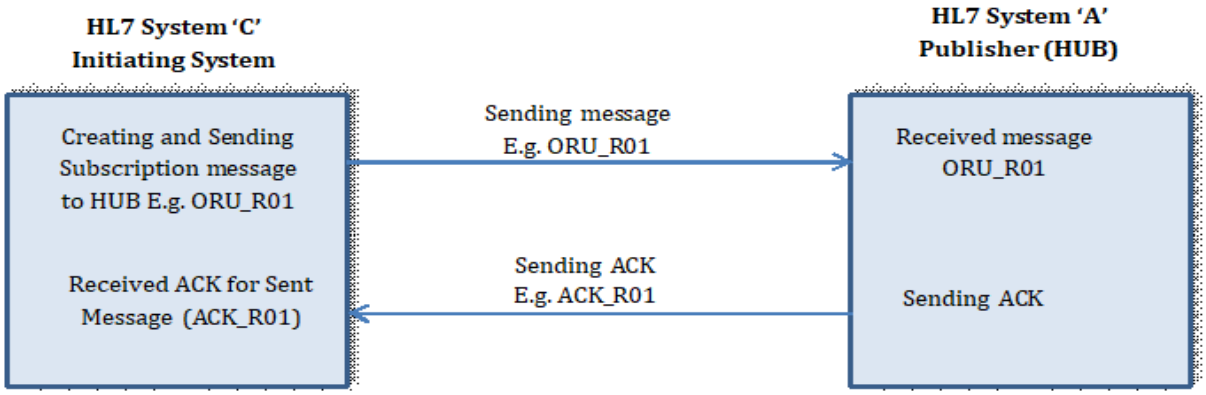



Diagram B: Sending Message to Publisher

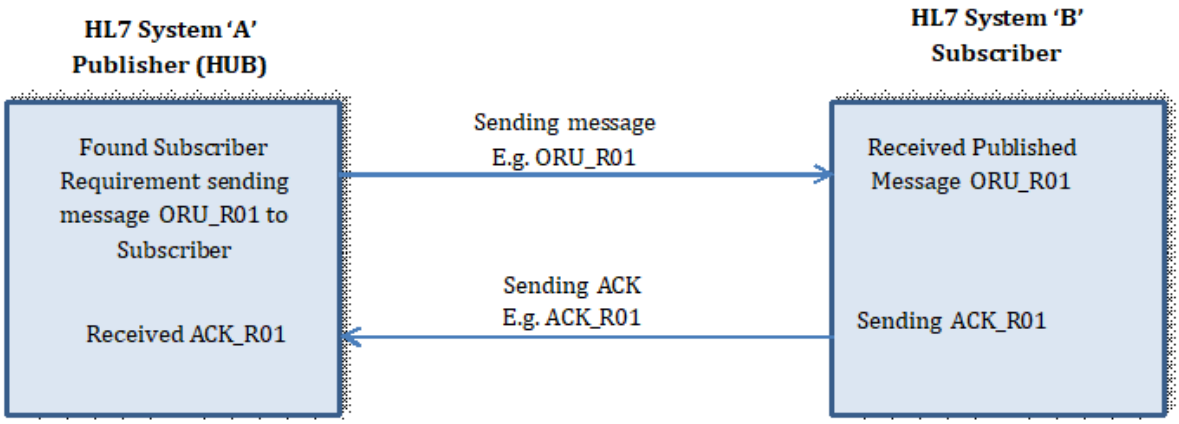



Diagram C: Sending Published Message to Subscriber

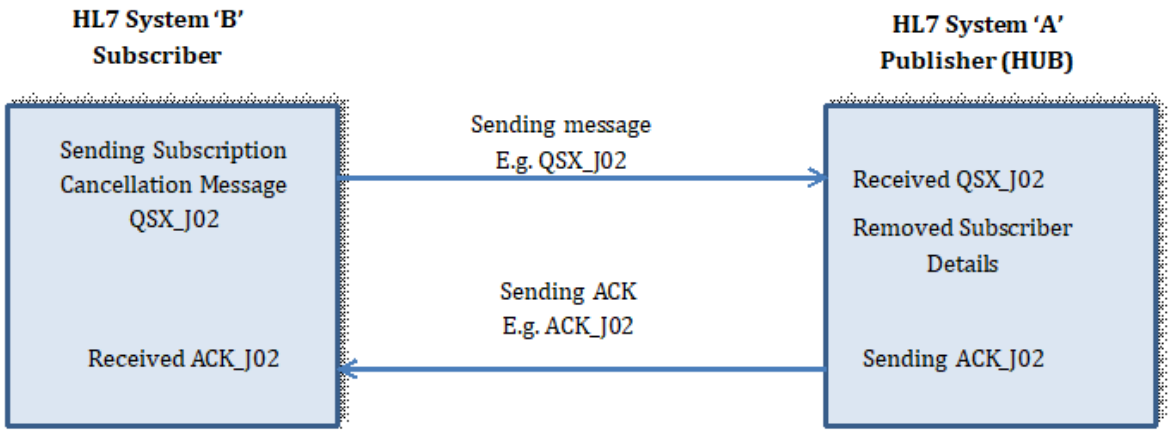



Diagram D: Sending Subscription Cancellation message to Publisher

Use case Scenarios:

Diagram A:

-As shown in diagram Publisher is mainly hub which receives data from different initiating system and subscription request for particular type of data and message. Subscriber is sending message QSB_Q16 which does registration for particular stream of data to the publisher. Publisher sends ACK for the same that subscription request is received successfully.

Diagram B:

One initiating system sends the message to publisher for which it sends ACK shows the successful reception of message ORU_R01.

Diagram C:

On receiving message ORU_R01 it is verified by publisher that whether it matches the data with subscriber requirement then it publishes all the messages to the Subscriber for all subscriber for which subscription is received.

Diagram D:

After receiving desired stream of data Subscriber can cancel its subscription towards publisher by sending cancellation query QSX_J02.which removes all the details whatever have towards publisher.

Back to top

## Configuration of a Publisher

"Publish and subscribe" refers to the ability of one system, the "Publisher", to offer a data stream that can be sent to recipient systems upon subscription. In one sense, the entire HL7 unsolicited

update paradigm, in which the sender sends out a stream of messages to recipients, is a kind of publish and subscribe mechanism. Subscriptions to unsolicited updates are established at interface set-up time when analysts on both sides agree to start sending a stream of data.

Basically publisher is one who possesses and transmits streams of data. The Publisher might be a mediator or a broker, such as an interface engine. The Publisher is not necessarily the system that collected the data, but it is the system willing to transmit it.

Subscription is a process/protocol that allows one system to request that prospective data be sent for a specified period of time, or for an open-ended period of time until further notice.

By taking consideration of current implementation of SDK, it can be viewed as there is Central Hub which acts as Publisher which receives Subscription message send from the Subscriber. Subscriber sends its requirement to publisher for particular data, and any other initiating system which exchange data with Hub i.e. Publisher.

Publisher is mainly HUB so it requires knowledge of any subscription message, subscription Cancellation message and also sources and recipients to receive message and Process the request.

For understanding of subscription message it requires the *Publish agent* which tells more about the message that coming message is Subscription message or Subscription cancellation message.

The default implementation of IPublishAgent is given by SDK. Where as user can provide his own implementation by implementing interface *IPublishAgent*.

All subscription and subscription cancellation messages hub can deal it separately by implementing interface *ISubscriptionListener* else it will be redirected to method *OnMessageReceive (ClientSession objClientSession)* of *IProcessHandler*.

Now set the implementer of ISubscriptionListener to client session like this:
objClientSession.setISubscriptionListener (ISubscriptionListener obj ISubscriptionListener);
Now all subscription and subscription cancellation message will be redirected to method of *onSubscriptionMessageReceive (ClientSession objClientSession)* and *onSubscriptionCancel (ClientSession objClientSession)* respectively. Now implementer of ISubscriptionListener can store all requirement of Subscriber while subscription with its address and remove data related to subscriber on cancellation of subscription.

Where it is already negotiated about the Subscription message type, Message Query name and at which port subscriber is listening for message for which it is subscribed for.

Back to top

## Configuration of a Subscriber

Subscriber is interested in particular type of Message and wants data of patient whose patient id is sent to publisher. Now for this it sends Subscription message in form of message QSB_Q16. It is already negotiated that this message will be used for the subscription of particular request.

Now subscriber just acts as initiator and sends the subscription message to publisher by Creating *User Session*.

```
//Creation of User Session.
UserSession objUserSession = new UserSession ();
```

For configuration of user session please refer link *Configuration of User Session*.

Now just sends the subscription message by using user session like this:

Subscription message can be created like this:

```
//Instantiate Query Recipient.
IMessageRecepient objQueryRecepient = new QryRecepient ();

//Creating subscription message QSB_Q16.
QSB_Q16 objQSB_Q16 = (QSB_Q16) objQueryRecepient.createMessage
(EnumMessageCode.QSB, EnumTriggerEvent.Q16);
```

Now all the Segments in it can be populated one by one like MSH, SFT, QPD, RCP, DSC.
In QPD segment user can specify the message query name.

For Segment population refer *Population of a Segment*. In this case in QPD segment in varies data type user can send the requested data. Now add these populated segments in message and send.

```
objUserSession.sendMessage (SubscriptionMessage objMessage);
```

```
//Creating subscription cancellation message QSX_J02.
QSX_J02 objQSX_J02 = (QSX_J02) objQueryRecepient.createMessage
(EnumMessageCode.QSX, EnumTriggerEvent.J02);
```

Now all the Segments in it can be populated one by one like MSH, SFT, and QID. In QID segment user can specify the message query name.

For Segment population refer *Population of a Segment*. In this case in QID segment in message Query name user can send the data for canceling particular type of message by this attribute. Now add these populated segments in message and send.

```
objUserSession.sendMessage (SubscriptionMessage objMessage);
```

While subscriber is acting as responding system for the various message which includes the receiving of those messages for which it has done subscription. This is negotiated between that for particular IP and Port Subscriber is acting as responding system.

Configuration of Initiating System:

This just act as initiating system which will generate any event which will be sending to Publisher.

For generating and sending this event to publisher it creates User Session like this:

```
//Creation of User Session.
UserSession objUserSession = new UserSession ();
```

For configuration of user session please refer link *Configuration of User Session*.

Now just sends the subscription message by using user session like this:

```
objUserSession.sendMessage (Message objMessage);
```

E.g. it has send ORU_R01 to publisher and gets the Acknowledgment for it.

According to this received ORU_R01 this message will be verified by Publisher which is checked against the requirements of Subscriber and this will be send for each subscriber for which subscription is received on its respective address by creating separate User Session for which Subscriber is listening on particular IP and Port which is known and negotiated between Publisher and Subscriber.

Back to top

## Introduction to Local Extension Protocol

HL7 protocol defines extension in specification of message, segment and data type through use of Local Extension capability. Local extension can be used for things which are not covered by specifications given by protocol. Extension in segment and message structure should follow the basic message/segment construction rules which are defined by HL7 standard.

HL7 SDK provides support for extension of message and segment. Through Local extension in SDK new message/segment can be created and existing message/segment can also be modified.

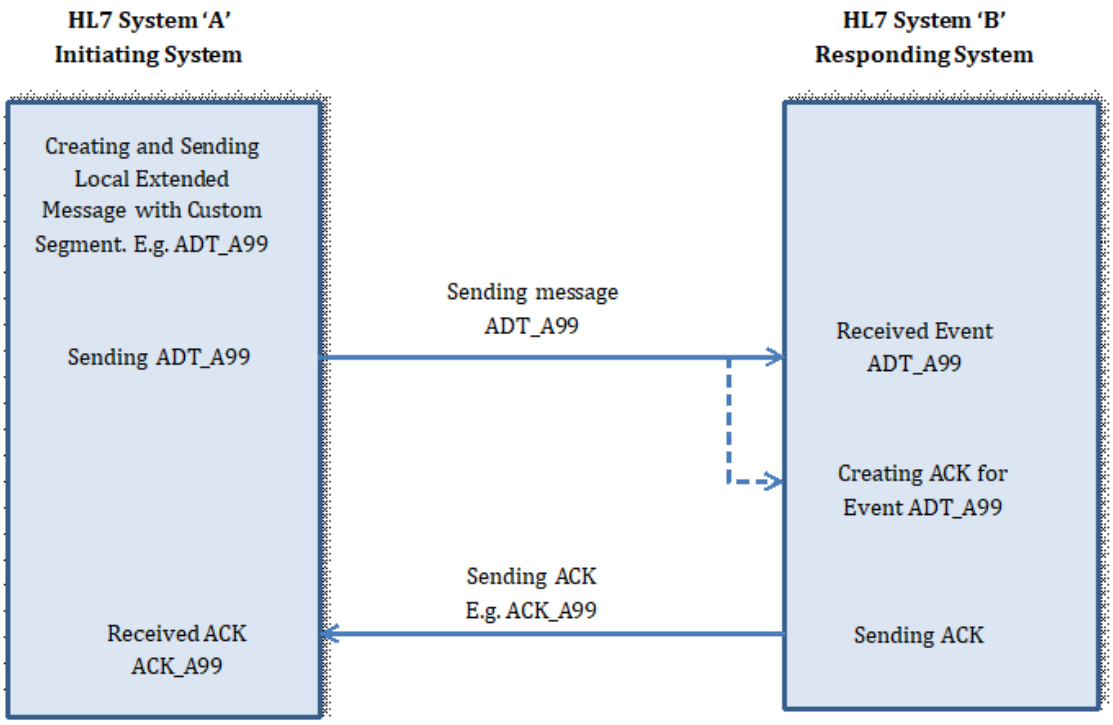



Diagram A: Diagrammatic representation of Local Extension prototype model with Local Extended Message

Use case Scenarios:

Diagram A:
-As shown in diagram initiating system creates locally extended message ADT_A99 which is kind of event which will be send to responding system. This responding system generates ACK for this event and sends it to Initiating system.
-In this way different type of locally extended messages with custom segments can be formed and send.

Back to top

## Implementation of Locally Extended DataType(Z-DataType)

Local extension for DataType includes modification in existing DataType structure or creation of new DataType. HL7 SDK supports both extensions for a DataType.

In HL7 SDK structure of a DataType is represented by *DataTypeMap*.

DataTypeMap is a list of ComponentItem. *ComponentItem* represents different attribute structure of a DataType.

For more details of DataType structure, DataTypeMap and ComponentItem refer "HL7

DataTypes".

A new datatype can be implemented by populating DataTypeMap. In HL7 SDK once an existing datatype is modified then it will be considered as a new datatype. For extension of an existing datatype a new class can be written in which modified DataTypeMap will be used accordingly. Name of this new datatype should be according to the naming constraints defined by HL7 v2.8.2 standard. New DataType class should be extending from PrimitiveDataType class or Composite DataType class or implementing IDataType . By extending from DataType class default parsing, serialization and validation process can be used, else it need to be implemented separately.

Modification in existing segment

SegmentMap modification for existing datatype:-

```
//creating DataTypeMapReader instance
IDataTypeMapReader objDataTypeMapReader =
DataTypeMapReader.createInstance();

//Retrieving DataTypeMap for AD DataType
IDataTypeMap objTempDataTypeMap =
objDataTypeMapReader.getDataTypeMap(EnumDataType.HL7_DT_AD.getValue());
//Making clone of DataTypeMap if clone is not created then actual AD datatype map
will be modified.
IDataTypeMap objDataTypeMap = objTempDataTypeMap.clone();

int[] table = new int[1];
table [0] = 8;
//Making a new ComponentItem with proper values.
IComponentItem objComponentItem = new
ComponentItem("simplefield",1,210,"IS",table,true);

//Adding ComponentItem at 4th position in new DataTypeMap.
objDataTypeMap.addComponentItem (objComponentItem, 3,
EnumMapModifyMode.AFTER);
```

Note: - *DataTypeMapReader* works as a factory of DataTypeMap for a specific datatype.

Similarly other ComponentItem can be added or replaced or deleted by specifying appropriate value for EnumMapModifyMode. Through this process a new DataType can be created by using structure of an existing DataType.

Creation of New DataType: -

New DataType should be extending from *PrimitiveDataType* or *CompositeDataType* class or implementing *IDataType*. If it is extending from Primitive class or Composite class then default parsing, serialization and validation process can be used. If new class is implementing IDataType all required definitions need to be provided.

For population and creation of DataTypeMap refer *DataTypMap and Component Item*

Back to top

## Implementation of Locally Extended Segment (Z-Segment)

Local extension for segment includes modification in existing segment structure or creation of new segment. HL7 SDK supports both extensions for a segment.

In HL7 SDK structure of a segment is represented by *SegmentMap*.

SegmentMap is a list of AttributeItem. *AttributeItem* represents different attribute structure of a segment.

For more details of segment structure, SegmentMap and AttributeItem refer “HL7 Segments”.

A new segment can be implemented by populating SegmentMap. In HL7 SDK once an existing segment is modified then it will be considered as a new segment. For extension of an existing segment a new class can be written in which modified SegmentMap will be used accordingly. Name of this new segment should be according to the naming constraints defined by HL7 v2.8.2 standard. New Segment class should be extending from Segment class or implementing *ISegment*. By extending from Segment class default parsing, serialization and validation process can be used, else it need to be implemented separately.

Modification in existing segment

SegmentMap modification for existing segment: -

```
//creating MessageMapReader instance
ISegmentMapReader objSegmentMapReader = SegmentMapReader.createInstance();

//Retrieving SegmentMap for EVN segment
ISegmentMap objTempSegmentMap =
objSegmentMapReader.getSegmentMap(EnumSegments.HL7_SEG_EVN.getValue());
//Making clone of SegmentMap if clone is not created then actual EVN
//segment map will be modified.
objSegmentMap = objTempSegmentMap.clone();

//Making a new AttributeItem with proper values.
IAttributeItem objAttributeItem = new AttributeItem ("simplefield", 2000, 4,  0, 3,
null,"IS", null, true, false);

//Adding AttributeItem at 4th position in new SegmentMap.
objSegmentMap.addAttributeItem (objAttributeItem, 3,
EnumMapModifyMode.AFTER);
```

Note: - *SegmentMapReader* works as a factory of SegmentMap for a specific segment.

Similarly other AttributeItem can be added or replaced or deleted by specifying appropriate value for EnumMapModifyMode. Through this process a new segment can be created by using structure of an existing segment.

Creation of New Segment: -

New Segment should be extending from *Segment* class or implementing *ISegment*. If it is extending from Segment class then default parsing, serialization and validation process can be used. If new class is implementing ISegment all required definitions need to be provided.

If new class is extending from *Segment* class then *SegmentMap* need to be populated according to the structure.

SegmentMap can be populated by populating structure of AttributeItem of a segment. For example ZL7 segment which contains structure given in table below:
//Population of new SegmentMap

HL7 Attribute Table – ZL7 Segment

| SEQ | LEN | C.LEN | DT | OPT | RP/# | TBL# | ITEM# | ELEMENT NAME |
|---|---|---|---|---|---|---|---|---|
| 1 | 0..4 | | IS | R | | | 09999 | setID_ZL7 |
| 2 | 0..20 | | ST | R | Y | | 09998 | Simple Field |
| 3 | 0..80 | | CWE | O | | | 09997 | Complex Field |

```
//Population of SegmentMap for ZL7 segment
SegmentMap objSegmentMap = new SegmentMap();
IAttributeItem objAttributeItem = new AttributeItem("setID_ZL7"
,9999,1,0,4,null,"IS", null, false, false);
objSegmentMap.addAttributeItem(objAttributeItem);
objAttributeItem = new AttributeItem("Simple Field", 9998, 2, 0, 20, null, "ST", null,
true, false);
objSegmentMap.addAttributeItem(objAttributeItem);
objAttributeItem = new AttributeItem("Complex Field", 9997, 3, 0, 80, null,
"CWE",null, false, true);
objSegmentMap.addAttributeItem(objAttributeItem);
```

Through this way a SegmentMap for a new segment can be implemented.

Back to top

## Implementation of Locally Extended Message (Z-Message)

Local extension for message includes modification in existing message structure or creation of new message. HL7 SDK supports both extensions for a message.

In HL7 SDK structure of a message is represented by MessageMap. MessageMap is a list of SegmentItem. SegmentItem represents properties of a member segment.

For more details of message structure, MessageMap and SegmentItem refer *HL7 Messages*.

A new message can be implemented by populating MessageMap according to the message structure. In HL7 SDK once an existing message is modified then it will be considered as a new message. For extension of an existing message a new class can be written in which modified *MessageMap* will be used accordingly. Name of this new message should be according to the naming constraints defined by HL7 v2.8.2 standard. New Message class should be extending from *Message* class or implementing *IMessage*. By extending from Message class default parsing, serialization and validation process can be used, else it need to be implemented separately.

Modification in existing message: -

MessageMap modification for existing message

Sample code given below is showing modification in ADT_A01 message and creating new message ADT_A99 using existing. ADT_A99 message is containing ZL7 segment at the third position so in from ADT_A99 MessageMap segment from third position is being replaced by ZL7 segment.

```
//Initializing MessageMapReader for Patient Administration System
IMessageMapReader objMessageMapReader =
MessageMapReader.createInstance(EnumHL7System.HL7_SYSTEM_PATIENTADMINISTR
ATION);

//Retrieving MessageMap of ADT_A01 message.
IMessageMap objTempMessageMap =
objMessageMapReader.getMessageMap(EnumMessageCode.ADT, EnumTriggerEvent.A01);

//Creating clone of MessageMap otherwise MessageMap for existing will
//be modified.
objMessageMap = objTempMessageMap.clone();

//Creating new SegementItem for ZL7 segment
ISegmentItem objZL7SegmentItem = new SegmentItem();

objZL7SegmentItem.setName(UserDefinedSegmentEnumeration.ZL7);
objZL7SegmentItem.setSegmentCardinality(EnumSegmentCardinality.SINGLE_COMPULSA
RY);

//Adding SegmentItem for ZL7 at 3rd position.
objMessageMap.addSegmentItem(objZL7SegmentItem,3,EnumMapModifyMode.REPLACE);
```

Note:- *MessageMapReader* works as a factory of MessageMap for a specific Message.

Similarly other SegmentItem for new segment or group can be added or replaced or deleted by specifying appropriate value for EnumMapModifyMode. Through this process a new message can be created by using structure of an existing message.

Population of new MessageMap: -

MessageMap can be populated by populating information of all member segments in terms of SegmentItem. For example ADT_A99 message which contains structure given in table below:

ADT^A99^ADT_A99: ADT Message

| Segments | Description | Status | Chapter |
|---|---|---|---|
| MSH | Message Header | | 2 |
| [{ SFT }] | Software Segment | | 2 |
| ZL7 | ZL7 Segment | | |
| [ PD1 ] | Additional Demographics | | 3 |
| [{ | --- PROCEDURE begin | | |
| PR1 | Procedures | | 6 |
| [{ ROL }] | Role | | 15 |
| }] | --- PROCEDURE end | | |
| [ PDA ] | Patient Death and Autopsy | | 3 |

```
//Population of MessageMap for ADT_A99 message

MessageMap objMessageMap = new MessageMap();

ISegmentItem objMSHSegmentItem = new SegmentItem();
objMSHSegmentItem.setName(EnumSegments.HL7_SEG_MSH);
objMSHSegmentItem.setSegmentCardinality(EnumSegmentCardinality.SINGLE_COMPULSARY);
objMessageMap.addSegmentItem(objMSHSegmentItem);

ISegmentItem objSFTSegmentItem = new SegmentItem();
objSFTSegmentItem.setName(EnumSegments.HL7_SEG_SFT);
objSFTSegmentItem.setSegmentCardinality(EnumSegmentCardinality.MULTIPLE_OPTIONAL);
objMessageMap.addSegmentItem(objSFTSegmentItem);

ISegmentItem objZL7SegmentItem = new SegmentItem();
ObjZL7SegmentItem.setName(UserDefinedSegmentEnumeration.ZL7);
ObjZL7SegmentItem.setSegmentCardinality(EnumSegmentCardinality.SINGLE_COMPULSARY);
objMessageMap.addSegmentItem(objZL7SegmentItem);
```

```
ISegmentItem objPD1SegmentItem = new SegmentItem();
objPD1SegmentItem.setName(EnumSegments.HL7_SEG_PD1);
objPD1SegmentItem.setSegmentCardinality(EnumSegmentCardinality.SINGLE_OPTION
AL);
objMessageMap.addSegmentItem(objPD1SegmentItem);

ISegmentItem objGroupSegmentItem = new SegmentItem();
objGroupSegmentItem.setName(EnumSegments.HL7_GROUP_PROCEDURE);
objGroupSegmentItem.setSegmentCardinality(EnumSegmentCardinality.
MULTIPLE_OPTIONAL);
objGroupSegmentItem.setIsGroup(true);

{
MessageMap objMessageMapInGroup = new MessageMap();

ISegmentItem objPR1SegmentItem = new SegmentItem();
objPR1SegmentItem.setName(EnumSegments.HL7_SEG_PR1);
objPR1SegmentItem.setSegmentCardinality(EnumSegmentCardinality.SINGLE_COMPU
LSARY);
objMessageMapInGroup.addSegmentItem(objPR1SegmentItem);
ISegmentItem objROLSegmentItem = new SegmentItem();
objROLSegmentItem.setName(EnumSegments.HL7_SEG_ROL);
objROLSegmentItem.setSegmentCardinality(EnumSegmentCardinality.
MULTIPLE_OPTIONAL);
objMessageMapInGroup.addSegmentItem(objROLSegmentItem);

objGroupSegmentItem.setGroupItems(objMessageMapInGroup);

}
objMessageMap.addSegmentItem(objGroupSegmentItem);

ISegmentItem objPDASegmentItem = new SegmentItem();
objPDASegmentItem.setName(EnumSegments.HL7_SEG_PDA);
objPDASegmentItem.setSegmentCardinality(EnumSegmentCardinality.SINGLE_OPTIO
NAL);
objMessageMap.addSegmentItem(objPDASegmentItem);
```

Note: - UserDefinedSegmentEnumeration defines enumerated values for name of Z-segments. This enum implements *ISegmentKey* interface.

Through this way a MessageMap for a new message can be implemented.

Back to top

## Local Extension Configuration

Locally extended segments and messages should be registered on *LocalExtensionAgent* for working in HL7 communication.

If a Z-Segment or Z-Message is not registered with LocalExtensionAgent or LocalExtensionAgent itself not available then new Z-segment/Z-message can not be processed.

Registration Process: -

```
//Initializing LocalExtensionAgent
LocalExtensionAgent objLocalExtensionAgent = new LocalExtensionAgent();

//Registration process of segment
objLocalExtensionAgent.registerSegment(strSegmentName, strQualifiedName);
```

Note: - strSegmentName defines name of Z-Segment.strQualifiedName defines qualified name for segment so that object can be initialized.

```
//Registration process of message
objLocalExtensionAgent. registerMessage(strMsgCode,strTriggerEvent,
strQualifiedName)
```

Note: - strMsgCode defines message code for message.strTriggerEvent defines trigger event for message.strQualifiedname defines qualified name for message so that object can be initialized.

Back to top

## Process communication for Z-Message or Z-Segment

For Z-Segment parsing *HL7Parser* should have knowledge of segment. This knowledge can be given to HL7Parser through LocalExtensionAgent. Z-Segment should be registered with *LocalExtensionAgent* and this agent should be set on HL7Parser.

```
//Initializing LocalExtensionAgent
LocalExtensionAgent objLocalExtensionAgent = new LocalExtensionAgent();

//Registration process of segment in java
objLocalExtensionAgent.registerSegment(strSegmentName, strQualifiedName);
```

Note: - strSegmentName defines name of Z-Segment.strQualifiedName defines qualified name for segment so that object can be initialized.

```
//Initializing HL7Parser and setting LocalExtensionAgent
HL7Parser objHL7Parser = new HL7Parser();
objHL7Parser.setLocalExtensionAgent(objLocalExtensionAgent);
```

To provide space for Z-Segments and Z-Messages in communication LocalExtensionAgent which contains knowledge should be set on *MessageFactory*.This factory should be set on *UserSession* and *ServerSession* while configuring session for communication.

For e.g.:

```
//Initializing LocalExtensionAgent
LocalExtensionAgent objLocalExtensionAgent = new LocalExtensionAgent();

//Registration process of message
objLocalExtensionAgent.registerMessage("ADT","A99","cdac.medinfo.sdk.hl7282.hl7net
.testnetwork.localmessages.ADT_A99");

//Registration process of segment
objLocalExtensionAgent.registerSegment("ZL7","cdac.medinfo.sdk.hl7282.hl7net.testnet
work.localmessages.ZL7");

//Configuring MessageFactory for LocalExtensionAgent
MessageFactory objMessageFactory = MessageFactory.createInstance();

objMessageFactory.setLocalExtensionAgent(objLocalExtensionAgent);
```

Back to top