DHCP HEALTH LEVEL SEVEN

(HL7)

DEVELOPER MANUAL:

DYNAMIC ADDRESSING SUPPLEMENT

Version 1.6*14

July 1998

Department of Veterans Affairs
V*IST***A** Software Development
Open**V***IST***A** Product Line

Table of Contents

1. Introduction

Unconditional broadcasting (the addressing mechanism used by the HL7 package prior to patch HL*1.6*14) is usually sufficient to distribute messages within a single site. However, when used for wide-area messaging, it results in too much unnecessary network traffic and processing load, and unreasonable complexity in interface design. At the enterprise level, without some refinement of the message addressing mechanisms, fixed point-to-point interface definitions for unconditional broadcasting would mushroom to an unmanageable number.

To help solve these problems, patch HL*1.6*14 introduces *dynamic addressing*:

- Recipients *on remote systems* for a message can be set dynamically for each message at run-time.
- Routing of messages to remote systems can be determined based on an individual message's content and business rules, not just on a broad event point.

Patch HL*1.6*14 also introduces a *subscription registry*:

- Applications can use the registry as a convenient location to maintain a list of subscribers on remote systems for dynamically addressed messages.

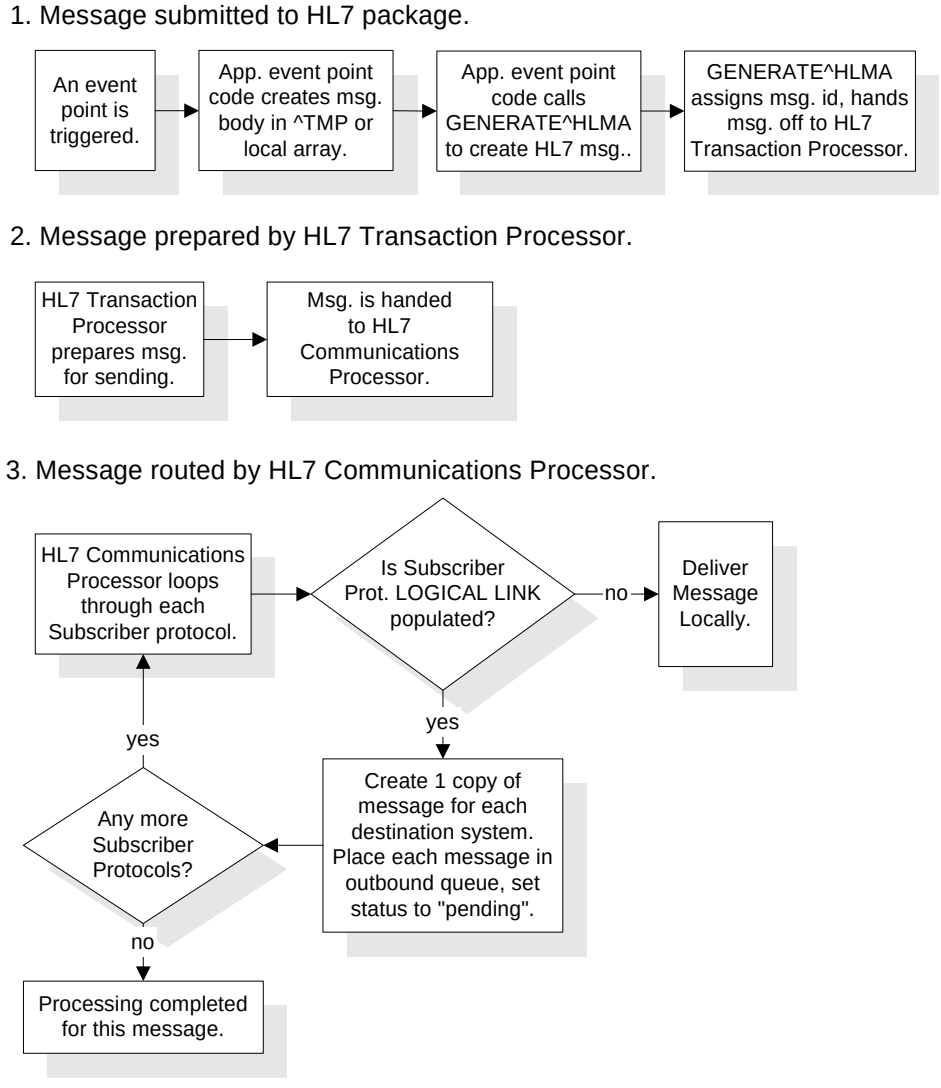
Dynamic addressing and the subscription registry enable large-scale use of HL7 messaging over a wide area network by **V*IST*A** packages, including the Clinical Information Resources Network (CIRN).

2. Dynamic Addressing

2.1. How Addressing Worked Prior to Patch HL*1.6*14

To distribute HL7 messages to interested subscribers, the unpatched DHCP HL7 V. 1.6 uses an *unconditional broadcast mechanism*. Each message is broadcast to all applications that attached Subscriber protocols to that Event Point protocol. Each receiving application then filters out unwanted messages.

Figure 2-1: Unconditional Broadcast Mechanism



2.2. How Dynamic Addressing Works

Use the HLL("LINKS") array to dynamically specify one or more recipients (on remote systems) for an HL7 message. Add one node per recipient to the HLL("LINKS") array. The expected format for each node is:

```
HLL("LINKS",n)="DESTINATION PROTOCOL^DESTINAION LOGICAL LINK"
```

Subscript *n* is a unique, arbitrary integer subscript. Any additional ^-pieces beyond piece 2 are ignored for any HLL array node.

For example:

```
HLL("LINKS",1)=CIRN CLIENT/ROUTER^TEST LLP
```

The pieces of an HLL("LINKS",n) act as follows:

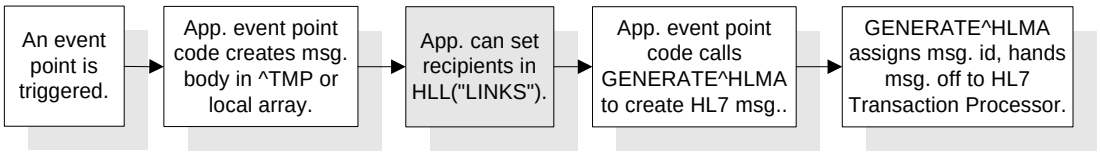
Piece	Piece Contents	Description
1	Destination Protocol (name or IEN)	(mandatory) The protocol associated with the application that will receive and process the HL7 message.
2	Destination LOGICAL LINK (name or IEN)	(mandatory) The Logical Link (file #870) that identifies the path to the target system for the HL7 message. The destination **must** be on a different system. It overrides any Logical Link setting in the local copy of the subscriber protocol specified in piece 1 (HL7 Subscriber Protocol).

You can dynamically add recipients to the HLL("LINKS") array for an **outbound** HL7 message at the following points:

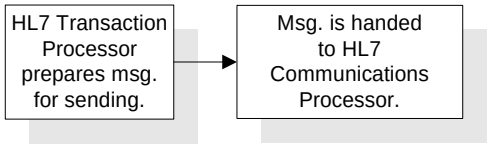
- When the sending application prepares a message for sending. Set nodes in HLL("LINKS") prior to calling GENERATE^HLMA to send the message.
- When, on a sending system, a local subscriber protocol (to the event point protocol associated with an outbound message) is set up to act as a router. See "*How To Set Up a Subscriber Protocol as a Router*" later in this chapter.

Figure 2-2: Broadcast Process Including Dynamic Addressing

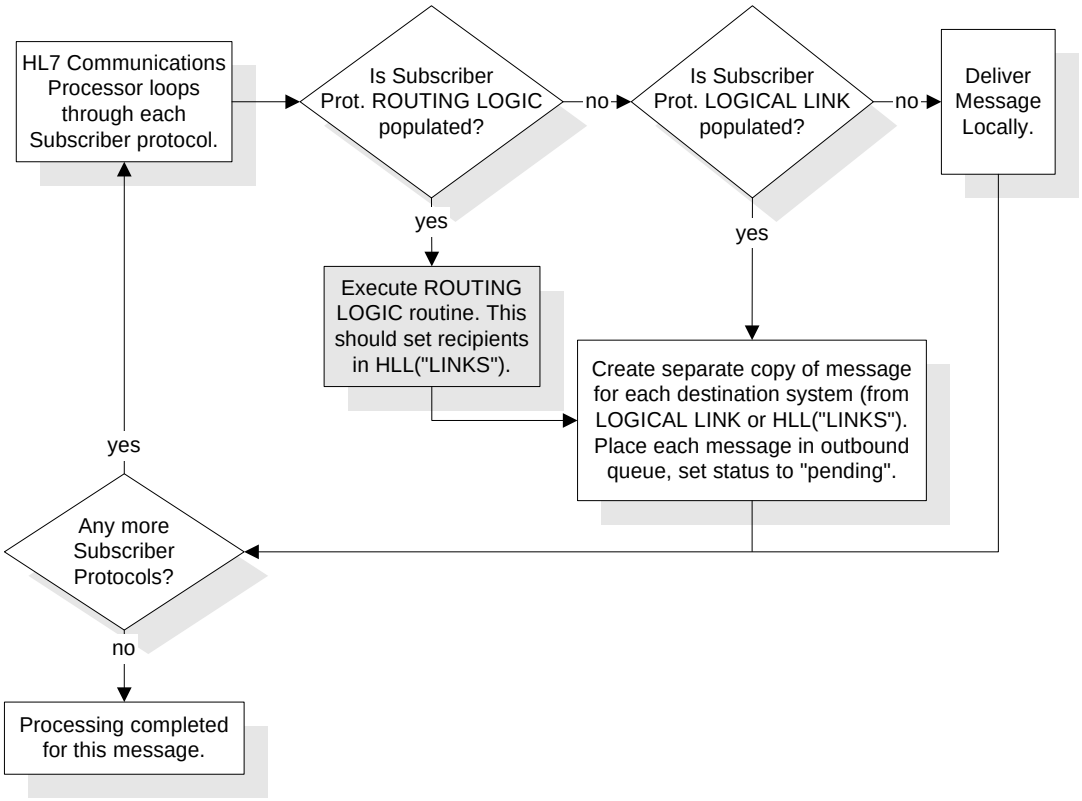
1. Message submitted to HL7 package.



2. Message prepared by HL7 Transaction Processor.



3. Message routed by HL7 Communications Processor.



2.3. How To Create a "Router" Subscriber Protocol

You can set up a subscriber protocol on the sending system to act as a router. A router's only purpose is to dynamically select message recipients for the outbound message. It can examine the message text, and conditionally route messages to interested systems based on message content and business rules.

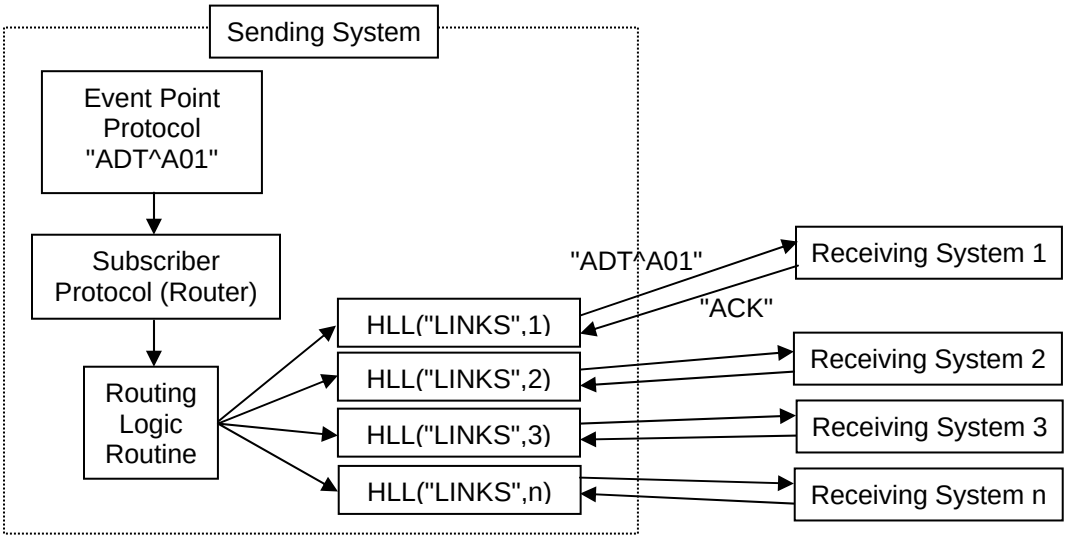
To set up a Subscriber protocol to be a Router:

1. Create a routing routine. The routine can examine each line of a message by executing 'X HLNEXT' in the documented manner. Using the environment variables HLFS (field separator) and HLECH (encoding characters), the routing routine can parse each segment for the appropriate fields to screen on to determine who the supplemental addressees are. At the time the routine is executed, no message header has been created yet; the router is scanning the original message text in the HL7 MESSAGE TEXT file.
2. To add additional message recipients, the routing routine should add one node to the HLL("LINKS") array for each application/system it determines should be a recipient. Remember that the format of HLL("LINKS") is

   ```
   HLL("LINKS",n)="DESTINATION PROTOCOL^DESTINATION LOGICAL LINK"
   ```

   where name or IEN can be used for both protocol and logical link.
3. Use the Interface Workbench to do the initial set up of the router protocol as a standard Subscriber protocol.
4. Set the ROUTING LOGIC field of the router protocol to a routine that you will use to perform routing. With ROUTING LOGIC defined:
   - The router protocol's PROCESSING ROUTINE, if defined, is ignored on the sending system.
   - The router protocol's LOGICAL LINK field, if defined, is ignored on the sending system.
5. Attach the new "router" Subscriber protocol to an event point protocol. It will then perform routing for that event point's messages.

Figure 2-3: Subscriber Protocol Acting as a Router



2.4. Responding to Dynamically Addressed Messages

On a receiving system, the subscriber protocol a message is addressed to does not have to be attached to any particular event point protocol for the message to be processed.

Because there is no linkage to an event point protocol, another means is needed to determine who to address replies to:

- For dynamically addressed messages received over serial connections and TCP/IP, the channel is assumed to be open and bi-directional. The response is immediate by default for both commits and application ACKs.

- For dynamically addressed messages received through Mailman, commit responses are addressed by the HL7 package using the Mailman environment variable, XMFROM (which contains the domain of origin, and can be resolved to the equivalent logical link).

- For dynamically addressed messages received through MailMan, to send an application ACK (typically needed for queries only), set a node in HLL("LINKS") based on the contents of the MailMan XMFROM variable, prior to calling GENACK^HLMA1. This in effect uses dynamic addressing to reply to a dynamically addressed message.

2.5. What About Local Recipients?

Dynamic addressing **cannot**, at this time, route messages to applications on the local system. This means that you must still use the unconditional broadcast mechanism to send messages to local recipients.

If a message needs to be processed/filed on the current (sending) system as well as being routed dynamically to remote systems, set up a separate Subscriber protocol for each local recipient. It should be a traditional V. 1.6 style Subscriber protocol, with its LOGICAL LINK field null (so that the message is received on the sending system) and its PROCESSING ROUTINE set to the routine that will process the (inbound) message on the current (local) system.

2.6. Avoiding Duplicate Messages

Duplicate messages occur when the same message text is sent to the same **application** on the same **system** more than once. For dynamically addressed messages, the HL7 package eliminates the possibility of duplicate destinations. If you set duplicate nodes into the HLL("LINKS") array, those duplicates are filtered out by the HL7 package.

However, if a Subscriber protocol has a fixed logical link definition (not a router) it is possible for duplicate messages to occur in the following, highly unlikely situations:

- If another Subscriber protocol to the same event point has the same logical link definition and receiving application.
- If another Subscriber protocol to the same event point is a router, and it dynamically addresses a message to the same logical link and recipient.

2.7. Role of the Subscription Registry

The subscription registry (described in the "Subscription Registry" chapter) is a convenient location where potential message recipients can be stored. An API is provided to store and retrieve subscribers.

The subscription registry, however, does not need to be involved for dynamic addressing to be performed. If application code or routing logic can determine the recipients of a message (destination logical link) on its own, it can go ahead and set nodes in HLL("LINKS") without consulting the subscription registry.

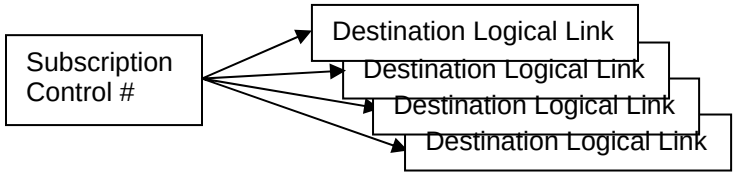
3. Subscription Registry

3.1. Overview

The subscription registry, also introduced by patch HL7*1.6*14, is a location where applications can maintain a list of subscribers for dynamically addressed messages.

Each registry entry consists of a subscription control number and a multiple containing destination subrecords (and related information) for each subscriber. This structure allows a single subscription control number to hold multiple destinations.

Figure 3-1: Subscription Registry Record



Applications are responsible for knowing which records in the subscription registry pertain to their application. For example, CIRN has a pointer from the PATIENT file to the subscription registry. This is the mechanism CIRN uses to associate a particular subscription registry control number (and all destinations within that control number) with a particular patient.

The subscription registry should only be updated through the following API calls (described in the "Dynamic Addressing/Subscription Registry API" chapter:

- Create new entries in the subscription registry with the $$ACT^HLSUB entry point.
- Add destination records to a subscription registry entry using the UPD^HLSUB entry point.
- Retrieve active destination records from a subscription registry entry using the GET^HLSUB entry point.

Purging functionality for the registry is not implemented in patch HL*1.6*14. It may be implemented in a future patch.

3.2. SUBSCRIPTION CONTROL File

The subscription registry is physically stored in the SUBSCRIPTION CONTROL File (#774). Its file structure is as follows:

```
FIELD     FIELD
NUMBER    NAME

.01       NUMBER (RNJ8,0X), [0;1]
1         DESTINATION (Multiple-774.01), [TO;0]
          .01  DESTINATION (MF), [0;1]
          1    RECEIVING APPLICATION (P771'), [0;2]
          2    DOMAIN (P4.2'), [0;3]
          3    LOGICAL LINK (P870'), [0;4]
          4    TYPE (S), [0;5]
          5    CREATION DATE/TIME (D), [0;6]
          6    ACTIVATION DATE/TIME (D), [0;7]
          7    TERMINATION DATE/TIME (D), [0;8]
          8    MODIFICATION DATE/TIME (Multiple-774.18), [HX;0]
               .01  MODIFICATION DATE/TIME (MD), [0;1]
               1    LAST CREATION DATE/TIME (D), [0;2]
               2    LAST ACTIVATION DATE/TIME (D), [0;3]
               3    LAST TERMINATION DATE/TIME (D), [0;4]
               4    LAST SUBSCRIPTION TYPE (S), [0;5]
```

Field Name	Description
Destination	Full network path to the receiving application, in a format similar to a mail address: RECEIVING APP@DOMAIN or RECEIVING APP@LOGICAL LINK. The contents of this field are automatically created by the UPD^HLSUB call. Note that both Domain and Receiving Application are, at this time, reserved for future use.
Receiving Application	Reserved for future use.
Domain	Reserved for future use.
Logical Link	The HL7 1.6 address of the receiving system (pointer to the HL LOGICAL LINK file).
Type	Subscription type (arbitrary; mainly for use by CIRN). Current types supported are: 0=patient descriptive only; 1=patient clinical and descriptive; 2=other (locally defined). Non-CIRN applications should probably use type 2.
Creation Date/Time	Date the destination subrecord was created.
Activation Date/Time	Date the subrecord should be considered active. GET^HLSUB only returns records considered active.
Termination Date/Time	Date the subrecord should be considered inactive. GET^HLSUB only returns records considered active.
Modification Date/Time	A multiple containing the history of modifications to this subscription registry record.

3.3. Case Study: CIRN

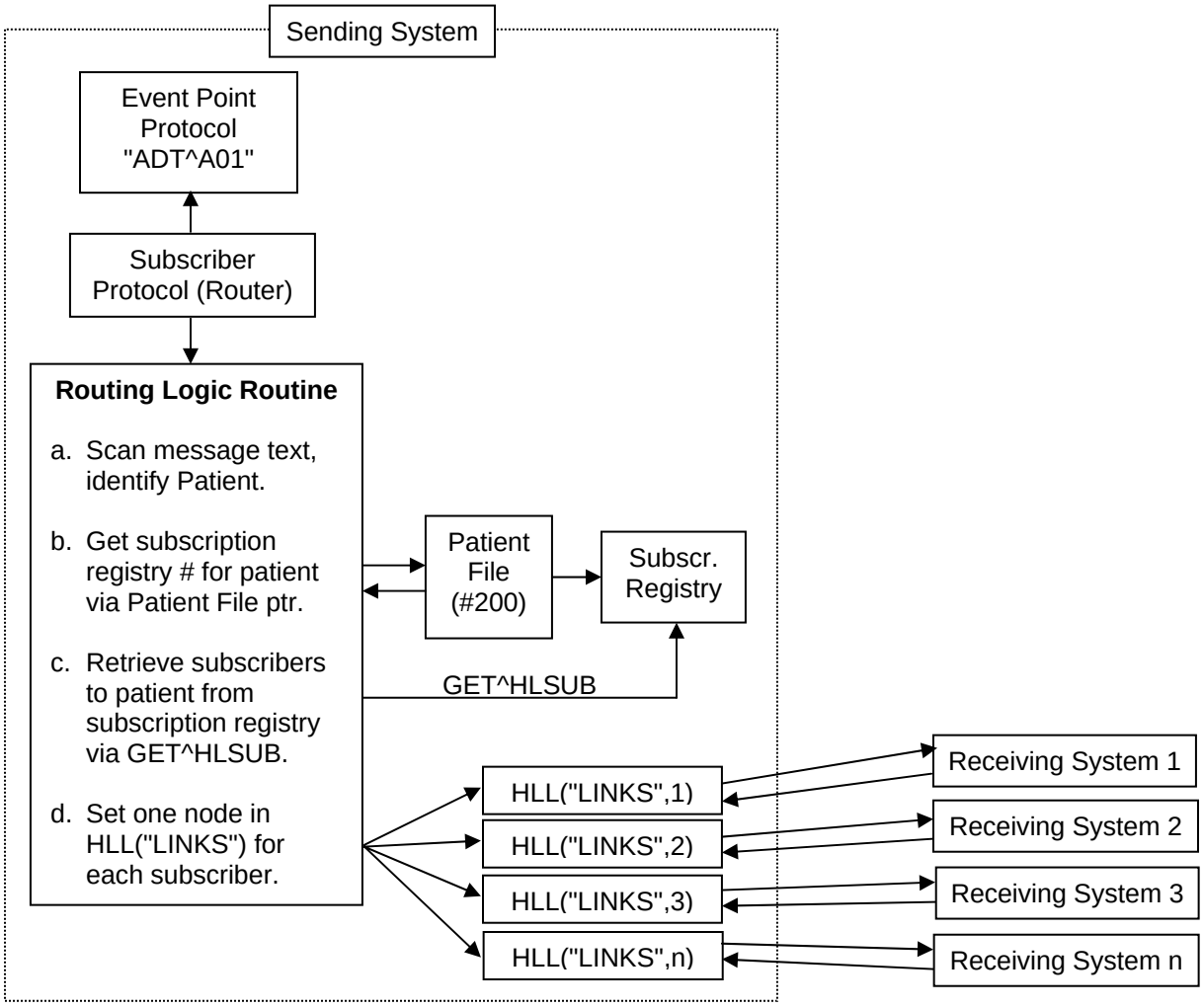
The first use of the subscription registry is by the CIRN package. CIRN provides message routers that scan messages, dynamically forwarding each message to all subscribers of the patient the message pertains to. This allows clinical information to be updated appropriately when a patient is seen at several facilities.

CIRN uses the subscription registry in the following manner:

1. CIRN creates an entry in the subscription registry with $$ACT^HLSUB for each patient it processes. A pointer from the Patient file to the Subscription Control file for the patient in question maintains the link between patient and subscription registry record.
2. When another application/facility becomes interested in a patient (e.g., the patient is seen at another facility) that application sends a subscription message to the patient's primary facility requesting to subscribe to that patient. When the message is processed, a call is made to UPD^HLSUB to add/update the destination for that patient's subscription registry record.
3. CIRN "Patient" router subscriber protocols subscribe to most Event Point protocols. The routers scan outgoing messages, parsing out the patient name. They dynamically route messages pertaining to a particular patient to all destinations listed in that patient's subscription registry record. The routers use GET^HLSUB to retrieve destinations for a particular patient.

CIRN routes messages based on the patient (implemented by a pointer from the Patient file to the Subscription Registry). Other applications should maintain their own associations with records they add to the subscription registry (one way is by pointing from one of that application's files to the subscription registry).

Figure 3-2: CIRN Use of Routers and the Subscription Registry



4. Dynamic Addressing/Subscription Registry API

4.1. $$ACT^HLSUB

Use this entry point to add a new record to the subscription registry. The only field populated in the new record is NUMBER (subscription control number). To add subscribers for this record, use the UPD^HLSUB call. Your application is responsible for tracking each subscription registry record it adds.

Usage $$ACT^HLSUB

Input None.

Output Return Value: A unique number that can be used to reference this registry entry in the future.

Example

```
>S X=$$ACT^HLSUB W X
2992
```

4.2. GET^HLSUB

Use this entry point to retrieve active subscriber records from a subscription registry entry.

Usage GET^HLSUB(subnum,subtype,client_p,.out_array)

Input

subnum: (required) Subscription control number.

subtype: (optional) Subscription type. Use this to return subscribers based on their TYPE field. If null, all subscribers for this subscription registry entry are returned. If 0, types 0 and 1 are returned. If 1, only type 1 is returned. If 2, only type 2 is returned.

client_p: (optional) Valid HL7 Client Protocol name for message routing. GET^HLSUB uses this as piece 1 of each record it adds to the output array. If your are making this call specifically to dynamically route a message, pass this parameter so that each record added to the output array is added in the format required for message routing -- the format of the HLL("LINKS") array. If null, piece 1 of each record in the output array will also be null.

out_array: (required) Used by GET^HLSUB for output; pass it by reference. GET^HLSUB adds any matching subscribers to this array as output. If it already exists, it will be appended to. Typically you would pass the HLL array, because HLL("LINKS") can then be used directly for routing by the HL7 package.

Output For each qualifying destination record found in the subscription registry entry, one array record is added to the output array (passed by reference as an input parameter)

ARRAY("LINKS",n) = Subscription Record

Each subscription record is in the format:

Piece	Description
1	Client Protocol name (if passed)
2	Logical Link name
3	Subscription Destination
4	(reserved for future use)
5	(reserved for future use)
6	Subscription Logical Link (IEN)
7	Subscription Type
8	Subscription Creation Date/Time
9	Subscription Activation Date/Time
10	Subscription Termination Date/Time

Description This call returns the list of subscribers for a particular subscription registry record. The HL7 package uses this array as the list of recipients to dynamically route a message to.

You can make multiple calls to GET^HLSUB. For example, to route a message to both Clinical and Descriptive subscribers of a particular subscription registry entry, first call GET^HLSUB with sub_type set to '0', and then call it with sub_type set to '1.' If the output array already exists, it will be appended to.

Examples

1. Return all subscriber information for Subscription Control record #4.

```
     >K HLL D GET^HLSUB(4,,,.HLL) ZW HLL

HLL("LINKS",1)=
     ^TEST LLP^@TEST LLP^^^11^2^2980414.141314^2980414.141314
     ^2980430.0001
```

2. Return all subscribers of type 2 for Subscription Control record #4, specifying the client protocol for message routing:

```
     >K HLL D GET^HLSUB(4,2,"CIRN CLIENT/ROUTER",.HLL) ZW HLL

HLL("LINKS",1)=
     CIRN CLIENT/ROUTER^TEST LLP^@TEST LLP^^^11^2^2980414.141314
     ^2980414.141314^2980430.0001
```

4.3. UPD^HLSUB

Use this entry point to add a new subscriber record or edit an existing subscriber record, in an existing subscription registry entry. Subscriber records are keyed on the Logical Link name: If an existing subscriber matches the Logical Link name, that subrecord is updated; otherwise, a new subrecord is added.

Usage UPD^HLSUB(sub_num,llname,sub_type,act_date,term_date,rec_app,.err_array)

Input

sub_num:	(required) Subscription Control Number to add/edit destination for.
llname:	(required) Logical Link name in file 870 for subscriber.
sub_type:	(optional) Subscription Type (arbitrary; types 0 and 1 mainly for use by CIRN). If null, defaults to type 0. Current types supported are: 0=patient descriptive only 1=patient clinical and descriptive 2=other (locally defined)
act_date:	(optional) Activation date/time. Time is required in addition to date. Defaults to 'now'. Subscribers are considered inactive before this date. GET^HLSUB won't retrieve inactive records.
term_date:	(optional) Termination date/time. Defaults to open-ended. Time is optional; if not included, will be appended to the date as ".0001". Passing null deletes the current termination date. Subscribers are considered inactive after this date; GET^HLSUB won't retrieve inactive records.
rec_app:	Reserved for future use. Pass null or no value for this parameter.
err_array:	(optional) Used to return error messages resulting from this call. Pass by reference.

Output

err_array	Error messages are returned in this variable (passed by reference as an input variable). Possible errors are: HLER(1)="Missing subscription control number" HLER(2)="Missing network address." HLER(4)="Invalid Subscription Control number" HLER(5)="Invalid Logical Link"

Example

```
K ERR D UPD^HLSUB(5,"TEST LLP",1,,,,.ERR)

                          resulting subscription record:

    NUMBER: 5                        DESTINATION: @TEST LLP
    LOGICAL LINK: TEST LLP           TYPE: Patient Clinical and Descriptive
    CREATION DATE/TIME: APR 17, 1998@11:41:06
    ACTIVATION DATE/TIME: APR 17, 1998@11:41:06
    MODIFICATION DATE/TIME: APR 17, 1998@11:41:06
```

4.4. LINK^HLUTIL3

Use this entry point to retrieve the logical link(s) associated with an Institution, VISN or Domain. Typically, you would use this entry point when setting up the HLL("LINKS") array to dynamically route a message.

Usage LINK^HLUTIL3(inst,.out_array, flag)

Input
inst: (required) Institution, VISN or Domain (pass either Name or IEN).
out_array: (required) Output array. Pass by reference.
flag: (optional) Flag indicating reference type passed in HLINST parameter:

null	inst is Institution Name or Number, or VISN Name
I	inst is Institution or VISN IEN
D	inst is Domain (name or IEN)

Output out_array: If one or more logical links are associated with the institution, VISN or domain, the IEN and name of each logical link are returned in the output array. The format of each entry returned in the output array is:

ARRAY(LINK_IEN)=LINK NAME

Description This entry point returns any logical link(s) associated with the Institution, VISN or domain you specify. The association between a logical link and an institution or domain is determined by setting the logical link's INSTITUTION and DOMAIN fields (introduced by patch HL7*1.6*14).

Institutions and VISNs are entries in the INSTITUTION file (#4). The ability to group other institutions within a VISN entry and return that information is introduced by Kernel patch XU*8*43. For VISNs, an entry can be set up in the INSTITUTION file that groups together the single institutions comprising the VISN.

Domains are entries in the DOMAIN file (#4.2).

Examples

```
>D LINK^HLUTIL3("BAY PINES, FL",.ZZLL) ZW ZZLL
ZZLL=
ZZLL(14)=VA516

>D LINK^HLUTIL3(516,.ZZLL,"I") ZW ZZLL
ZZLL=
ZZLL(14)=VA516
```