ADVANCED

Your in-depth guide for implementing HL7 solutions with BizTalk

HL7 for BizTalk

Howard Edidin
and Vikas Bhardwaj

Apress®

Contents at a Glance

CHAPTER 1

BizTalk and HL7

Although most of this book is primarily about the BizTalk Accelerator, you first need to understand what the HL7 is and what the HL7 standards are all about. This chapter will serve as an introduction to these standards and will also provide you with a foundation that will make it easier for you to understand the topics in all the chapters.

Before you take a look at the HL7 standards, you should know a little bit about HL7 (Figure 1-1). Since HL7 can do a better job describing itself then we can, the following sections contain excerpts from the HL7 web site (`www.hl7.org/about/index.cfm?ref=nav`).

Figure 1-1. *HL7 International logo*

What Is HL7?

Health Level Seven International (HL7) is a not-for-profit, ANSI-accredited, standards-developing organization. HL7 is dedicated to providing a comprehensive framework and related standards for the exchange, integration, sharing, and retrieval of electronic health information. HL7 supports clinical practice and the management, delivery, and evaluation of health services.

What Does HL7 Mean?

> *"**Level Seven**" refers to the seventh level of the International Organization for Standardization (ISO) seven-layer communications model for Open Systems Interconnection (OSI) - the application level.*

From `www.hl7.org/about/index.cfm?ref=nav`

What Is the Open Systems Seven-Layer Communications Model?

The HL7 organization describes this model as follows:

> *The application level interfaces directly to and performs common application services for the application processes. Although other protocols have largely superseded it, the OSI model remains valuable as a place to begin the study of network architecture.*
>
> From `www.hl7.org/about/index.cfm?mode=2`

Introduction to HL7 Standards

HL7 and its members have worked over the years to provide world-wide healthcare with a framework that includes related standards. The standards are for the exchange, integration, sharing, and retrieval of electronic health information.

- The HL7 standards define how this information is packaged and communicated from one party to another.
- The HL7 standards also set the language, structure, and data types that are required to provide a seamless integration between systems.
- HL7 standards provide support for the following:
    - Clinical practice of health services
    - Management of health services
    - Delivery of health services
    - Evaluation of health services
- The HL7 standards are recognized as the most commonly used in the world.

Seven Referenced Categories

HL7 standards are grouped into seven reference categories or sections. The HL7 organization describes these as follows:

- *Section 1: Primary Standards*: The primary standards are the most popular standards integral for system integration and interoperability.
- *Section 2: Foundational Standards*: The foundational standards define the fundamental tools and building blocks used to build the standards, and the technology infrastructure that implementers of HL7 standards must manage.
- *Section 3: Clinical and Administrative Domains*: Messaging and document standards for clinical specialties and groups are found in this section.
- *Section 4: EHR Profiles*: These standards provide functional models and profiles that enable the constructs for management of electronic health records.

- *Section 5: Implementation Guides*: This section is for implementation guides and/or support documents created for use in conjunction with an existing standard.
- *Section 6: Rules and References*: These are technical specifications, programming structures, and guidelines for software and standards development.
- *Section 7: Education & Awareness*: You can find HL7's Draft Standards for Trial Use (DSTUs) and current projects here, as well as helpful resources and tools to further supplement understanding and adoption of HL7 standards.

Source: `www.hl7.org/implement/standards/index.cfm?ref=nav`

We could describe these categories in more detail, but that is not what this book is all about. Besides, the HL7 organization does a much better job at it than we ever could. If you want to read more about these standards, you can access the information directly at `www.hl7.org/index.cfm?ref=nav`.

The HL7 Versions

Although there are several versions published by the HL7 organization, we will be concentrating on *HL7 Version 2 Product Suite of the HL7 Primary Standards.* The HL7 Version 2 Product Suite is part of Section 1 of the HL7 standards. The reason for the name ***primary standards*** is due to its popularity, since it is the most frequently used standard today.

■ **Note** Throughout this book I will be referring to HL7 Version 2 as HL7 Version 2.x. The "x" stands for the release version, such as Version 2.5. In addition, there are subreleases, such as Version 2.5.1. At the time of writing this book, Version 2.6 is the latest version supported by the BizTalk Server 2013 R2.

Many in the global healthcare industry say that "HL7's Version 2.x (V2) messaging standard is the workhorse of electronic data exchange in the clinical domain," as stated in the HL7 – Standards – Master Grid documentation. In addition, according to the HL7 – Standards – Master Grid documentation, several healthcare publications have described HL7 Version 2.x as the "most widely implemented standard for healthcare in the world."

It is designed to support the following:

- A central patient care system
- A distributed environment where data resides in departmental systems
- A distributed environment where data resides in multiple repositories

---

■ **Note** The HL7 Version 2.X product suite targets both healthcare IT vendors and providers.

Key Benefits of Version 2.x

The key benefits provided by Version 2.x are the following:

- Supports the majority of common interfaces used in the global healthcare industry.
- Provides a framework for negotiating what is not in the standard.
- Greatly reduces implementation costs.

HL7 Version 2.x Message Structure

In the last two sections you were introduced to the HL7 organization and its standards and versions. Moving forward, let's take a high-level look at the HL7 Version 2.x message structure.

To help you understand the message structure, you need to take a closer look at the components that make up the message structure:

- Delimiters
- Segments
- Fields
- Data Types
- Escape Sequences

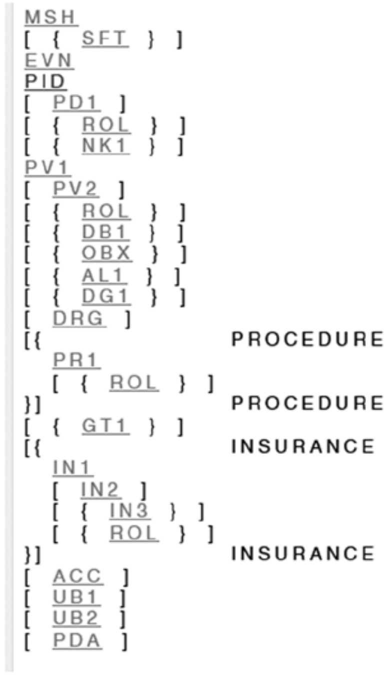
■ **Note** We refer to the parts of the message structure as components.

One of these key components is the segment. Let's take a brief look at the segment.

Segment

A segment is a logical grouping of data fields that represents a collection of related and unique information. A HL7 Version 2.x message can contain multiple segments. Segments can also contain child segments, commonly referred to as subsegments. These too can contain child segments.

Let's take a look at a common message structure. Figure 1-2 shows the message structure for the *Admit/Visit Notification* message type. This is a very commonly used message. You will be learning more about this and other message types in Chapter 2.



```
MSH
[ { SFT } ]
EVN
PID
[ PD1 ]
[ { ROL } ]
[ { NK1 } ]
PV1
[ PV2 ]
[ { ROL } ]
[ { DB1 } ]
[ { OBX } ]
[ { AL1 } ]
[ { DG1 } ]
[ DRG ]
[{                    PROCEDURE
  PR1
  [ { ROL } ]
}]                    PROCEDURE
[ { GT1 } ]
[{                    INSURANCE
  IN1
  [ IN2 ]
  [ { IN3 } ]
  [ { ROL } ]
}]                    INSURANCE
[ ACC ]
[ UB1 ]
[ UB2 ]
[ PDA ]
```

Figure 1-2. *Admit/Visit Notification message structure*

At the top of the message structure is a MSH segment. This is the message header segment and is required in all HL7 messages. Your knowledge of what is contained within this segment will make it easier to understand the technical terminology used throughout this book. Going forward, we will refer to this segment using its three letter identifier, MSH.

Contained within each segment are predefined data types. Let's take a look at the data types for the MSH segment.

The MSH Segment Data Fields

There are 21 data fields in the MSH segment. These are identified by a sequence number. Table 1-1 will make it easier for you to understand what each sequence number represents. The values in the R/O/C column are R = Required, O = Optional, and C = Conditional.

Table 1-1. *MSH Segment Data Fields (The source of the information contained within the table comes from the HL7 Version 2.x Standards Implementation Guide.)*

Seq	Description	Length	R/O/C
1	Field Separator	1	R
2	Encoding Characters	4	R
3	Sending Application	227	O
4	Sending Facility	227	O
5	Receiving Application	227	O
6	Receiving Facility	227	O
7	Date/Time of Message	26	R
8	Security	40	O
9	Message Type	15	R
10	Message Control ID	20	R
11	Processing ID	3	R
12	Version ID	60	R
13	Sequence Number	15	O
14	Continuation Pointer	180	O
15	Accept Acknowledgment Type	2	O
16	Application Acknowledgment Type	2	O
17	Country Code	3	O
18	Character Set	16	O
19	Principal Language of Message	250	O
20	Alternate Character Set Handling Scheme	20	O
21	Message Profile Identifier	427	O

As you can see, the descriptions are fairly easy to understand. But there is more to the data fields. The HL7 Version 2.x standard contains many different data types. There are both simple and complex types. You will learn more about the data types in Chapter 2.

■ **Tip** Chapter 5 contains a scenario mapping Version 2.x to the HL7 Version 3 CDA. This scenario describes the base data types used in all segments. In addition, Appendix III provides more information on the data types.

Moving Forward

Now that you have learned a little bit about the HL7 Version 2.x message structure, let's see what you will learn about in the rest of the chapters.

As previously mentioned, in Chapter 2, you will learn all there is to know about HL7 Version 2.x message encoding. You will also learn about the rest of the components that make up the HL7 Version 2.x message structure.

■ **Tip** I recommend that you read Chapters 2 through 4 before reading Chapter 5. The content contained within these chapters will make it easier to understand the advanced topics contained within Chapter 5.

In Chapter 3, you will learn all about the accelerator's capabilities.

In Chapter 4, you will view a few scenarios that will show you how to use the accelerator.

As previously mentioned, in Chapter 5, you will be presented with three scenarios. One of these scenarios was taken from a recently implemented solution.

In Chapter 6, you will get a preview of the new HL7 standard that the HL7 organization is currently working on.

Chapter 7 will provide you with best practices.

Summary

This chapter provided you with an introduction to HL7. You were also introduced to the HL7 Version 2.x standard. You had a quick look at the HL7 Version 2.x message structure and learned a little about the MSH segment. And finally, you had a glimpse of what the rest of the chapters are all about.

CHAPTER 2

HL7 Message Encoding

In the first chapter you were introduced to the HL7 standards. You were also introduced to the following:

- HL7 Version 2.x Message Structure
- Segments
- The MSH Segment

In this chapter, you will be concentrating on the message encoding for HL7 Version 2.x. Let's start off by looking at the HL7 V2.x message and the encoding types.

Tip If you are already experienced with HL7 and are knowledgeable about HL7 message encoding, you can skip ahead to the next chapter.

The HL7 Message

An HL7 message is used to transfer patient information from one system to another healthcare system. There are various types of HL7 messages defined to carry different types of patient information; for example, the ADT message type is used for patient's patient administration information. The HL7 message structure defined by HL7 organization is also referred to as HL7 message encoding, which we will talk about in this chapter.

Message Encoding Types

Message encoding types describe how an HL7 message is formatted to carry information from one system to another. There are two different types of encoding for the HL7 2.x version:

- *Delimiter-based Encoding*
- *XML Encoding*

Delimiter-based Encoding

Delimiter-based encoding, as the name suggests, defines data fields of variable lengths and separates them using a delimiter. In this encoding, five different delimiters are used to lay out the message structure; for example, a carriage return delimiter is used to group message into different segments, a pipe (|) is used to group each segment into fields, and so on. Listing 2-1 shows an HL7 message with delimiter-based encoding.

Listing 2-1. HL7 Message with Delimiter-based Encoding

```
MSH|^~\&|ADM|HUN|||201302260415||ADT^A01|125|P|2.2|||AL|NE
EVN||201302260250|||REG.RN^Name^Name^^^^|201302260250
PID|1||M123123123^^^^MR^HUN||Name^Name^^^^^L||19891203|F||W|126 LK^^HU
NK1|1|B^K^^^^|MO|2 Street^^City^State^Zip^Country^^^DES|Number CELL
PV1|1|P|F.ECC^^|||||YTU.FR^Y^F^R^^^MD|EDS|||||||ER||MCD|||||||||||||||
ZBC|1
ZCS|N|N^^UNKNOWN^UN^00000|N|||01371
```

You will learn more about the message structure and its details in the next sections.

Message Structure

The HL7 delimiter-based encoded message structure contains all the information within different components. There are five components that make up the message structure.

- Delimiters
- Segments
- Fields
- Data Types
- Escape Sequences

Delimiters

Delimiters are the key components of the HL7 message. *Delimiters* allow you to identify these key components within the HL7 message. Table 2-1 provides descriptions of the default delimiters.

Table 2-1. *The Default Delimiters of an HL7 2.X Message*

Delimiter	Description
`0D 0A hex (\r\n)`	Group message into segments
`|`	Fields: Group data into different fields within segments
`^`	Subfields: Group field data into subfields
`&`	Sub-subfields: Group subfields into sub-subfields
`~`	Indicates if a field is a repeatable field

These delimiters are found at the start of MSH segment, as shown in Listing 2-2.

Listing 2-2. Delimters in the MSH Segement

```
MSH|^~\&|ADM|HUN|||201302260415||ADT^A01|125|P|2.2|||AL|NE
```

These delimiters can be customized. For example, if you want # to be a field delimiter, then the code in Listing 2-2 will look like the code in Listing 2-3.

Listing 2-3. Customized Field Delimter

```
MSH#^~\&#ADM#HUN###201302260415##ADT^A01#125#P#2.2###AL#NE
```

■ **Note** In Listing 2-3, the \ character (backslash) before the & character is used as *escape character* to escape special characters in the message.

Segments

In Chapter 1, you were introduced to the key segments (components) of an HL7 V.2.x message. As previously mentioned, the HL7 message contains different segments; each segment represents specific information such as patient information, patient visit, patient insurance, lab results, etc., and groups the information into different data fields.

- Each segment is separated using carriage return (`\r` on UNIX-based systems and `\r\n` on Windows) and has a uniquely identified segment name, which is three characters in length. These identifiers can be found at the beginning of the segment.
- As per the HL7 messaging standard defined by HL7, a segment in a HL7 message may be required or optional, and also may or may not repeat depending on the message type.

There are hundreds of predefined segments as part of HL7 2.X specification. Each segment indicates the kind of information it contains; for example, a patient admit HL7 message may contain segments as shown in Table 2-2.

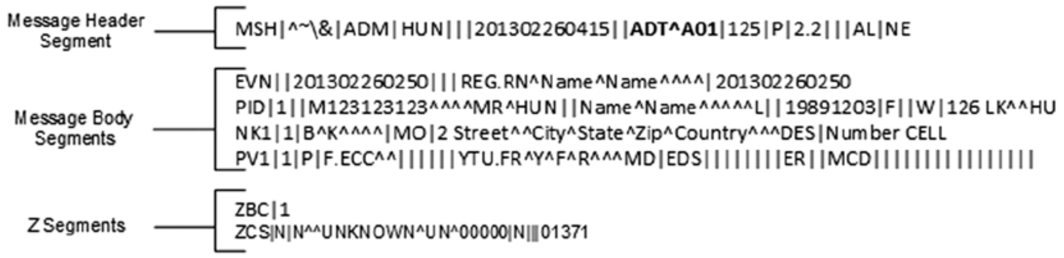
Table 2-2. *Sample List of Segments*

Segment	Name	Description
MSH	Message Header Segment	Contains metadata about the message
PID	Patient Identification	Patient identification information such as name, patient id, etc.
PV1	Patient Visit	Patient visit information
IN1	Insurance	Patient insurance information

■ **Tip** A complete list of these segments and their definitions can be found as part of HL7 Standard documentation on the HL7 web site at `www.hl7.org/implement/standards/product_brief.cfm?product_id=185`. Download a specific version of HL7 2.x and refer to the Appendix.

An HL7 2.X message is all about one or more segments. These segments divide the HL7 message into three different parts, as shown in Figure 2-1:

- One message header segment (MSH)
- One or more message body segments
- Z segments



```
MSH|^~\&|ADM|HUN|||201302260415||ADT^A01|125|P|2.2|||AL|NE

EVN||201302260250|||REG.RN^Name^Name^^^^|201302260250
PID|1||M123123123^^^^MR^HUN||Name^Name^^^^^L||19891203|F||W|126 LK^^HU
NK1|1|B^K^^^^|MO|2 Street^^City^State^Zip^Country^^^DES|Number CELL
PV1|1|P|F.ECC^^||||||YTU.FR^Y^F^R^^^MD|EDS||||||||ER||MCD||||||||||||||||

ZBC|1
ZCS|N|N^^UNKNOWN^UN^00000|N|||01371
```

Figure 2-1. *HL7 message parts*

Let's see what these segments contain.

1. *Message Header Segment (MSH)*: As the name suggests, this contains information about the message, such as the message type, message event, version etc. It provides metadata about the message. Later in this chapter we will discuss the MSH segment in more detail.
2. *Message Body Segments*: Body segments are one or more segments containing the message data; for example, for a patient administration (ADT) the message contains various segments like PID, PV1, NK1, etc. The body of message may contain any segment as specified in HL7 specification for a particular message type and message triggering event.
3. *Z Segments*: Z segments are optional segments in a message. These segments are defined for customized solutions based on need. Basically Z segments provide customization to HL7 messages based on particular requirements. The HL7 organization has not or will not define any segment with the letter Z.

Fields

Fields in an HL7 message contain message information; in the OOPS world, we would consider them as properties of a class where the class is the segment. In this section, you will learn about following:

- Field Identification
- Subfields
- Sub-subfields
- Repeatable Fields

Field Identification

As you learned in the delimiter section, fields in an HL7 segment are identified using a delimiter, which is a pipe (|) by default. Listing 2-4 shows how the delimiter is used.

Listing 2-4. Field Delimiter Sample

```
PID|1|123456|M123123123^^^^MR^HUN||Name^Name^^^^^L||19891203|F||W|126
```

Each field in the segment is referred to by its ordinal position in the segment; in Listing 2-3, the highlighted field is referred to as PID1, PID2, and PID3, and the values are 1, 123456, and M123123123^^^^MR^HUN, respectively.

■ **TIP** There is one exception to this. In the MSH, the field is still referred to by its ordinal position. However, the ordinal position value is one more than the ordinal position of the other segments. In the following example, the first field is referred to as MSH2 rather than MSH1. So where is MSH1? MSH1 is always a field separator, which in this case is a pipe (|).

```
MSH|^~\&|ADM|HUN|||201302260415||ADT^A01|125|P|2.2|||AL|NE
```

In a HL7 message, if you want to make sure that the receiving application deletes the data in their data store corresponding to the field, then the message should have the field value as “” e.g. |“”|. If the field is sent like ||, it will indicate that no change is required to the data store.

Subfields (Components)

Subfields are referred as *components* in the HL7 message specification. Throughout this book we will refer to them as *subfields* to make it easier to understand the relationship between fields and subfields.

A field can have subfields. You can find subfields within fields using the subfield delimiter, which is a caret (^) by default. In the following example, PID3 has six subfields highlighted; these fields are referred as PID3.1, PID3.2, and so on.

```
PID|1|123456|M123123123^^^^MR^HUN||Name^Name^^^^^L||19891203|F||W|126
```

Sub-subfields (Subcomponent)

A subfield can also have subfields; they are referred as *subcomponents* in the HL7 specification. We will refer to them as *sub-sub*fields.

Sub-subfields are separated by ampersand (&) within the subfield. In following example, SFT segment field 1, subfield 6th has three sub-subfields referred to as SFT1.6.1, SFT1.6.2, and SFT1.6.3.

```
SFT|MEDITECH, Inc. ^L^^^^MEDITECH&1.3.6.1.4.1.24310&ISO^XX^^^MEDITECH|5.6.6
```

Repeatable Fields

The repeatable field is like an array of a field and is identified by repeat delimiter (~) in the field. In following example, there are three PID3 fields:

```
PID|1||0266432^^^Test&Test^MR^.~151-22-5907^^^^SS^.~104475^^^^PI^.||
```

Each PID3 will have its defined subfields and so on.

■ **Tip** Repeatable fields are not applicable for subfields or sub-subfields. They are only defined at field level. Defining them at subfield or sub-subfield level will break the message structure.

Data Types

All fields (PID1, PID2, etc.) have data types defined as part of HL7 2.x standard. The data type definition includes following key information:

1. *Position* (SEQ): It defines the position of the field in the segment. This number is also used to refer to the field, such as PID1.
2. *Length (LEN)*: *It indicates the number of characters the data field may contain.*
3. *Data Type (DT)*: Similar to any programming language variable data types, data types represent the type of data field. Data types may be either primitive such as "text" or a complex type mde of of primitive types.
4. *Optionality*: A field may be required, or optional, or it may be required in a particular condition in a segment. Conditionally required fields are either based on a triggered event or on some other field.
5. *Repetition*: For each field, repetition is defined to indicate whether the field can repeat or not. In the HL7 specification, a field man not repeat, may repeat indefinitely, or may repeat up to a specified number depending on the definition.
6. *Optionality*: A field may be required, or optional, or may be required in a particular condition in a segment. Conditionally required fields are either based on a triggered event or on some other field.
7. *Name*: Each field has a purpose, and its name in the segment is defined to indicate that purpose (such as PID3).
8. *ID Number*: *It uniquely identifies* the data item throughout the standard.

■ **Tip** Appendix 3 contains the definitions of the data types currently supported in BizTalk.

■ **Note** The ID Number in HL7 V2.1 is a placeholder for undefined data types.

Use of Escape Sequences

The *escape sequence* allows you to have special characters in the HL7 message text that are not allowed normally; for example, delimiter characters can be included in the message field by using an escape sequence. An escape sequence as defined by HL7 organization (`www.hl7.org/implement/standards/index.cfm?ref=nav`) consists of an *escape character* which is defined in MSH.2, normally '\', followed by an escape code ID of one character then zero or more data characters, and another occurrence of the escape character, according to the spec. Table 2-3 shows the escape sequences that are defined for delimiters and other characters.

Table 2-3. *The HL7 Specification Provides the Escape Sequence for These Characters (Source: `www.hl7.org/implement/standards/index.cfm?ref=nav`)*

Escape Sequence	Description/Characters to Escape
\F\	Pipe "|"
\S\	Caret "^"
\T\	Ampersand "&"
\R\	Tilda "~"
\E\	Escape character "\"
\H\	Highlighting text start
\N\	Normal text (end highlighting)
\Xdddd. . .\	Hexadecimal data
\Zdddd. . .\	Locally defined escape sequence

Some examples of escape sequences are shown in the following sections.

Highlight Text

The escape sequence of \H\ can be used to format the text in many ways; for example, a receiving application can display the highlighted text in bold or in red color. The message fragment `"OBX| TOTAL \H\240*\N\   [90 - 200]"` can be used by receiving application to display the text as `"TOTAL 240* [90 - 200]" or` show the 240* in red.

Delimiter Escape Sequences

The HL7 messaging standard delimiters can be included in the message text by using the escape sequence of "\F\, \S\, \R\, \T\, and \E\". For example, the message fragment

```
OBX|25|TX|||     \F\A\F\. PROCEDURE - WIDE RESECTION.||||||F|
OBX|26|TX|||     B. TUMOR \T\ SITE - LEFT LOWER BACK.||||||F|
```

will be displayed as

```
|A|. PROCEDURE - WIDE RESECTION
B. TUMOR & SITE - LEFT LOWER BACK.
```

Usage and Examples of Formatted Text

Apart from the normal escape sequence, there are different commands that can be used alone or together with an escape sequence to format the text especially for fields with the FT data type. Formatted text commands start with a dot '.' Table 2-4 shows different formatting commands available.

Table 2-4. *Commands for Formatted Text and Description*

Command	Description
.sp <n>	Indicates the end of the current line and skips n number of spaces, where n is the number of spaces to skip.
.br	Indicates the beginning of a new line.
.fi	Indicates the beginning of word wrap or fill mode.
.nf	Indicates the beginning of no-wrap mode.
.in <n>	Indicates that the text should be indented by n number of spaces.
.ti <n>	Indents temporarily n number of spaces.
.sk <n>	Skips <n> spaces to the right.
.ce	Indicates the end of the current line and the center of the next line.

Message Types

Message types (MSH9.1) in the HL7 2.x specification correspond to a group of real world events. For example, Patient Administration has number of real world events, such as patient admit, patient registration, and many more. All such patient administration messages are grouped into one message type, which is referred to as ADT. There are more than 100 message types defined within HL7 specification. Some of the most common message types are listed in Table 2-5.

Table 2-5. *Most Common Message Types*

Message Type	Description
ADT	Admit Discharge Transfer; used for patient administration
ORM	Order Message; Order can be defined as a request of service, such as a lab service request
ORU	Observation Results, such as clinical lab results or imaging study results
DFT	Detail Financial Transaction; used for patient account purposes
MDM	Medical Document Management, helps manage medical records

■ **TIP** A complete list of these message types and their definitions can be found as part of the HL7 standard's documentation at on ay `www.hl7.org/implement/standards/product_brief.cfm?product_id=185`. Download a specific version of HL72.x and refer to Appendix A.

Message Trigger Event

Each message contains one or more trigger events (MSH9.2), which correspond to real-world events in the healthcare system. Some of the real world examples are the following.

1. Patient is admitted.
2. A request of service is created for lab.
3. Patient is discharged.
4. Patient appointment notification.

These real world events initiated the need for the HL7 2.x messaging standard to transfer information from one system to another. These events are bundled into specific message types; for example, Patient Administration-related events are bundled into message type ADT.

```
MSH|^~\&|ADM|HUN|||201302260415||ADT^A01|125|P|2.2|||AL|NE
```

The message type and trigger event has a one-to-many relationship, so one message type can have more than one trigger event; however, one trigger event can only be part of one message type. The combination of message type and trigger event governs the layout of the message, so which segments are required or optional all depends on this combination. Table 2-6 is a partial list of message types and their events.

Table 2-6. *Partial List of Message Types and Their Events*

Message Type^Event	Description
ADT^A01	Patient admit
ADT^A02	Transfer a patient
ADT^A04	Register a patient
ORM^O01	Order message
ORU^R01	Observation message
SIU^S12	Notification of new appointment booking

For each message type, a complete list of triggered events can be found in specific HL7 standard documents. If you need to find all events for ADT messages, then look for the CH03_PatientAdmin document of the HL7 message standard, which can be downloaded from: `www.hl7.org/implement/standards/product_brief.cfm?product_id=185`.

Message Version

A message also carries its HL7 2.x version to which it conforms within in message header MSH12 field. In the following sample, the version of the message is 2.4:

```
MSH|^~\&|ADT1|MCM|LABADT|MCM|198808181126|SECURITY|ADT^A01|MSG00001|P|2.4
```

Message Control ID

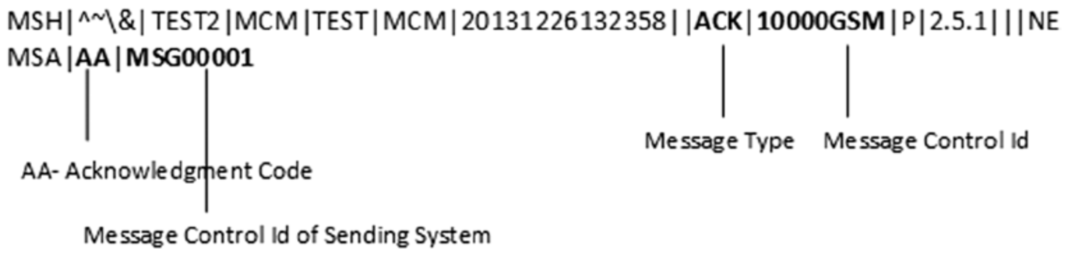
The message control id (MSH10) field in the message header represents an identity for the message, which the sending application should generate so that it is unique and can be used to correlate the message acknowledgment sent by receiving application. In an enterprise, the message control id field should be used to track messages. The following example shows how to create the message control id:

```
MSH|^~\&||COCFH|||201304210056||ADT^A08|FHGTADM.1.7230428|P|2.1
```

■ **Tip** The format in a message control id can be anything as long it is unique for each message.

Message Acknowledgment

In an HL7 message exchange between two systems, the receiving application sends an acknowledgment of the message received to the sending application. This way message delivery is guaranteed between two systems. Also, the sending application does not send the next message until the acknowledgment is received from the receiving application. Figure 2-2 shows an example of the acknowledgment message.



```
MSH|^~\&|TEST2|MCM|TEST|MCM|20131226132358||ACK|10000GSM|P|2.5.1|||NE
MSA|AA|MSG00001
```

Figure 2-2. *Message acknowledgment sample*

Key Elements of the Acknowledgement Message

There are few key elements of an acknowledgment message.

1. *Message Type (MSH9.1)*: The acknowledgement message type is ACK and it is same for all other message types.
2. *Acknowledgment Code*: MSA1 contains the acknowledgment code returned by the receiving application, which lets the sending application know the status of message, such as the receiving application has returned an error, rejected the message, or processed the message successfully. Using these acknowledgment codes, the sending application can decide the next step.
3. *Message Control ID (MSH10)*: Sent by the sending application, this is also returned in the acknowledgment message, which makes it easier to correlate.

Acknowledgement Modes

There are two modes that determine how an acknowledgment should be returned. They are the *original* and *enhanced* modes.

Original Mode

In original mode, an application acknowledgment is returned, which means that before returning an acknowledgment to sending application not only is the message received by the receiving application but also it is processed successfully.

■ **Tip** For original mode, both MSH15 (Accept Acknowledgment Type) and MSH16 (Application Acknowledgement Type) in the MSH segment need to be null or not present.

The message processing by the receiving application may vary depending on the functional requirements of a system; for example, a system can return the acknowledgment after storing the message in an input queue for later processing.

■ **Note** HL7 does not have any requirements for the design and architecture of a receiving application for processing of the message.

Enhanced Mode

The enhancement mode breaks the acknowledgment into two parts:

- *Accept Acknowledgement*: The receiving application sends the accept acknowledgment after storing the message to safe storage for later processing.
- *Application Acknowledgment*: Optionally, an application acknowledgment may also be returned to indicate the final message status to the sending system.

■ **Note** For enhanced mode, at least MSH15 (Accept Acknowledgment Type) and MSH16 (Application Acknowledgement Type) in the MSH segment need to be present.

Acknowledgment Types

There are four acknowledgment types that can be used by the sending application in enhanced mode to request an acknowledgment from the receiving application. Table 2-7 lists them.

Table 2-7. *List of Acknowledgement Types*

Code	Description
AL	Always send the acknowledgment.
ER	Send the acknowledgment only if there is any error or rejection.
NE	Never send an acknowledgment.
SU	Send acknowledgment only after successful completion.

These same acknowledgment type can be used for both MSH15 and MSH16 and accordingly the receiving application sends the acknowledgment back.

■ **Tip** MSH15 and MSH16 should only be used for enhanced mode acknowledgment and should not be used for original mode.

Acknowledgment Code

The Acknowledgment code (MSA1) in the acknowledgment message returns the acknowledgment code that indicates the message status returned by the receiving application. Table 2-8 lists the acknowledgment codes.

Table 2-8. *Acknowledgement Codes*

Code	Description
AA	Application Acknowledgment; message has been processed successfully.
AE	Application Error; there is some error with message content or validation errors.
AR	Application Reject; if the message cannot be processed due to some issue, usually such messages can be resent.
CA	Commit Accept; Used in enhanced mode when the message is committed to safe storage.
CE	Commit Error; validation error due to message content.
CR	Commit Reject; message rejected due to commit issue with receiving application.

ERR Segment

The error segment (ERR) is used to return error information in the acknowledgment message when the returned acknowledgment code is not AA or CA. One example of an acknowledgment with ERR segment is shown below:

```
MSH|^~\&|TEST2|MCM|TEST|MCM|20131226132358||ACK|10000GSM|P|2.5.1|||NE
MSA|AA|MSG00001
ERR||PID^1^5|102^Data type error^HL79999|E||||||||^^^^^^^^^^^^
ERR||NK1^1^2|102^Data type error^HL79999|E||||||||^^^^^^^^^^^^
```

As you can see, the ERR segment include details about the errors the receiving application sends in acknowledgment. Some of the key fields of ERR segment are defined in Table 2-9.

Table 2-9. *Key fields for ERR Segment*

Field	Name	Description
ERR2	Error Location	This field contains details about the location of error in HL7 message.
ERR 2.1	Segment ID	Segment Name where error occurred, such as PID.
ERR2.2	Segment Sequence	This indicate which segment has the error; this is useful when one segment is present multiple times in an HL7 message. If the segment is present only once, then it will be 1.
ERR2.3	Field Position	The field position where the error occurred.
ERR3	HL7 Error Code	Identifies HL7 error code for the error. The HL7 standard defines standard error codes for specific issues; for example, a Data Type Error relates to an issue where a field data type does not match. These standard error code can be found in Table 357 at `www.hl7.org`.
ERR 3.1	Identifier	Identifier of the error code; for example, 102 represents data type error.
ERR 3.2	Text	Error description.
ERR4	Severity	Severity of error: W - Warning I - Information E - Error F - Fatal Error

XML Encoding

Using XML is an alternate encoding for an HL7 message that can be used where both the sender and receiver can understand XML. The XML representation of HL7 Version 2.x messages is algorithmically derived directly from the HL7 database. This is done to prevent work has to be done by hand, which often is susceptible to errors. Furthermore, deriving the XML representation algorithmically allows generating schemas for future HL7 v2.x versions easily.

Message Structure

In XML encoding, message segment fields are represented as XML elements. Listing 2-2 shows an example of an HL7 acknowledgment message in both delimiter and XML encoding.

Listing 2-2. Sample Delimiter and XML Encoding

```
MSH|^~\&|TEST2|MCM|TEST|MCM|20131226132358||ACK|10000GSM|P|2.5.1|||NE
MSA|AA|MSG00001
ERR||PID^1^5|102^Data type error^HL79999|E||||||||^^^^^^^^^^^
ERR||NK1^1^2|102^Data type error^HL79999|E||||||||^^^^^^^^^^^

<?xml version="1.0" encoding="UTF-8"?>
<ACK>
  <MSH>
    <MSH.1>|</MSH.1><MSH.2>^~\&amp;</MSH.2><MSH.3><HD.1>TEST2</HD.1></MSH.3>
    <MSH.4><HD.1>MCM</HD.1></MSH.4><MSH.5><HD.1>TEST</HD.1></MSH.5>
    <MSH.6><HD.1>MCM</HD.1></MSH.6><MSH.7><TS.1>20131226132358</TS.1>
    </MSH.7><MSH.9><MSG.1>ACK</MSG.1></MSH.9><MSH.10>10000GSM</MSH.10>
    <MSH.11><PT.1>P</PT.1></MSH.11><MSH.12><VID.1>2.5.1</VID.1></MSH.12>
    <MSH.15>NE</MSH.15>
  </MSH>
  <MSA>
    <MSA.1>AE</MSA.1><MSA.2>MSG00001</MSA.2>
  </MSA>
  <ERR>
    <ERR.2><ERL.1>PID</ERL.1><ERL.2>1</ERL.2><ERL.3>5</ERL.3></ERR.2>
    <ERR.3><CWE.1>102</CWE.1><CWE.2>Data type error</CWE.2><CWE.3>HL79999</CWE.3>
    </ERR.3><ERR.4>E</ERR.4><ERR.12><XTN.12> </XTN.12></ERR.12>
  </ERR>
  <ERR>
    <ERR.2><ERL.1>NK1</ERL.1><ERL.2>1</ERL.2><ERL.3>2</ERL.3></ERR.2>
    <ERR.3><CWE.1>102</CWE.1><CWE.2>Data type error</CWE.2><CWE.3>HL79999</CWE.3>
    </ERR.3><ERR.4>E</ERR.4><ERR.12><XTN.12> </XTN.12></ERR.12>
  </ERR>
</ACK>
```

■ **Note** The above XML format has been changed to fit display the content properly.

Root Element—Message Structure IDs

In the HL7 Version 2.x standard, several trigger events share the same abstract message syntax. An example of this could be an ADT message that is triggered by event A04, which the same message structure as ADT A01. In addition to A04, other events like A05, A08, A13, A14, A28, and A31 share the same A01 message structure. Until Version 2.3.1, there was no way to represent this in an HL7 message; however, with version 2.3.1, this fact became standardized and represented as Message Structure ID in an HL7 message by adding a new field called **MSH9.3** to the message type **MSH9** field. The following is an example of the ADT_A01 delimited message structure used by ADT A04 message event:

```
MSH|^~\&|ADT2||LAB||198808181121|SECURITY|ADT^A04^ADT_A01|MSG00001|P|2.4
```

The message structure ID is relevant for XML encoding since this is used as a root element in an XML instance document, as shown:

```
<?xml version="1.0" encoding="UTF-8"?>
<ADT_A01>
        ...(Segment Elements)
</ADT_A01>
```

Segments

Message structures contain segments also represented as XML elements. Each segment name is three characters long, similar to HL7 V2.x encoding; these names are used as an XML element. The following example shows a few segments as an XML element:

```
<?xml version="1.0" encoding="UTF-8"?>
<ADT_A04>
        <MSH>
              ...(MSH Field Elements)
        </MSH>
        <PID>
             ...(PID Field Elements)
        </PID>
        <PV1>
             ...(PV1 Field Elements)
        </PV1>
        ...(Other Segment Elements)
</ADT_A04>
```

Segment Groups

One benefit of XML encoding is the ability to represent segment grouping easily. In a group, whether a segment is optional or repeatable is represented using brackets [...] or braces { ... } in the standard. So segments [{ IN1, [IN2], [{ IN3 }], [{ ROL }] }] represent the INSURANCE group, where each group itself is optional or may repeat, and within the group segment, IN1 is required, IN2 is optional, IN3 is optional or may repeat, and ROL is optional and may repeat as well.

The explicit grouping is one major difference between delimiter and XML encoding as there is no grouping in traditional delimiter encoding.

Segment group naming includes upper case names. So ADT_A01.INSURANCE and the name itself represents the purpose of the segment group. In other words, the "INSURANCE" group bundles all INx segments, and the "VISIT" group bundles PV1 and PV2 segments, as shown in following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<ADT_A04>
        <MSH>
        ...(MSH Field Elements)
        </MSH>
        <EVN>
        ...(EVN Field Elements)
        </EVN>
        <PID>
        ...(PID Field Elements)
        </PID>
        <ADT_A04.INSURANCE>
                <IN1>
                ...
                </IN1>
                <IN2>
                ...
                </IN2>
                <ROL>
                ...
                </ROL>
        </ADT_A04.INSURANCE>
</ADT_A04>
```

■ **Note** Segment group naming includes the message structure id as a prefix, as shown in the above example, to differentiate between different message structures, so the INSURANCE group can be found in ADT_A01 with different cardinality.

Another way of grouping segments is by a choice; here a decision has to be made as to which set of segments should be conveyed in a message: This is indicated in the HL7 standard documents by angle brackets: < and >. The different options for a choice are then separated by a vertical bar. In the XML schema, the choice is represented using the xsd: choice element.

■ **Note** This vertical bar is independent from the vertical bar in the conventional encoding reflecting the standard field delimiter.

Fields

In XML encoding, XML elements representing fields and their naming conventions are based on segment names and the field positions in the segment. So the first field in PID segment is named PID.1, the second is PID.2, etc., like so:

```
...
<PID>
  <PID.1>1</PID.1>
  <PID.2>
    ...
  </PID.2>
</PID>
```

In delimiter-based encoding, empty fields are also required to be separated by a pipe; however, in XML encoding, an element with no contents simply can be omitted (unless explicit use of the "" is required to force a data delete action by the receiving application).

Data Types

The data types for the fields are defined similarly to that of delimiter-based encoding for each field. Please refer to the "Data Types" section of the "Delimiter-based Encoding" section for more details.

Use of Escape Sequence

There are some basic differences in the escape sequence between delimiter- and XML-based encoding. These differences are as follows:

- In XML encoding, there is no use of a delimiter character escape sequence, as these are not used to form the message structure.
- Instead of using "\" as an escape sequence character, XML encoding uses an XML element to define escape sequence, as shown in Table 2-10.

Table 2-10. *Escape Characters in XML Encoding*

Escape Characters as Defined Earlier	Replacement for XML Encoding	Description
\H\	<escape V="H"/>	Start highlighting text.
\N\	<escape V="N"/>	Normal text (stop highlighting).

Similarly, the command also indicates using XML elements, as shown in Table 2-11.

Table 2-11. *Formatting Comments in V2.x and Their Replacements in V2.xml*

Escape Characters as Defined in the v2.x Standard	v2.xml Encoding
\.br\	<escape V=".br"/>
\.spn\	<escape V=".spn"/>
\.in±n\	<escape V=".in±n"/>
\.ti±n\	<escape V=".ti±n"/>

Message Header Segment

As mentioned before, with XML encoding there is no change to message semantics. All header properties and their definitions remain as in delimiter-based encoding.

Message Acknowledgment

Both message acknowledgment modes, original and enhanced processing, remain the same in XML encoding, except that validation of messages now includes well-formed and valid XML by the receiving system. This means sender system must ensure that the HL7 message's XML is well-formed and valid as per its schema.

XML Schemas

HL7 provides XML schemas for each version supported by HL7 Version 2.x.XML specification, as shown in Table 2-12.

Table 2-12. *XML Schema (Source: `www.hl7.org/implement/standards/index.cfm?ref=nav`)*

Schema File	Description
MessageStructureID.html	A set of many files in HTML format containing a short description of the message and links to the corresponding schemas.
<MessageStructureID>.xsd	A set of many schemas each containing the schema definitions for a specific message structure specified by MessageStructureID. For example, ADT_A01.xsd contains the definitions for the ADT A01 message structure, ADT_A02.xsd for ADT A02, and so forth.
Segments.xsd	Schema for all segment definitions, imports fields definitions.
fields.xsd	Schema for all field definitions, imports data type definitions.
datatypes.xsd	Schema for all data type definitions for v2.
batch.xsd	Schema containing definition of batch.
Messages.xsd	Schema containing all message definitions together.

An XML instance of a specific message should refer to the corresponding schema. The following code shows a schema reference within a v2.XML XML message instance fragment:

```
<ADT_A04
        xmlns="urn:hl7-org:v2xml"
        xmlns:xsi="http://www.w3.org/2001/XmlSchema-instance"
        xsi:schemaLocation="urn:hl7-org:v2xml ADT_A04.xsd">

        <MSH>...
<ADT_A04>
```

■ **Note** When you install the BizTalk 2013 R2 HL7 Accelerator (BTAHL7), the setup wizard creates the `<drive>:\ Program Files\Microsoft BizTalk 2013 R2 Accelerator for HL7\Templates\Schemas` folder structure that contains templates to support the events for HL7. Beneath this folder, a `V2.X` folder includes a subfolder for each of the HL7 2.X versions: 2.1, 2.2, 2.3, 2.3.1, 2.4, 2.5, 2.5.1, and 2.6.

Summary

In this chapter, you took an in-depth tour of the HL7 message structure, encoding, and acknowledgment model. *Understanding these concepts is key to any implementation of healthcare-based system.* In Chapter 3, you will learn about the Microsoft BizTalk HL7 Accelerator and its capabilities for HL7 2.x message processing.

CHAPTER 3

Understanding the HL7 Accelerator

Introduction

The BizTalk HL7 Accelerator, commonly referred to as the BTAHL7, is an add-on capability provided by Microsoft BizTalk Server to integrate diverse healthcare systems using HL7V2.x and HL7 V2.xml encoding standards. The add-on is available as part of BizTalk media installation and provides various add-on components like pipelines, schemas, adapter, utilities etc.

The BizTalk HL7 Accelerator provides all required components necessary to implement such scenarios and includes the following in-built functionalities:

- The MLLP Adapter to receive and transmit HL7 messages
- The ability to parse a HL7 V2.x-encoded message into XML or vice versa
- The ability to parse a HL7 V2.xml-encoded message
- Return acknowledgment to the sending application based on the mode
- Batch processing
- Validation of message
- Logging

In addition to using the HL7 Accelerator's built-in functionalities, you can use other BizTalk capabilities such as Business Activity Monitoring, BizTalk Mapper, Business Rule Engine, etc. to build HL7-based solutions.

In this chapter, you are going to learn about all these capabilities and the components provided by the HL7 Accelerator for the development and runtime environment, such as project templates, the HL7 Disassembler, the HL7 Configuration Explorer, etc.

Architecture

Figure 3-1 shows the architecture diagram, which includes all HL7 Accelerator components. The diagram divides components into two categories, the Development Environment and the BizTalk Runtime Environment, to help you understand the components required to develop, build, test, and run HL7-based applications.

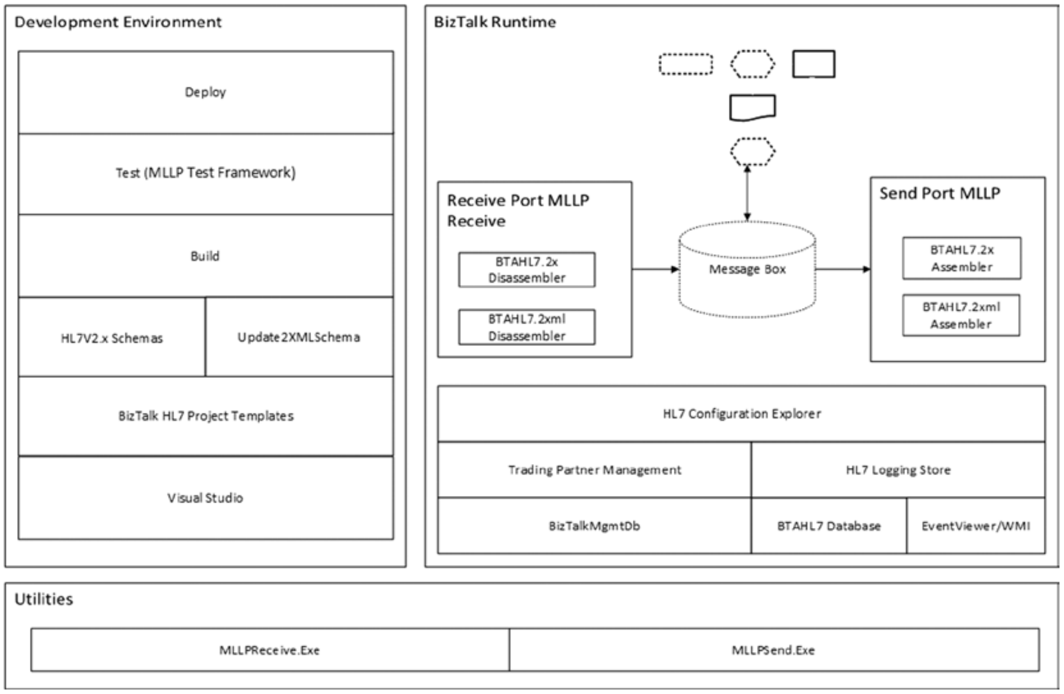


Figure 3-1. *HL7 Accelerator architecture*

Development Environment

Development environment includes visual studio artifacts such as schemas, project templates and other tools required for developing HL7 solutions with BizTalk. All these artifacts are explained below.

HL7 V2.x XML Schemas

As you are aware, many BizTalk capabilities are dependent on XML schemas, which everyone defines for a solution. The BizTalk HL7 Accelerator also needed a way to define schemas for the HL7 v2.x standard for working with HL7 messages and using BizTalk capabilities like transformation, orchestrations, etc. In order to define these schemas, the BizTalk team used a Microsoft Access database to autogenerate these schemas and shipped them with the HL7 Accelerator installation. This Access database is maintained by the HL7 organization and is available for a cost on the HL7 web site at `www.hl7.org/store/viewitem.cfm?Item=ACCESS`. This database includes all the details, including field definition, segment definition, data types, message triggering events, etc.

■ **Tip** The BizTalk 2013 R2 HL7 Accelerator provides schemas for version 2.1 to 2.6 and can be found at `%SystemDrive% \Program Files (x86)\Microsoft BizTalk 2013 R2 Accelerator for HL7\Templates\Schemas\V2.X` after the installation for 64-bit installations and in `%SystemDrive% \Program Files\Microsoft BizTalk 2013 R2 Accelerator for HL7\Templates\Schemas\V2.X` for 32-bit installations.

These schemas use a naming convention based on a format of `<Message Type>_<Trigger Event>_<HL7 V2.x version>_GLO_DEF`, so a patient admit triggering event in a healthcare system using HL7 messaging standard version 2.5.1 will map to a ADT_A01_251_GLO_DEF schema file name.

This essentially means that there is one schema defined for each triggering event of every message type of a specific version.There are three other schemas for each version of schemas, as shown below. These schemas are used by every message schemas using a XSD import and they define segment definition, element data types, and table values for HL7 messages based on the standard.

- datatypes_<version>
- segments_<version>
- tablevalues_<version>

In addition to HL7 V2.x message schemas and data-type schemas, the HL7 Accelerator also provides three other schemas. They are referred as message header and acknowledgment schemas.

- MSH_25_GLO_DEF
- ACK_26_GLO_DEF
- ACK_25_GLO_DEF
- ACK_24_GLO_DEF

These schemas are common to all HL7 V2.x versions of the schemas. MSH_25_GLO_DEF corresponds to the MSH segment of message and the other three ACK schemas correspond to the HL7 acknowledgement.

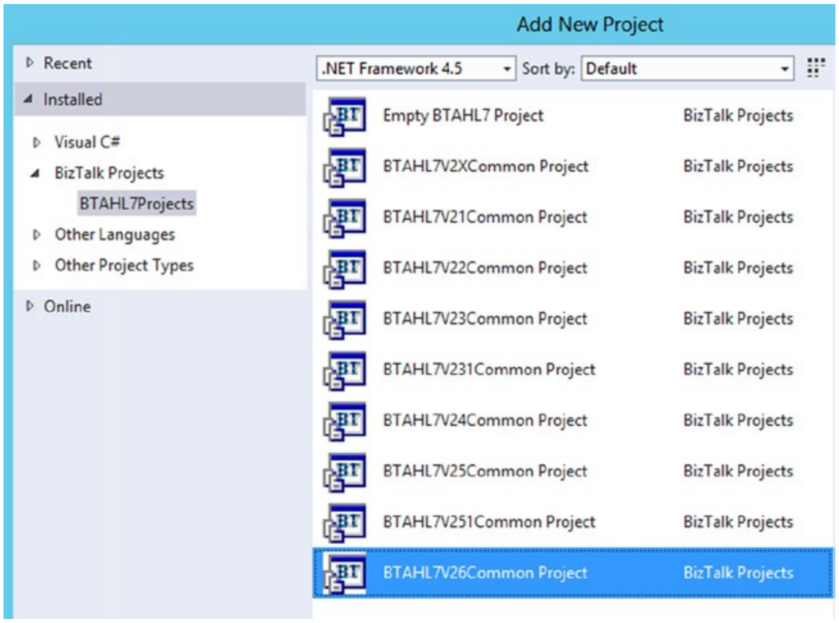
■ **Note** There can be only one message header and acknowledgment schema deployed in a BizTalk group.

Visual Studio Project Templates

The first step in the development environment to build any HL7-based solution is to add schemas to your BizTalk project depending on the HL7 version, message type, and triggering event. To help with this, the HL7 Accelerator provides Visual Studio project templates to create HL7V2.x schemas projects easily. Once the HL7 Accelerator is installed, the following project templates are available in Visual Studio under BizTalk Projects:

- *BTAHL7V2XCommon*: This project adds a message header and acknowledgment schemas to your project.
- *BTAHL7V21XCommon - BTAHL7V251XCommon*: There are eight of these project templates, one for HL7 versions 2.1 to 2.6. These projects add datatypes, segments, and tablevalue schemas for that version.

These two projects are added as first step in any HL7 solution development. Figure 3-2 shows the templates in Visual Studio.



Empty BTAHL7 Project	BizTalk Projects
BTAHL7V2XCommon Project	BizTalk Projects
BTAHL7V21Common Project	BizTalk Projects
BTAHL7V22Common Project	BizTalk Projects
BTAHL7V23Common Project	BizTalk Projects
BTAHL7V231Common Project	BizTalk Projects
BTAHL7V24Common Project	BizTalk Projects
BTAHL7V25Common Project	BizTalk Projects
BTAHL7V251Common Project	BizTalk Projects
BTAHL7V26Common Project	BizTalk Projects

Figure 3-2. *Visual Studio project templates*

■ **Note** In Chapter 7, you will learn more about the best practices for organizing schema projects during development.

If you don't see the HL7 project templates in Visual Studio after the installation of the HL7 Accelerator, most likely your machine is missing the SQL Server Data Tools and Client Tools SDK. Install these from the SQL Server installation and then reinstall the HL7 Accelerator to fix the issue.

Visual Studio Item Templates

The HL7 Accelerator also installs Visual Studio BizTalk project item templates, which allow you to add a message schema based on a message type, triggering event, and versioning to your schema project just like you create/add a new schema to any other BizTalk project. Once BizTalk projects using HL7 project templates are added, the specific message schema can be added to your project using the Ctrl+Shift+A option and selecting the BTAHL7Schema. Figure 3-3 shows the HL7 schema selector; this selector pops up as soon as you select BTAHL7Schema to add to BizTalk project.

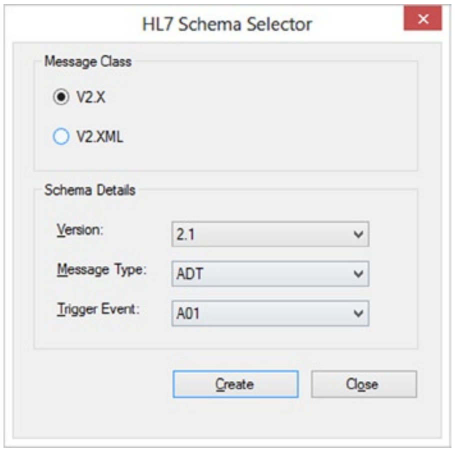


Figure 3-3. *Visual Studio schema selector item templates*

HL7 V2.xml XML Schemas

BizTalk does not provide any schemas for HL7 XML-encoded messages. These schemas are provided by the HL7 organization and can be used directly with the BizTalk HL7 Accelerator after a little modification. BizTalk provides a utility called Update2XmlSchema.Exe to modify the schemas. You can download these schemas from `www.hl7.org/implement/standards/product_brief.cfm?product_id=214` and then use the utility to modify these schemas.

This utility places the modified schemas in the `%SystemDrive% \Program Files (x86)\Microsoft BizTalk 2013 R2 Accelerator for HL7\Templates\Schemas\V2.XML` folder for 64-bit installations and in the `%SystemDrive%\Program Files\Microsoft BizTalk 2013 R2 Accelerator for HL7\Templates\Schemas\V2.XML` folder for 32-bit installations.

MLLP Test Framework

The HL7 Accelerator also provides a MLLP test framework library which can be used along with Visual Studio Unit Test Framework to test a MLLP-based solution. This library can be found in the `%SystemDrive%\Program Files (x86)\Microsoft BizTalk 2013 R2 Accelerator for HL7\SDK\MLLP Utilities` folder. You can send and receive HL7 messages within the test environment using this library. This will allow you to create unit tests for your solution during development.

Runtime Environment

Runtime environment includes visual studio artifacts such as pipelines and other tools required to configure HL7 solution at runtime within BizTalk. Each of these are explained below:

HL7 Configuration Explorer

The HL7 Configuration Explorer allows you to configure parties for your HL7 message exchange. A party is not required to be configured for a HL7 message exchange with BizTalk Server; however, party configuration provides many flexibilities for the implementation. So what is a party? A party is often referred to as a “trading partner” or a “trading party.” A trading partner is single business entity that can send or receive messages to or from any other partner. BizTalk has full capabilities to manage trading partners from BizTalk Management Console (Option Parties), which is prominently used for EDI message exchange; the HL7 Configuration Explorer is an enhancement on top of the trading partner management.

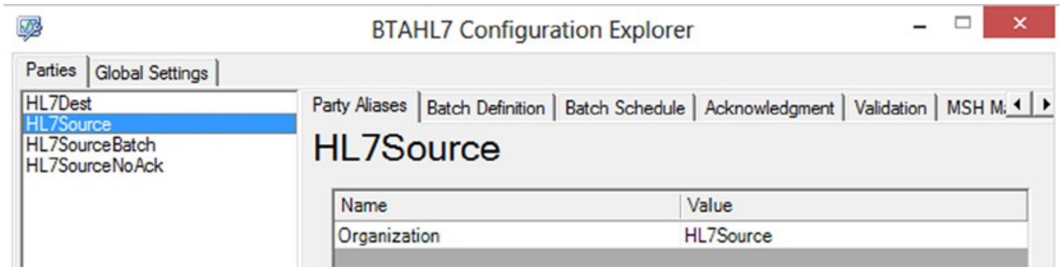
■ **Note** To learn more about trading partner management (TPM), please refer to `http://msdn.microsoft.com/en-us/library/bb259970.aspx`.

Creating a Party

From the Parties option on the BizTalk Management Console, choose Create New Party with a Name. There are no further details required, just the name.

Configuring a Party

After party is created, open the HL7 Configuration Explorer named "BTAHL7 Configuration Explorer" and there you should see the party created, as shown in Figure 3-4.



Name	Value
Organization	HL7Source

Figure 3-4. *HL7 Configuration Explorer*

The Configuration Explorer allows you to configure following:

- *Batch Definition*: If processing a batch HL7 message, on this tab you can configure details about the batch, such as whether to split the inbound batch message.
- *Batch Schedule*: For outbound batches, batch frequency, such as how many messages should create one batch, can be configured on this tab.
- *Acknowledgment*: On this tab, you can configure the acknowledgment modes you want to use for message exchange (original or enhanced).
- *Validation*: You can configure if you want to validate the message body while parsing the message or not, and some other properties.
- *MSH Map*: This tab allows you to override the MSH fields in the outgoing message, such as when sending a message to a destination, MSH3 should always have a static value.

You will learn more about each of these configuration as you move on to different components and where these configurations are relevant.

Configuring Global Settings

Global settings applies to HL7 logging service globally and is not party specific. You will learn more about HL7 logging service and its configuration options later in this chapter.

HL7 V2.x Receive and Send Pipelines

BizTalk provides custom pipelines to disassemble an HL7 V2.x messages into XML and to assemble the XML into HL7 V2.x message. These pipelines are deployed to the BizTalk Application 1 application after the HL7 Accelerator installation and can be used in BizTalk receive and send ports like any other pipelines. Receive and send pipelines use a custom HL7 Disassembler (Microsoft.Solutions.BTAHL7.HL72fDasm) and Assembler (Microsoft.Solutions.BTAHL7.HL72fAsm) pipeline components. These are the two key components used in any HL7 solution as they provide the ability to convert an HL7 flat file message into XML. Let's discuss each of these components in more detail.

HL7 Disassembler

The HL7 Disassembler pipeline component is one of the key components that implements the bulk of the HL7 standard's functionality. Figure 3-5 shows the diagram of the components used by HL7 Disassembler to disassemble an HL7 message into XML.

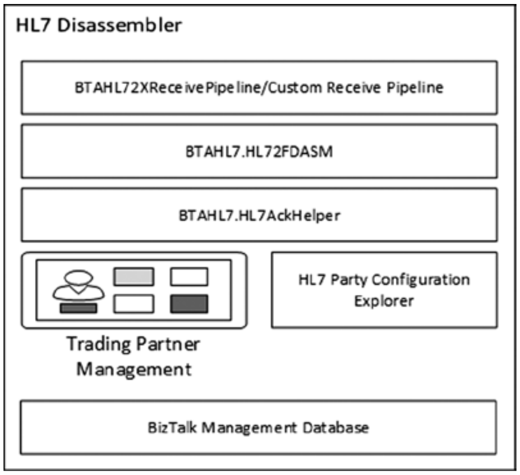

HL7 Disassembler
BTAHL72XReceivePipeline/Custom Receive Pipeline
BTAHL7.HL72FDASM
BTAHL7.HL7AckHelper
Trading Partner Management
HL7 Party Configuration Explorer
BizTalk Management Database

Figure 3-5. *HL7 Disassembler architecture components*

The HL7 Disassembler performs various task while disassembling the message, and each of these tasks are explained in the following sections.

Multipart Message

The HL7 Disassembler creates a multipart message with the following three parts on successful disassemble of the message:

1. *MSHSegment*: The HL7 Disassembler uses the MSH_25_GLO_DEF schema to disassemble the HL7 message MSH segment.
2. *BodySegments*: The HL7 Disassembler uses one of the message schemas based on the HL7 message type, message version, and trigger event to disassemble the body of the message.
3. *ZSegments*: Any Z segments in the message are disassembled to this part as string data type.

In order to consume this message in an orchestration, you need to create a multipart message with the above three parts.

Identifying a HL7 Party

The HL7 Disassembler identifies if there is a HL7 party configured for the incoming HL7 message. The HL7 party, sometimes referred to as the "trading partner," allows you to configure various options like acknowledgement, validation, batching, etc. You can find more details on party configuration in the "HL7 Configuration Explorer" section in this chapter.

The HL7 Disassembler uses a MSH3 value to find a corresponding HL7 party; so if the MSH3 value is "HL7Source", then HL7 disassembler will look for a party with the name of "HL7Source", and if a party is found, it will use the configuration as required. If not, then the HL7 Disassembler will try to disassemble the message with default configurations. If in a message MSH3.1, MSH3.2, and MSH3.3 all are populated, then the HL7 Disassembler concatenates all the three values and tries to find the matching party. You will see how party different configurations are used by the HL7 Disassembler in future sections.

Disassembling a Message Header

The HL7 Disassembler disassembles the MSH segment to the MSHSegment part of the message. The message schema MSH_25_GLO_DEF with default namespace `http://microsoft.com/HealthCare/HL7/2X` is used by the Disassembler. If this schema is not present in the BizTalk, the message is failed with a parsing error and the following error is logged in the event viewer:

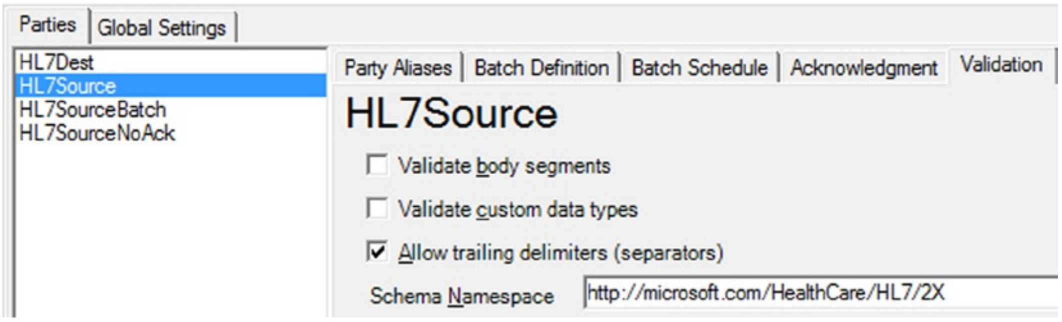
```
Alternate Error Description: Schema http://microsoft.com/HealthCare/HL7/2X#MSH_25_GLO_DEF not found
```

Disassembling Other HL7 Segments to Message Body Schema

The HL7 Disassembler first tries to identify the message body schema based on MSH9.1 (Message Code), MSH9.2 (Trigger Event), and MSH12.1 (Version); for example, the message schema for a HL7 message with the following message header will be ADT_A01_251_GLO_DEF. It concatenates all the values with underscore and suffixes it with GLO_DEF.

```
MSH|^~\&|HL7Source|MCM|HL7Dest||199601121005||ADT^A01|000007|P|2.5.1
```

Message schema identification also depends on schema namespace. The default namespace for all schemas is `http://microsoft.com/HealthCare/HL7/2X`, and at first the HL7 Disassembler uses the default schema namespace to find the schema in BizTalk. You can override the default schema namespace by configuring a TPM Party and overriding the namespace in the HL7 Configuration Explorer party validation tab, as shown in Figure 3-6.


Parties | Global Settings

HL7Dest
HL7Source
HL7SourceBatch
HL7SourceNoAck

Party Aliases | Batch Definition | Batch Schedule | Acknowledgment | Validation

HL7Source

☐ Validate body segments

☐ Validate custom data types

☑ Allow trailing delimiters (separators)

Schema Namespace: http://microsoft.com/HealthCare/HL7/2X

***Figure 3-6.** Party configuration validation tab*

If the HL7 Disassembler does not find the schema with matching namespace and schema name, it fails to parse the HL7 message and logs the error in Event Viewer with the following description:

```
Alternate Error Description: Schema http://microsoft.com/HealthCare/HL7/2X#ADT_A01_231_GLO_DEF not found
```

■ **Note** It's not necessary to create a HL7 party for the HL7 Disassembler to identify the message schema. It uses the default namespace and other configuration in case it does not find the party configured.

The message schema forms the message body (BodySegments) of the multipart message and contains the bulk of the HL7 message segments.

Disassembling Z Segments

Z segments are custom-defined segments and are not part of the message body schemas. Z segments are disassembled to the third part of a multi-part message as a string data type by the Disassembler. If there are no Z segments in the message, then this part remains empty. You can include Z segments as part of the message body by changing the default body schemas.

Optionally Validating the Message Body

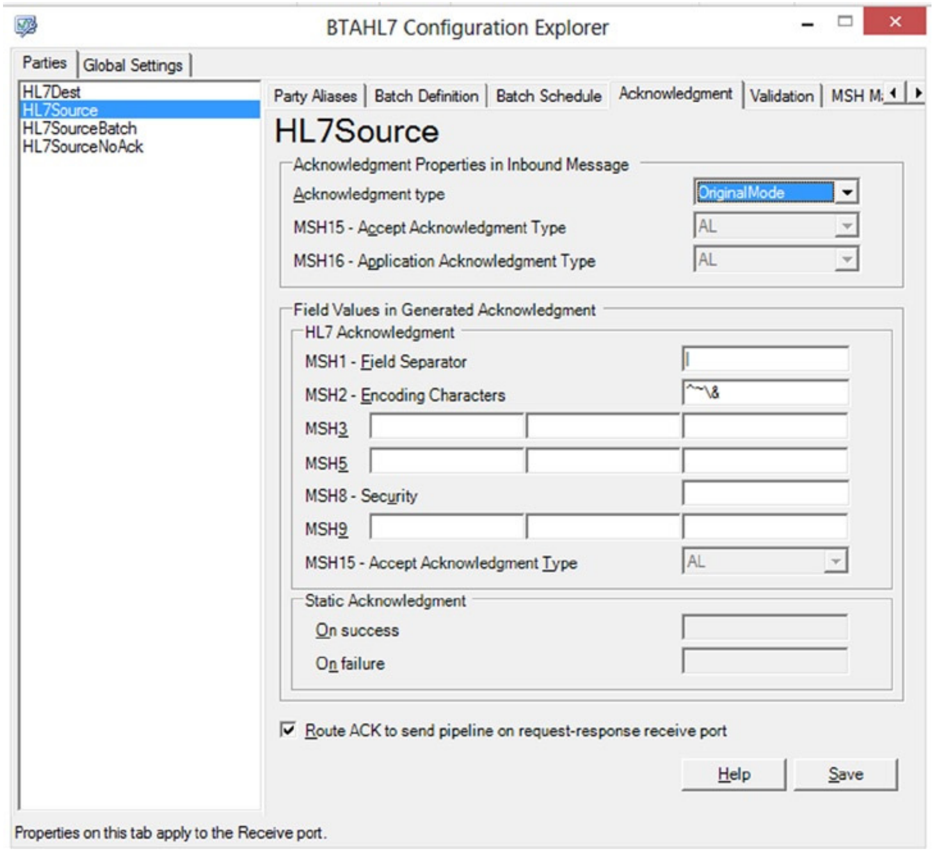
The HL7 Disassembler by default validates the message body segments data from the message schema definition. The validation include the data type and expected values for the field validations; for example, if the Marital Status field in the PID segment expects to have only set of values, and in the message the value is outside that range, then a validation error occurs. In case of a validation failure, the message fails to parse and an error is logged to the event viewer. The error details returned by the HL7 Disassembler follow the HL7 standard and return the error in the following format:

```
Error happened in body during parsing
Error # 1
Segment Id: PID
Sequence Number: 1
Field Number: 16
Error Number: 103
Error Description: Table value not found
Encoding System: HL70396
```

This error indicates that field PID16 (Marital Status) of first occurrence (sequence) of segment PID has an invalid value. There can be multiple errors in one HL7 message body; all of them are repeated in event viewer logs.

Generating the Acknowledgment

The HL7 Disassembler also generates the acknowledgment as per the HL7 messaging standard. The HL7 Disassembler implements both acknowledgment modes, original and enhanced. In addition to original and enhanced acknowledgment modes, the HL7 Disassembler also provides support for deferred and static acknowledgment modes. All these modes can be configured using the Acknowledgment tab of the HL7 Configuration Explorer, as shown in Figure 3-7.


BTAHL7 Configuration Explorer

Parties | Global Settings

HL7Dest
HL7Source
HL7SourceBatch
HL7SourceNoAck

Party Aliases | Batch Definition | Batch Schedule | Acknowledgment | Validation | MSH M:

HL7Source

Acknowledgment Properties in Inbound Message

Acknowledgment type	OriginalMode
MSH15 - Accept Acknowledgment Type	AL
MSH16 - Application Acknowledgment Type	AL

Field Values in Generated Acknowledgment

HL7 Acknowledgment

MSH1 - Field Separator	|
MSH2 - Encoding Characters	^~\&
MSH3	
MSH5	
MSH8 - Security	
MSH9	
MSH15 - Accept Acknowledgment Type	AL

Static Acknowledgment

On success	
On failure	

☑ Route ACK to send pipeline on request-response receive port

Help | Save

Properties on this tab apply to the Receive port.

Figure 3-7. *Party Explorer Acknowledgment tab*

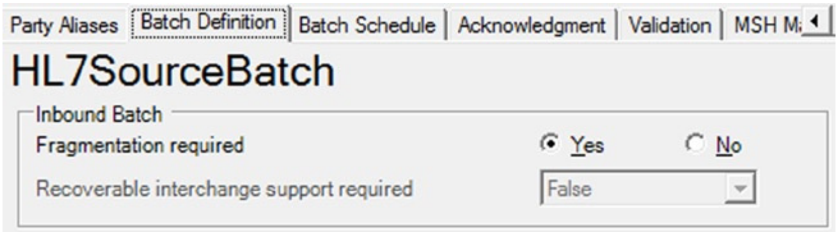
In Figure 3-7, the acknowledgment type drop-down allows you to select one of the acknowledgment modes and related properties to configure. These different modes are explained below.

- *None*: There is no mode defined; in this case, the Disassembler generates the acknowledgment only if requested by the sending application by sending an acknowledgment type in MSH15 or MSH16.
- *Original Mode*: In this mode, the Disassembler generates an application acknowledgment if MSH15 and MSH16 are empty. If the sending application populates MSH15 or MSH16, then it takes the precedence over party acknowledgment configuration. This mode also allows to you to configure the MSH header fields for generated acknowledgment. These fields are used when MSH15 and MSH16 are empty in the message.
- *Enhanced Mode*: In this mode, unlike with original mode, the Configuration Explorer allows you to modify MSH15 and MSH16 depending on the requirements and the Disassembler generates the acknowledgment accordingly. Again, MSH15 and MSH16 in a message take precedence if they're not provided in-message. This mode similar to original mode as it allows you to configure MSH header fields for a generated acknowledgment message, and if configured, unlike original mode, acknowledgment is always generated using configured values.
- *Deferred Mode*: This mode exists due to backward compatibility with the HL7V2.1 messaging standard delayed acknowledgment mode. The acknowledgment is generated the same way as original mode except that the generated acknowledgment also has a MSA5 field with a value of D indicating a delayed acknowledgment.
- *Static*: In this, a static acknowledgment can be configured in the Configuration Explorer for both success and failure cases. The HL7 Disassembler use the exact text to return an acknowledgment and ignores any value of MSH15 and MSH16 in the message header.

■ **Note** In all acknowledgment modes, the key point is whether the message itself has MSH15 or MSH16, or both, and if yes, any acknowledgment mode except static works like the enhanced mode.

Fragment Batch Messages

The HL7 Disassembler also identifies if an HL7 message is a batch or individual message. It can split the message into individual messages if fragmentation is configured in the Configuration Explorer, as shown in Figure 3-8.



HL7SourceBatch

Inbound Batch

Fragmentation required [✓ Yes] No

Recoverable interchange support required False

Figure 3-8. *Party Explorer Batch Definition tab*

Fragmentation should only be configured for a source party as it applies to inbound batch. A sample HL7 batch message is shown below:

```
FHS|^~\&|HL7SourceBatch|A01001A08|HISd2d5|HISdf2df6||fhs8|fhs9|batch212310ff|fhs11|fhs12
BHS|^~\&|bLABs2s3|bLABsf2sf3|bHISd2d3|bHISdf2df3||bhs8|bhs9|BatchComment10|BatchControlID11|Referen
ceBatchcontrolID12
MSH|^~\&|HL7SourceBatch|src4|HL7Dest|dest6|200307092343||ADT^A01^ADT_A01|msgidRx|P|2.5.1
EVN|A01|198808181318||01
PID|9|M11|M11||JOHN^DOE^A^JR||19310615|M||1002-5|303 ANY DRIVE^ANY
CITY^NA^27410|GL|(919)555-5555|(919)555-5555||M|A|M11|987654321|143257NC
NK1|1|HUSBAND
PV1|1|I|2000|A|||004777,SMITH,XXX,J.|||SUR||||AO
MSH|^~\&|HL7SourceBatch|XYZ_ADMITTING|HL7Dest|XYZ_HOSPITAL$|200307092343||ADT^A02|msgidLAB|
P|2.5.1|||
EVN||200008161900||||200008161900
PID|||583085^^^ADT1||AAAD^BBBBN||19201102|M||1002-5|ONE ANY WAY^^ST. TOWN^NA^00130||||||20-98-3085
PV1||E|ED||||1234^AAA^TXXX^P^^DR|5101^AAA^FRED^P^^DR||||||||||V1085^^^AD
T1||||||||||||||||||||||||200008161300
BTS|3|Batch,MessageCount,Comment,Totals|3
FTS|1|File,BatchCount,TrailerComment
```

As you can see, a batch message has both header (FHS and BHS) and trailer segments (BTS and FTS). These are required for the HL7 Disassembler to successfully parse this message as a batch message.

Identify Batch Schedule

The HL7 Disassembler also checks if there is batch schedule configured and sends the message to batch orchestration for batching in scenarios where incoming messages need to be batched before sending to the destination. The HL7 Disassembler uses the MSH5 value to identify if there is a batch schedule configured or not. The batch schedule can be configured in the Configuration Explorer, as shown in Figure 3-9.

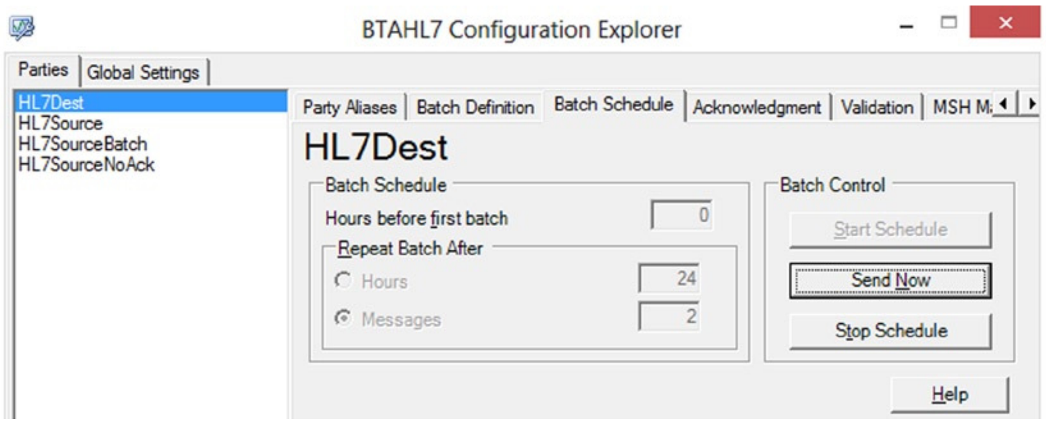


Figure 3-9. *Party Explorer Batch Schedule*

You will learn more about batching in the "Batch Orchestration" section of this chapter.

HL7 Assembler

The HL7 Assembler pipeline component (Microsoft.Solutions.BTAHL7.HL72fAsm) assembles the HL7 disassembled message into a flat HL7 V2.x message. The HL7 Assembler is a simpler component than the HL7 Disassembler component and performs the tasks explained in the following sections.

Identifies the HL7 Party

Similar to the HL7 Disassembler, the HL7 Assembler identifies if there is a HL7 party configured for the outgoing HL7 message. The HL7 Assembler uses the MSH5 value to find a corresponding HL7 party, so if the MSH5 value is "HL7Dest", then the HL7 Assembler will look for a party with the name "HL7Dest", and if a party is found, it will use the configuration as required. If not found, the HL7 Assembler will try to assemble the message with default configurations. If in a message MSH5.1, MSH5.2, and MSH5.3 all are populated, then the HL7 Assembler concatenates all three values and then tries to find the matching party.

Assembling the XML-to-HL7 V2.x Message

The HL7 Assembler main functionality is to convert the single BizTalk multipart HL7 message to a HL7 V2.x message. Each part of the message is converted into the corresponding HL7 segment, so the MSHSegment part converts to MSH segment, and BodySegments converts to all other HL7 segments.

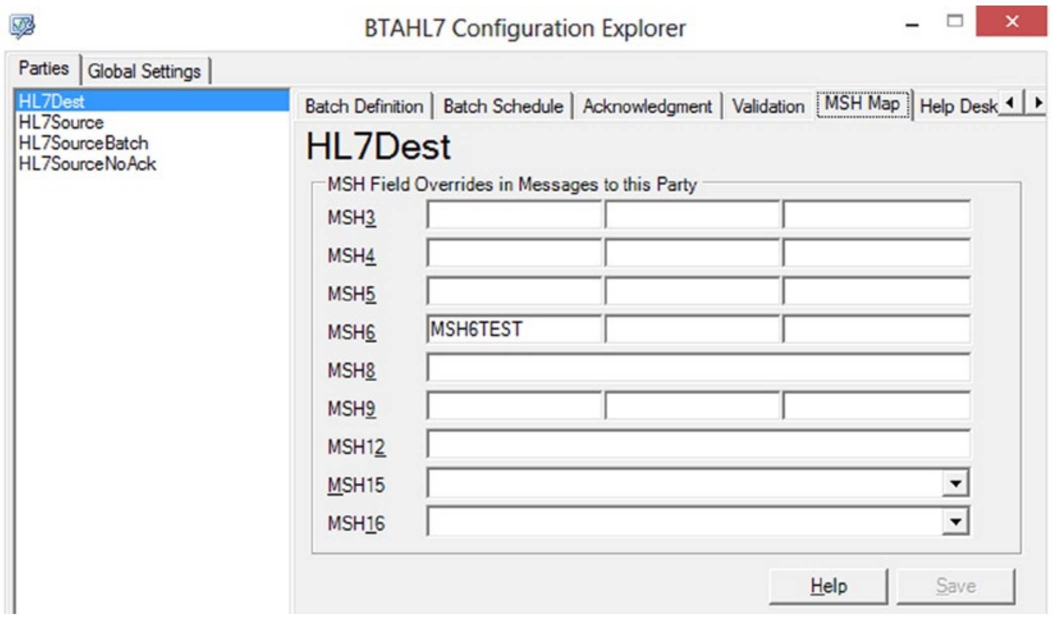
The Assembler also handles the batched XML message and converts it into an HL7 batch message. An HL7 batch XML in BizTalk contains five message parts: FHS, BHS, BTS, FTS, and Messages. You will learn more about the batching later in the section "Batch Orchestration."

Optionally Validating the Body Message

Similar to the HL7 Disassembler, the HL7 Assembler optionally validates the outgoing message. By default, body validation is not applied; in order to do it, the HL7 Assembler should be able to find a destination party with the "Validate Body Segments" option checked on the Validation tab.

Updating the MSH Segment

The Assembler also overrides various MSH segment fields while assembling the message, if a destination party in configuration explorer has MSH map defined as shown in Figure 3-10.



MSH6: MSH6TEST

Figure 3-10. *Party Explorer MSH Map*

The HL7 Assembler overrides the value of fields with the values defined in MSH map; in Figure 3-10, the MSH6 value is set to MSH6TEST for the HL7Dest party. When a message with MSH5 having a HL7Dest value arrives at a send port, the send port updates the value of MSH6 to MSH6TEST irrespective of the original value, as shown below:

```
MSH|^~\&|HL7Source|MCM|HL7Dest|Tes|198808181126|SECURITY|ADT^A01|MSG00001|P|2.5.1|||NE|AL
```

```
MSH|^~\&|HL7Source|MCM|HL7Dest|MSH6TEST|198808181126|SECURITY|ADT^A01|MSG00001|P|2.5.1|||NE|AL
```

This is a very useful functionality in cases where destination systems want specific MSH segment fields.

Batch Orchestration

In many scenarios, you need to batch HL7 messages before sending them to the destination, so one destination system requirement could be to receive batched HL7 message once a day. Batch orchestration provides the functionality to create a batch message of individual HL7 messages before sending the message out.

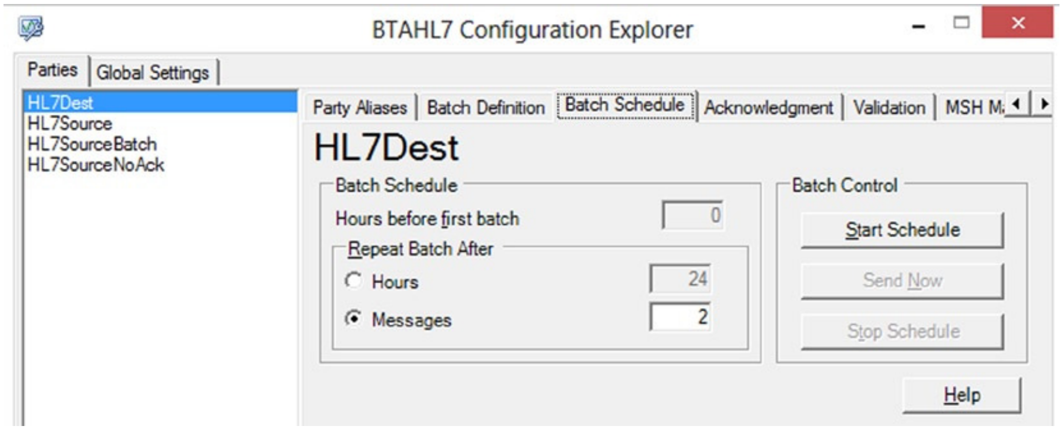
This orchestration is installed in BizTalk Application 1 after the HL7 Accelerator is installed. Let's learn how batch orchestration works and is configured.

Orchestration Design

Batch orchestration design is based upon the BizTalk sequential convoy pattern. It gathers all the messages need to be batched and, once a batch is complete, creates a batched HL7 message and sends it to the message box. Please go to the MSDN article at `http://msdn.microsoft.com/en-us/library/ms942189(v=bts.10).aspx` for more details about the pattern.

Batch Orchestration Activation and Batch Schedule

The batch orchestration needs to be activated before it can start batching HL7 messages. This activation is basically receiving the first message by orchestration via Batch Control Receive Port (BatchControlPort) in BizTalk Application 1. The first message is sent during the configuration of the batch schedule from the HL7 Configuration Explorer, as shown in Figure 3-11.


BTAHL7 Configuration Explorer

Parties | Global Settings

HL7Dest
HL7Source
HL7SourceBatch
HL7SourceNoAck

Party Aliases | Batch Definition | Batch Schedule | Acknowledgment | Validation | MSH M

HL7Dest

Batch Schedule
Hours before first batch: 0
Repeat Batch After
Hours: 24
Messages: 2 (selected)

Batch Control
Start Schedule
Send Now
Stop Schedule

Help

Figure 3-11. *Party Explorer batch schedule*

The batching scheduler allows you to configure a batch schedule for a destination party (MSH5). You can configure how you want to batch the messages, such as whether you want to send a batch every few hours or once you have created a batch of a specified number of messages.

Once you have configured the batch schedule, you need to start the schedule by pressing the Start Schedule button. The Start Schedule button creates a batch activation message, shown below, and drops it into the `\Program Files (x86)\Microsoft BizTalk 2013 R2 Accelerator for HL7\CreateBatchFileDrop` folder.

```
MSH|^~\&|||HL7Dest||20140222182257||BTAHL7^BatchActivation|361d5fe1-c7f4-4497-a454-
28711aa571c9|P|2.4^GLO^DEF
Bat|HL7Dest|0|8760000|2|true
```

This message is then parsed by the HL7 Disassembler like any other HL7 message; in addition, it also reads the batch schedule configuration using the party and creates a message with three parts, a message with the body containing all the schedule details including schedule time and message count as per party batch schedule configruation. This message is then picked up by orchestration. See Figure 3-12.

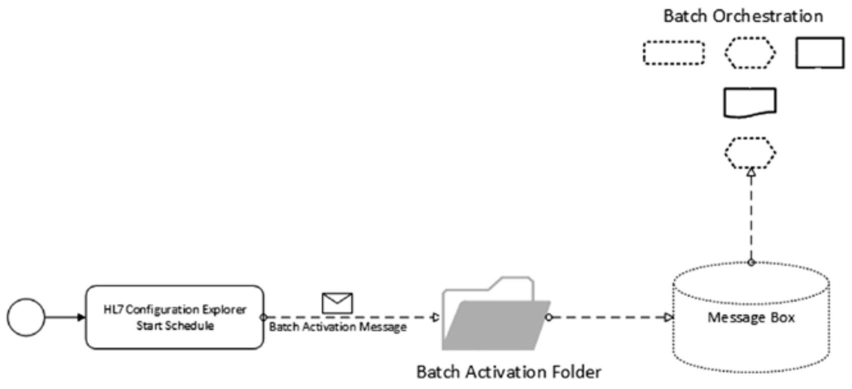
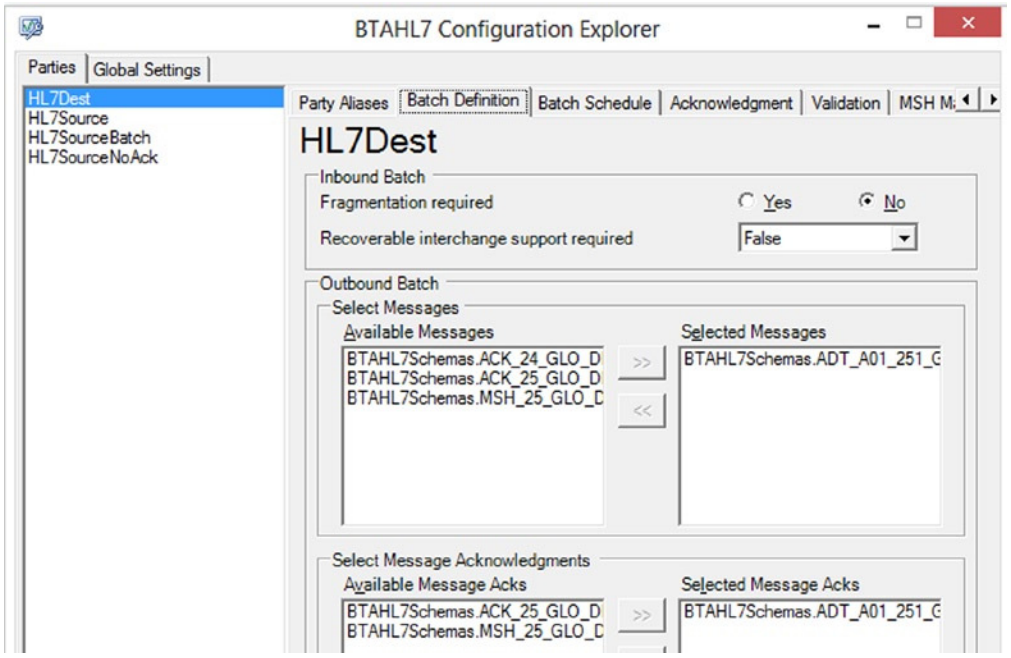


Figure 3-12. *Batch process*

After being activated, orchestration uses correlation on the *MSH5* and *ToBeBatched* context property values to receive messages for batching. You can have number of batch orchestrations activated at the same time for different batch schedules; the correlation makes sure that each batch orchestration receives the message it needs to batch.

Receiving Message for Batching

As you learned earlier, the HL7 Disassembler identifies whether an incoming HL7 message needs to be sent for batching by checking the batch definition from the party configuration. Batch definition allows you to configure which message types need to be batched, as shown in Figure 3-13.



HL7Dest
HL7Source
HL7SourceBatch
HL7SourceNoAck

HL7Dest

Inbound Batch

Fragmentation required ○ Yes [✓ No]

Recoverable interchange support required: False

Outbound Batch

Select Messages

Available Messages:
BTAHL7Schemas.ACK_24_GLO_D
BTAHL7Schemas.ACK_25_GLO_D
BTAHL7Schemas.MSH_25_GLO_D

Selected Messages:
BTAHL7Schemas.ADT_A01_251_G

Select Message Acknowledgments

Available Message Acks:
BTAHL7Schemas.ACK_25_GLO_D
BTAHL7Schemas.MSH_25_GLO_D

Selected Message Acks:
BTAHL7Schemas.ADT_A01_251_G

Figure 3-13. *Party Explorer batch definition selected schemas*

As you can see, an ADT A01 message is configured for batching under the Outbound Batch section. The Available Messages section shows all of the HL7 message schemas deployed in the BizTalk environment.

■ **Note** You need to make sure that the message schema assembly namespace starts with BTAHL7Schemas, otherwise message schemas will not available under the list.

The HL7 Disassembler promotes the context property of ToBeBatched to True along with the MSH5 context property when it finds that the corresponding message type schema is configured for batching. Figure 3-14 defines the process we just described.

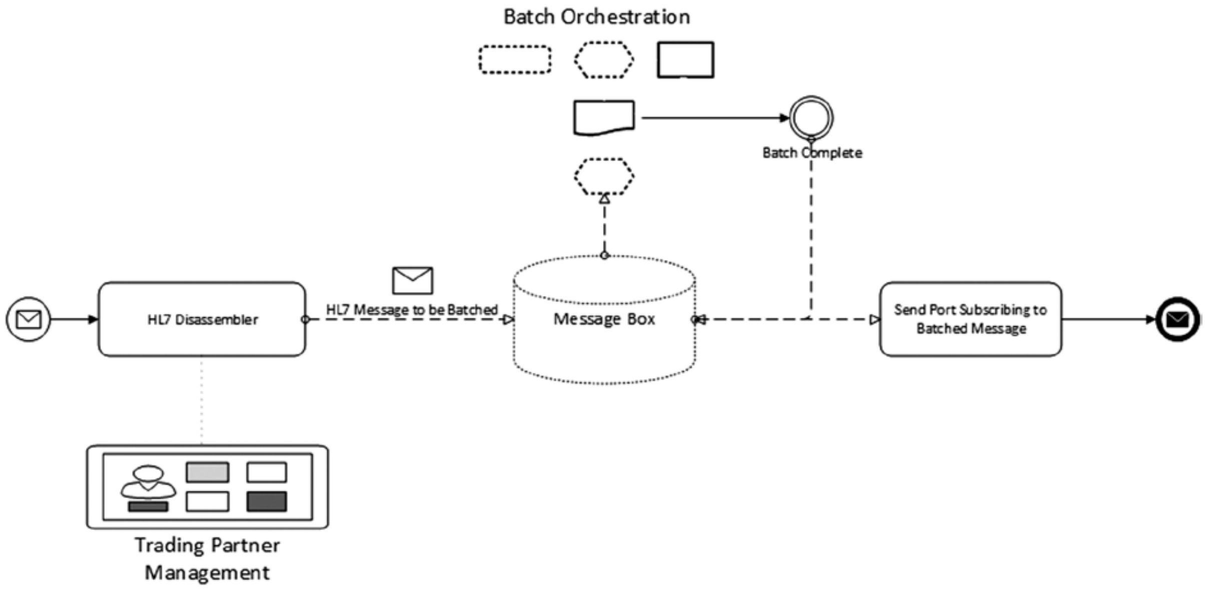

Batch Orchestration
Batch Complete
HL7 Disassembler
HL7 Message to be Batched
Message Box
Send Port Subscribing to Batched Message
Trading Partner Management

Figure 3-14. *Batch orchestration process*

You can also batch acknowledgement messages using the same process.

Batch Termination

You can terminate batch orchestration for a specific party at any point you want via the Configuration Explorer's Batch Schedule tab by pressing the Stop Schedule button. The termination process is same as the activation process: a terminate message is sent to the folder, which is then sent to orchestration for termination. At this time, orchestration creates batch for any pending messages before terminating.

Batch Message

Once a batch is complete, based on the batch schedule, orchestration creates the HL7 batch message consisting of five message parts.

1. *Bhs*: Bhs corresponds to the batch header segment as defined by the HL7 V2.x standard. This segment adds details about the whole batch, similar to a message header segment.
2. *Bts*: Bts corresponds to the batch trailing segment as defined by the HL7 V2.x standard. This segment is the second-to-last segment in the message containing details like the message count in the batch.
3. *Fhs*: Fhs corresponds to the file header segment as defined by the HL7 V2.x standard. This segment is the first segment of the batch message containing header information about the entire batch.
4. *Fts*: Fts corresponds to the file trailing segment as defined by the HL7 V2.x standard. This segment is the last segment in the message, ending the batch message.
5. *Messages*: This is the last part of the multipart batch message containing all the HL7 messages of the batch.

This multipart message is of message type `http://OutboundBatch.BatchMessages#BatchMessages` and has a couple of promoted properties that can be used for creating a subscription of batch messages. Some of these key properties are defined below:

- *BTAHL7MessageType*: The value of this property is set to OutboundBatch.
- *Party*: The value of this property is set to the value of MSH5 of the message.

■ **Note** The entire batching process is based upon MSH5 in the message. You can have different batch schedule for different MSH5 values, which allows different destination messages to have different batching schedules.

MLLP Adapter

MLLP (Minimum Lower Layer Protocol) is the most commonly used protocol to transmit HL7 messages within a local area network (LAN) such as a hospital network. The MLLP protocol uses TCP/IP sockets to transmit messages. Since the TCP/IP protocol is a continuous transmission of data, the MLLP protocol wraps the HL7 message using header and trailing characters to indicate the start and end of an HL7 message. These header and trailing characters are also referred as block characters, as shown in Table 3-1.

Table 3-1. *Block Characters*

Element	Characters	Description
SB	0x0B	Start of HL7 message
EB	0x1C	End of HL7 message
CR	0x0D	Carriage return after the End block

The BizTalk HL7 Accelerator provides a MLLP adapter, which implements MLLP protocol to facilitate HL7 message transfer. The MLLP adapter supports both one-way and two-way communication on both the receive and send side. Table 3-2 lists the properties that can be configured for a MLLP adapter on the receive and send ports.

Table 3-2. *MLLP Adapter Properties*

Property	Send/Receive	Description
<CR> Carriage Return	Both	One of the block characters described above; the default is 0d hex code. The adapter allows you to configure the CR in case it is some other hex code.
<EB> End Block	Both	Configures the End Block characters; the default is 1c hex code.
<SB> Start Block	Both	Configures the Start Block characters; the default is 0b hex code.
Order Delivery	Receive	The MLLP adapter supports order delivery of messages. Configure this property to True to submit messages in the order they are received in your message box. *Please note this does not guarantee the end-to-end ordered delivery of messages; it only ensures messages are received in order and submitted to message box in order.*
Connection Name	Both	This is the name of the connection you are setting up as part of the receive or send ports. The connection name is useful for performance monitoring through performance counters, as counters are available per connection.
Host	Both	For receive, this is the server IP on which you want to receive messages; it should match with the BizTalk host instance server IP. For Send, this is the server IP where you want to send the messages.
Port	Both	The server port on which you want to listen to messages or want to send the messages.
Persistent Connection	Both	This property allows you to configure timeout on the receiving or sending side of the connection. If this is set to True, there will be no socket connection timeout; otherwise, the timeout is set as per the Receive or Send Timeout property.
Receive Timeout	Receive	The connection timeout in milliseconds when the persistent connection is set to False; otherwise it needs to be set to 0 to indicate no timeout.
Send Timeout	Send	Same as Receive Timeout.
Accept ACK Codes	Send	When sending a message to a destination, the destination can send different types of acknowledgment codes. BizTalk can suspend the message being sent if the acknowledgment code is not an expected one. This property allows you to configure acceptable acknowledgment codes for your applications. If an unacceptable acknowledgment code is received, the message being sent will be suspended.
Use Direct Synchronous HL7 ACK	Receive	The HL7 acknowledgment generated by the HL7 Disassembler is submitted to the BizTalk Message Box before it is transmitted to the sending application. This message box round trip introduces latency in overall throughput, especially for the two-way receive port where the sending application is waiting for acknowledgment before sending another message. In order to reduce this latency and increase the message throughput in a two-way scenario, you set this property to True. In this case, the MLLP adapter takes responsibility for generating the acknowledgment and returning the acknowledgment without submitting it to message box. *Please note this setting is ignored for one-way ports.*

(*continued*)

Table 3-2. (*continued*)

Property	Send/Receive	Description
Use MLLP Transport Acknowledgment	Both	Similar to Use Direct Synchronous HL7 ACK, this property enables the adapter to send transport layer acknowledgment to sending application without a message box round trip. There are few differences between the two: 1) The transport layer acknowledgment format is different from the HL7 acknowledgment and looks like <SB><ACK/NAK><EB><CR> where ACK = 0x06 hex code and NAK = 0x15 hex code characters. It's only an indicator whether the message has been successfully submitted to the BizTalk message box or not. 2) This property only works for both receive and send ports; however, both should only be one way. The send port should have Ordered Delivery set to true in the Transport Advanced Options.
Suspend Request Message On MLLP Transport NAK	Send	This property allows you to configure whether you want to suspend the messages sent to a destination in case the receiving system returns a NAK MLLP transport acknowledgment. This configuration is used if you are using a MLLP transport acknowledgment, otherwise it should remain false.
Solicit Response Enabled	Send	In order to receive HL7 acknowledgment sent by the destination system on a one-way send port, set this property to true along with the Submit Receive location URI. This configuration can be used to achieve an asynchronous send operation and still allow you to receive the acknowledgment message. For example, in an orchestration, if a requirement is to send a message on a one-way port and you want to make sure to save the acknowledgment sent by the destination system, use this method.
Submit Receive Location URI for ACK	Send	If Solicit Response is enabled, you need to provide the URI (IP:Port) of the receive location where you want to receive the acknowledgment. As part of the HL7 Accelerator installation, a receive port called "TwowayAckReceivePort" is created for this purpose; however, you can create a new receive location as well. *There is a limitation that for every send host bound to MLLP send port, you need to have one receive location running in the same host to receive acknowledgment. The send host should only be running on one server.*
Accept Missing <SB>	Receive	This is a new property added as part of BizTalk 2013 R2 to accept messages without a Start Block character.
Do not Send <SB>	Send	The same as Accept Missing <SB> option on the Receive side, this property is for Send, to send the messages without the Start Block character.

Some of the usage of these properties depends on the scenarios you want to implement. We will discuss some of these scenarios in Chapter 4.

■ **Note** The MLLP adapter, HL7 Disassembler, and HL7 Assembler all are 32-bit components, so make sure you have a 32-bit host to run them for all BizTalk versions prior to the BizTalk 2013 R2 release. BizTalk 2013 R2 added support for 64-bit.

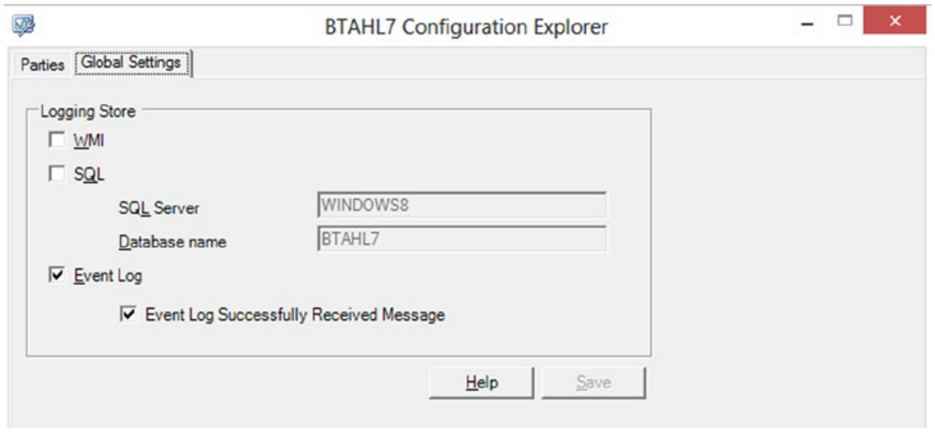
HL7 Logging Service

The HL7 logging service is a windows service installed as part of the HL7 Accelerator installation. The purpose of the HL7 logging service is to provide you with log events related to message processing. By default, the HL7 logging service is configured to log events to the event viewer, and it logs events for all incoming messages on that server including successfully received messages and messages with errors.

The HL7 logging service supports three different log stores, and it can write the events in parallel to all three log stores. These log stores are the following:

- *SQL*: As part of the HL7 Accelerator installation, a database called BTAHL7 is created, which contains table called `EventLog`. If this option is turned on, the HL7 Accelerator log events to this table.
- *WMI*: The HL7 Accelerator allows you to write events to the WMI store. WMI events can be used by monitoring tools such as System Center Operations Manager (SCOM).
- *Event Log*: This is the default option selected and is the only option which allows you to filter successful message events.

All these options can be configured from HL7 Configuration Explorer Global Settings tab, as shown in Figure 3-15.


BTAHL7 Configuration Explorer

Parties | Global Settings

Logging Store
- [ ] WMI
- [ ] SQL
  - SQL Server: WINDOWS8
  - Database name: BTAHL7
- [✓] Event Log
  - [✓] Event Log Successfully Received Message

***Figure 3-15.** Party Explorer global settings*

SQL Logging Store

If you are logging to SQL Server, all the event information is inserted into the `dbo.EventLog` table. Table 3-3 shows the fields contained within the `dbo.EventLog` table.

Table 3-3. *EventLog SQL Table*

Field Name	Type	Nullable
ID	bigint	No
Data	ntext	Yes
CategoryNumber	int	No
EntryType	int	No
EventId	int	No
MachineName	nvarchar(256)	No
Message	ntext	No
Source	nvarchar(212)	No
TimeGenerated	datetime	No
UserName	nvarchar(256)	No
MsgGuid	uniqueidentifier	Yes
SvcGuid	uniqueidentifier	Yes
Artifact	nvarchar(128)	Yes
ArtType	tinyint	Yes
Operation	tinyint	Yes
MgmtServer	nvarchar(256)	Yes
MgmtDatabase	nvarchar(256)	Yes

■ **Tip** One advantage of using the SQL Event Log table is that you can query the table to create reports. This can come in handy if you need to provide information about a particular transaction to the sender or receiver.

WMI Event Logging Store

The Windows Management Instrumentation (WMI) Logging Store allows for the capturing of events. Table 3-4 shows the properties available from the WMI Event Store.

Table 3-4. *WMI Event Properties*

Property	Data Type
artifact	String
artifactType	SInt32
blob	String
categoryNumber	SInt32
entryType	SInt32

(*continued*)

Table 3-4. (*continued*)

Property	Data Type
eventId	SInt32
machineName	String
message	String
messageInstance	String
mgmgtDB	String
mgmtServer	String
operation	UInt8
SECURITY_DESCRIPTOR	UInt8
serviceInstance	String
source	String
TIME_CREATED	UInt8
timeGenerated	String
username	String

WMI Event Reporting

You can create custom classes to view the events captured using WMI logging. The following samples show you how to do this. Listing 3-1 and 3-2 show ***accessing WMI events on a local machine***.

Listing 3-1. Asynchronous Method

```
using System;
using System.Management;
using System.Windows.Forms;

namespace WMIHL7Logging
{
    public class WMIReceiveEvent
    {
        public WMIReceiveEvent()
        {
            try
            {
                WqlEventQuery query = new WqlEventQuery("SELECT * FROM PackageEvent");

                ManagementEventWatcher watcher = new ManagementEventWatcher(query);
                Console.WriteLine("Waiting for an event...");

                watcher.EventArrived += new EventArrivedEventHandler(HandleEvent);

                // Start listening for events
                watcher.Start();
                // Do something while waiting for events
                System.Threading.Thread.Sleep(10000);

                // Stop listening for events
                watcher.Stop();
                return;
            }
            catch(ManagementException err)
            {
                MessageBox.Show("An error occurred while trying to receive an event: " + err.Message);
            }
        }

        private void HandleEvent(object sender,
            EventArrivedEventArgs e)
        {
            Console.WriteLine("PackageEvent event occurred.");
        }

        public static void Main()
        {
            WMIReceiveEvent receiveEvent = new WMIReceiveEvent();
            return;
        }

    }
}
```

Listing 3-2. Synchronous Method

```
using System;
using System.Management;
using System.Windows.Forms;

namespace WMIHL7Logging
{
    public class WMIReceiveEvent
    {
        public static void Main()
        {
            try
            {
                WqlEventQuery query = new WqlEventQuery("SELECT * FROM PackageEvent");

                ManagementEventWatcher watcher = new ManagementEventWatcher(query);
                Console.WriteLine("Waiting for an event...");

                ManagementBaseObject eventObj = watcher.WaitForNextEvent();

                Console.WriteLine("{0} event occurred.", eventObj["__CLASS"]);
                // Cancel the event subscription
                watcher.Stop();
                return;
            }
            catch(ManagementException err)
            {
                MessageBox.Show("An error occurred while trying to receive an event: " + err.Message);
            }
        }
    }
}
```

Listings 3-3 and 3-4 show accessing events on remote machines.

Listing 3-3. *Synchronous Method*

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Management;

namespace WMIHL7Logging
{
    public class WMIReceiveEvent : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Label userNameLabel;
        private System.Windows.Forms.TextBox userNameBox;
        private System.Windows.Forms.TextBox passwordBox;
        private System.Windows.Forms.Label passwordLabel;
        private System.Windows.Forms.Button OKButton;
        private System.Windows.Forms.Button cancelButton;

        private System.ComponentModel.Container components = null;

        public WMIReceiveEvent()
        {
            InitializeComponent();
        }

        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }
        private void InitializeComponent()
        {
            this.userNameLabel = new System.Windows.Forms.Label();
            this.userNameBox = new System.Windows.Forms.TextBox();
            this.passwordBox = new System.Windows.Forms.TextBox();
            this.passwordLabel = new System.Windows.Forms.Label();
            this.OKButton = new System.Windows.Forms.Button();
            this.cancelButton = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // userNameLabel
            //
            this.userNameLabel.Location = new System.Drawing.Point(16, 8);
            this.userNameLabel.Name = "userNameLabel";
            this.userNameLabel.Size = new System.Drawing.Size(160, 32);
            this.userNameLabel.TabIndex = 0;
            this.userNameLabel.Text = "Enter the user name for the remote computer:";
            //
            // userNameBox
            //
            this.userNameBox.Location = new System.Drawing.Point(160, 16);
            this.userNameBox.Name = "userNameBox";
            this.userNameBox.Size = new System.Drawing.Size(192, 20);
            this.userNameBox.TabIndex = 1;
            this.userNameBox.Text = "";
            //
            // passwordBox
            //
            this.passwordBox.Location = new System.Drawing.Point(160, 48);
            this.passwordBox.Name = "passwordBox";
            this.passwordBox.PasswordChar = '*';
            this.passwordBox.Size = new System.Drawing.Size(192, 20);
            this.passwordBox.TabIndex = 3;
            this.passwordBox.Text = "";
            //
            // passwordLabel
            //
            this.passwordLabel.Location = new System.Drawing.Point(16, 48);
            this.passwordLabel.Name = "passwordLabel";
            this.passwordLabel.Size = new System.Drawing.Size(160, 32);
            this.passwordLabel.TabIndex = 2;
            this.passwordLabel.Text = "Enter the password for the remote computer:";
            //
            // OKButton
            //
            this.OKButton.Location = new System.Drawing.Point(40, 88);
            this.OKButton.Name = "OKButton";
            this.OKButton.Size = new System.Drawing.Size(128, 23);
            this.OKButton.TabIndex = 4;
            this.OKButton.Text = "OK";
            this.OKButton.Click += new System.EventHandler(this.OKButton_Click);
        //
        // cancelButton
        //
        this.cancelButton.DialogResult = System.Windows.Forms.DialogResult.Cancel;
        this.cancelButton.Location = new System.Drawing.Point(200, 88);
        this.cancelButton.Name = "cancelButton";
        this.cancelButton.Size = new System.Drawing.Size(128, 23);
        this.cancelButton.TabIndex = 5;
        this.cancelButton.Text = "Cancel";
        this.cancelButton.Click += new System.EventHandler(this.cancelButton_Click);
        //
        // MyQuerySample
        //
        this.AcceptButton = this.OKButton;
        this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
        this.CancelButton = this.cancelButton;
        this.ClientSize = new System.Drawing.Size(368, 130);
        this.ControlBox = false;
        this.Controls.Add(this.cancelButton);
        this.Controls.Add(this.OKButton);
        this.Controls.Add(this.passwordBox);
        this.Controls.Add(this.passwordLabel);
        this.Controls.Add(this.userNameBox);
        this.Controls.Add(this.userNameLabel);
        this.Name = "MyQuerySample";
        this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
        this.Text = "Remote Connection";
        this.ResumeLayout(false);

    }

    [STAThread]
    static void Main()
    {
        Application.Run(new WMIReceiveEvent());
    }

    private void OKButton_Click(object sender, System.EventArgs e)
    {
        try
        {
            ConnectionOptions connection = new ConnectionOptions();
            connection.Username = userNameBox.Text;
            connection.Password = passwordBox.Text;
            connection.Authority = "ntlmdomain:<DOMAINNAME>";

            ManagementScope scope = new ManagementScope(
                "\\\\<FullComputerName>\\root\\Default", connection);
            scope.Connect();

            WqlEventQuery query = new WqlEventQuery(
                "SELECT * FROM PackageEvent");
                ManagementEventWatcher watcher = new ManagementEventWatcher(scope, query);
                Console.WriteLine("Waiting for an event on <FullComputerName>...");

                watcher.EventArrived +=
                    new EventArrivedEventHandler(
                    HandleEvent);

                // Start listening for events
                watcher.Start();

                // Do something while waiting for events
                System.Threading.Thread.Sleep(10000);

                // Stop listening for events
                watcher.Stop();
                return;
            }
            catch(ManagementException err)
            {
                MessageBox.Show("An error occurred while trying to receive an event: " + err.Message);
            }
            catch(System.UnauthorizedAccessException unauthorizedErr)
            {
                MessageBox.Show("Connection error (user name or password might be incorrect): " +
                unauthorizedErr.Message);
            }
        }

        private void cancelButton_Click(object sender, System.EventArgs e)
        {
            Close();
        }

        private void HandleEvent(object sender,
            EventArrivedEventArgs e)
        {
            Console.WriteLine("PackageEvent event occurred.");
        }
    }
}
```

Listing 3-4. *Asynchronous Method*

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Management;
```

```
namespace WMIHL7Logging
{
    public class WMIReceiveEvent : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Label userNameLabel;
        private System.Windows.Forms.TextBox userNameBox;
        private System.Windows.Forms.TextBox passwordBox;
        private System.Windows.Forms.Label passwordLabel;
        private System.Windows.Forms.Button OKButton;
        private System.Windows.Forms.Button cancelButton;
        private System.ComponentModel.Container components = null;

        public WMIReceiveEvent()
        {
            InitializeComponent();
        }

        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        private void InitializeComponent()
        {
            this.userNameLabel = new System.Windows.Forms.Label();
            this.userNameBox = new System.Windows.Forms.TextBox();
            this.passwordBox = new System.Windows.Forms.TextBox();
            this.passwordLabel = new System.Windows.Forms.Label();
            this.OKButton = new System.Windows.Forms.Button();
            this.cancelButton = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // userNameLabel
            //
            this.userNameLabel.Location = new System.Drawing.Point(16, 8);
            this.userNameLabel.Name = "userNameLabel";
            this.userNameLabel.Size = new System.Drawing.Size(160, 32);
            this.userNameLabel.TabIndex = 0;
            this.userNameLabel.Text = "Enter the user name for the remote computer:";
            //
            // userNameBox
            //
            this.userNameBox.Location = new System.Drawing.Point(160, 16);
            this.userNameBox.Name = "userNameBox";
            this.userNameBox.Size = new System.Drawing.Size(192, 20);
            this.userNameBox.TabIndex = 1;
            this.userNameBox.Text = "";
            //
            // passwordBox
            //
            this.passwordBox.Location = new System.Drawing.Point(160, 48);
            this.passwordBox.Name = "passwordBox";
            this.passwordBox.PasswordChar = '*';
            this.passwordBox.Size = new System.Drawing.Size(192, 20);
            this.passwordBox.TabIndex = 3;
            this.passwordBox.Text = "";
            //
            // passwordLabel
            //
            this.passwordLabel.Location = new System.Drawing.Point(16, 48);
            this.passwordLabel.Name = "passwordLabel";
            this.passwordLabel.Size = new System.Drawing.Size(160, 32);
            this.passwordLabel.TabIndex = 2;
            this.passwordLabel.Text = "Enter the password for the remote computer:";
            //
            // OKButton
            //
            this.OKButton.Location = new System.Drawing.Point(40, 88);
            this.OKButton.Name = "OKButton";
            this.OKButton.Size = new System.Drawing.Size(128, 23);
            this.OKButton.TabIndex = 4;
            this.OKButton.Text = "OK";
            this.OKButton.Click += new System.EventHandler(this.OKButton_Click);
            //
            // cancelButton
            //
            this.cancelButton.DialogResult = System.Windows.Forms.DialogResult.Cancel;
            this.cancelButton.Location = new System.Drawing.Point(200, 88);
            this.cancelButton.Name = "cancelButton";
            this.cancelButton.Size = new System.Drawing.Size(128, 23);
            this.cancelButton.TabIndex = 5;
            this.cancelButton.Text = "Cancel";
            this.cancelButton.Click += new System.EventHandler(this.cancelButton_Click);
            //
            // MyQuerySample
            //
            this.AcceptButton = this.OKButton;
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.CancelButton = this.cancelButton;
            this.ClientSize = new System.Drawing.Size(368, 130);
            this.ControlBox = false;
            this.Controls.Add(this.cancelButton);
            this.Controls.Add(this.OKButton);
            this.Controls.Add(this.passwordBox);
            this.Controls.Add(this.passwordLabel);
            this.Controls.Add(this.userNameBox);
            this.Controls.Add(this.userNameLabel);
```

```
    this.Name = "MyQuerySample";
    this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
    this.Text = "Remote Connection";
    this.ResumeLayout(false);

}

[STAThread]
static void Main()
{
    Application.Run(new WMIReceiveEvent());
}

private void OKButton_Click(object sender, System.EventArgs e)
{
    try
    {
        ConnectionOptions connection = new ConnectionOptions();
        connection.Username = userNameBox.Text;
        connection.Password = passwordBox.Text;
        connection.Authority = "ntlmdomain:<DOMAINNAME>";

        ManagementScope scope = new ManagementScope(
            "\\\\<FullComputerName>\\root\\Default", connection);
        scope.Connect();

        WqlEventQuery query = new WqlEventQuery(
            "SELECT * FROM PackageEvent");

        ManagementEventWatcher watcher = new ManagementEventWatcher(scope, query);
        Console.WriteLine("Waiting for an event on <FullComputerName> ...");

        watcher.EventArrived +=
            new EventArrivedEventHandler(
            HandleEvent);

        // Start listening for events
        watcher.Start();

        // Do something while waiting for events
        System.Threading.Thread.Sleep(10000);

        // Stop listening for events
        watcher.Stop();
        return;
    }
    catch(ManagementException err)
    {
        MessageBox.Show("An error occurred while trying to receive an event: " + err.Message);
    }
```

```
            catch(System.UnauthorizedAccessException unauthorizedErr)
            {
                MessageBox.Show("Connection error (user name or password might be incorrect): "
                + unauthorizedErr.Message);
            }
        }

        private void cancelButton_Click(object sender, System.EventArgs e)
        {
            Close();
        }

        private void HandleEvent(object sender,
            EventArrivedEventArgs e)
        {
            Console.WriteLine("PackageEvent event occurred.");
        }
    }
}
```

■ **Note** It's important for this service to run all the time even if in global settings all logging settings are turned off.

HL7 V2.xml Receive and Send Pipelines

BizTalk provides custom pipelines to disassemble an HL7 V2.xml-encoded messages and to assemble the XML into HL7 V2.xml-encoded messages. These pipelines are deployed to the BizTalk Application 1 application after the HL7 Accelerator installation and can be used in BizTalk Receive and Send Ports like any other pipelines.

Utilities

There are few utilities provided as part of the HL7 Accelerator installation in the `%SystemDrive%\Program Files (x86)\Microsoft BizTalk 2013 R2 Accelerator for HL7\SDK\MLLP Utilities` folder for 32-bit installations and `%SystemDrive%\Program Files (x86)\Microsoft BizTalk 2013 R2 Accelerator for HL7\SDK\2XML Utilities` `folders` for 64-bit installations.

MLLPSend

The MLLPSend utility can be used to send an HL7 message to any location from the command prompt. This utility is useful to test the HL7 message in local/test environment. Table 3-5 shows the parameter details.

Table 3-5. *MLLPSend Parameters*

Parameter	Description
/I	<IP> denotes the address to send to. The default is localhost.
/P	<PORT> denotes the port number to send to. The default is 11000.
/F	<File Name | Text> donates either the file full path of the message you want to send or hl7 message as text.
/REPEAT <n>	Sends the message n number times.
/TWOWAY	Waits for acknowledgment after sending the message. In case you use a file name to send, the acknowledgment is saved in the `<file name>.RESPONSE` file.
/UseMLLPTransACK	Waits for MLLP transport acknowledgment.
/SB	Sets the ASCII value of the Start Block Delimiter Byte; the default is none. Its value usually is 11.
/EB	Sets the ASCII value of the End Block Delimiter Byte; the default is none. Its value usually is 28.
/CR	Sets the ASCII value of the Carriage Return Delimiter Byte; the default is none. Its value usually is 13.

Following exmaple is sending message in c:\temp\hl7.txt to a IP 127.0.0.1 on port 11001 in a twoway communication.

```
e.g. mllpsend /I 127.0.0.1 /P 11001 /F C:\temp\hl7.txt /TWOWAY /SB 11 /EB 28 /CR 13
```

MLLPReceive

MLLPReceive can be used to create a host to receive HL7 messages. This utility is useful for sending a message from BizTalk to a destination. MLLPReceive parameters detail are provided in Table 3-6.

Table 3-6. *MLLPReceive Parameters*

Parameter	Description
/I	<IP> denotes the address where BizTalk can send data to.
/P	<PORT> denotes the listening send port. The default is 12000.
/D	The directory where all received messages will be stored; the default directory is `%TEMP%`.
/SPLIT	Splits the received data in to separate messages based on delimiters SB, EB, and CR.
/STATICACK	Sends a static acknowledgment to a sender.
/MLLPTransACK	Sends a MLLP transport positive acknowledgment
/MLLPTransNAK	Sends a MLLP transport negative acknowledgment
/HL7ACK	<File Full Path> sends the HL7 acknowledgment as specified in file full path.
/SB	Sets the ASCII value of the Start Block Delimiter Byte; the default is none. Its value usually is 11.
/EB	Sets the ASCII value of the End Block Delimiter Byte; the default is none. Its value usually is 28.
/CR	Sets the ASCII value of the Carriage Return Delimiter Byte; the default is none. Its value usually is 13.

Following example is creating a listner on local host port 21110. When a message arrives, message will be saved to C:\Work\TestFiles\MLLPReceivedFiles folder and an acknowledgment from C:\Work\TestFiles\Acks folder will be sent to the sender.

```
e.g. mllpreceive /I 127.0.0.1 /P 21110 /SB 11 /EB 28 /CR 13 /HL7ACK
C:\Work\TestFiles\Acks\ack.txt /D C:\Work\TestFiles\MLLPReceivedFiles
```

Update2XmlSchema

This utility is used to update HL7 V2.xml-encoded message schemas to modify them so that they can be used within a BizTalk solution. This utility only supports version 2.3.1, 2.4 or 2.5. Table 3-7 shows the parameter details.

Table 3-7. *Update2XmlSchema Parameters*

Parameter	Description
/s	Full path of the original HL7 schemas downloaded from the HL7 web site.
/v	Version of HL7 2.3.1 or 2.4 or 2.5.

This utility updates the message schemas as required by BizTalk and moves the updated schemas to the `\Program Files (x86)\Microsoft BizTalk 2013 R2 Accelerator for HL7\Templates\Schemas\V2.XML` folder.

■ **Note** After running the utility and adding these schemas to the BizTalk project, make sure you change the Root Reference property of schema; so for ADT_A01_25_GLO_DEF schema, change the root reference to ADT_A01.

Summary

In this chapter, you learned about the BizTalk HL7 Accelerator architecture, its various components, and how they work together. *Understanding these concepts is the key to any BizTalk implementation using the HL7 Accelerator.* In Chapter 4, you are going to see the HL7 Accelerator in action.

CHAPTER 4

The HL7 Accelerator in Action

In the previous chapter, you learned about the HL7 Accelerator architecture, including all the components of the HL7 Accelerator. In this chapter, you will learn implementation of various scenarios using the HL7 Accelerator's components. We will discuss the following scenarios:

- Message exchange
- Simple message routing with default configurations
- Turning off message body validation
- Accepting messages with customized fields
- Customized acknowledgment behavior
- Message transformation
- Sending a HL7 message via the MLLP adapter
- Batching

Message Exchange

In an HL7 message exchange, there are always two entities, the initiating and responding system. Each system is both a sender and receiver of messages. The initiator system first sends then receives, and the responding system first receives and then sends, as shown in Figure 4-1.

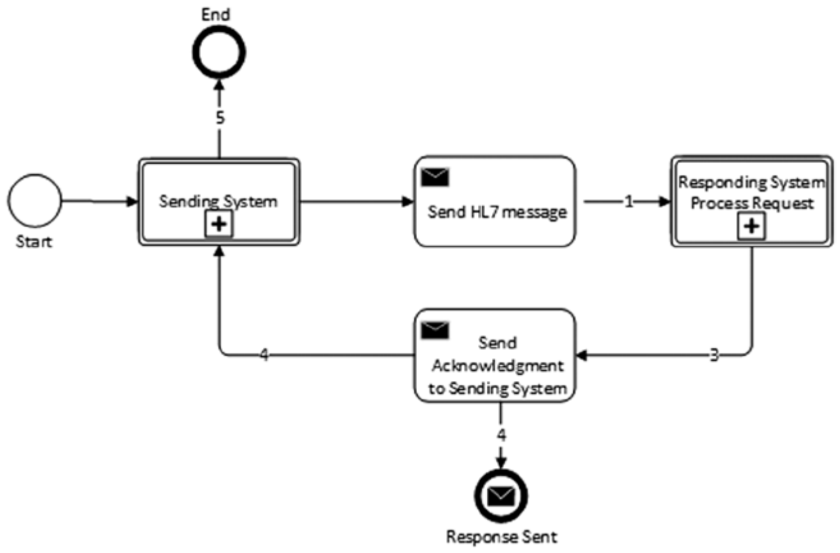

End
5
Start
Sending System
Send HL7 message
1
Responding System Process Request
3
Send Acknowledgment to Sending System
4
4
Response Sent

Figure 4-1. *Basic message exchange scenario*

The following are the steps for any message exchange between the initiating and responding systems.

1. The message initiator sends the message to the responding system.
2. The responding system receives the message and processes it.
3. The responding system sends the acknowledgment message.
4. The initiator receives the response message and process it.

Message processing on the responding system side depends on the acknowledgment mode used. How the processing differs is detailed in upcoming subsections.

Message Processing—Original Mode

The responding system processing in original mode is shown in Figure 4-2.

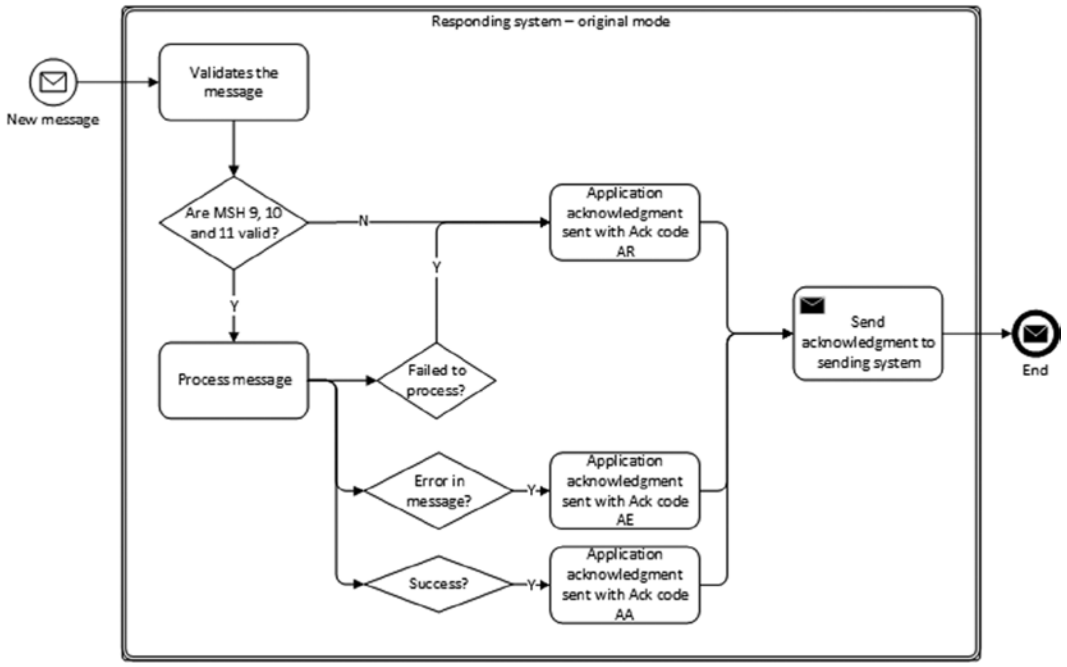

Responding system – original mode
New message
Validates the message
Are MSH 9, 10 and 11 valid?
N
Y
Application acknowledgment sent with Ack code AR
Y
Process message
Failed to process?
Send acknowledgment to sending system
End
Error in message?
Y
Application acknowledgment sent with Ack code AE
Success?
Y
Application acknowledgment sent with Ack code AA

Figure 4-2. *Responding system—original mode*

The responding system after receiving the message does at least the following validation in original mode.

1. The value in MSH9 (message type) is one that is acceptable to the receiver.
2. The value in MSH12 (version) is one that is acceptable for receiver.
3. The value is MSH11 (processing ID) is appropriate for the application process handling the message.

If any of the above validation fails, then the responding system returns an acknowledgment with acknowledgment code AR to the sending system; in other words, it will reject the message.

If the above validation fails, then it moves to the following steps.

4. It processes the message successfully, generating the acknowledgment with acknowledgment code AA in MSA1.
5. In case of an error, it sends the acknowledgment with acknowledgment code AE with details of the error in ERR segment.
6. It fails to process the message due to internal system issues like the system is down or an internal error, it rejects the message, and it sends the acknowledgment with acknowledgment code AR. In such a situation, the sending application should be able to resubmit the message when the receiving system issue is resolved.

The acknowledgment with the appropriate code is returned to the sending system. Then the sending system processes the response.

Message Processing—Enhanced Mode

In enhanced mode, the message processing by the responding system is shown in Figure 4-3. As you know, the message is committed to safe storage for further processing. The responding system determines, based on following factors, whether message is good to commit to safe storage:

- Availability of safe storage
- The syntactical correctness of the message, if the design of the receiving application includes such validation at this stage
- The validation of MSH9, MSH11, and MSH12, if the design of the receiving application includes such validation at this stage

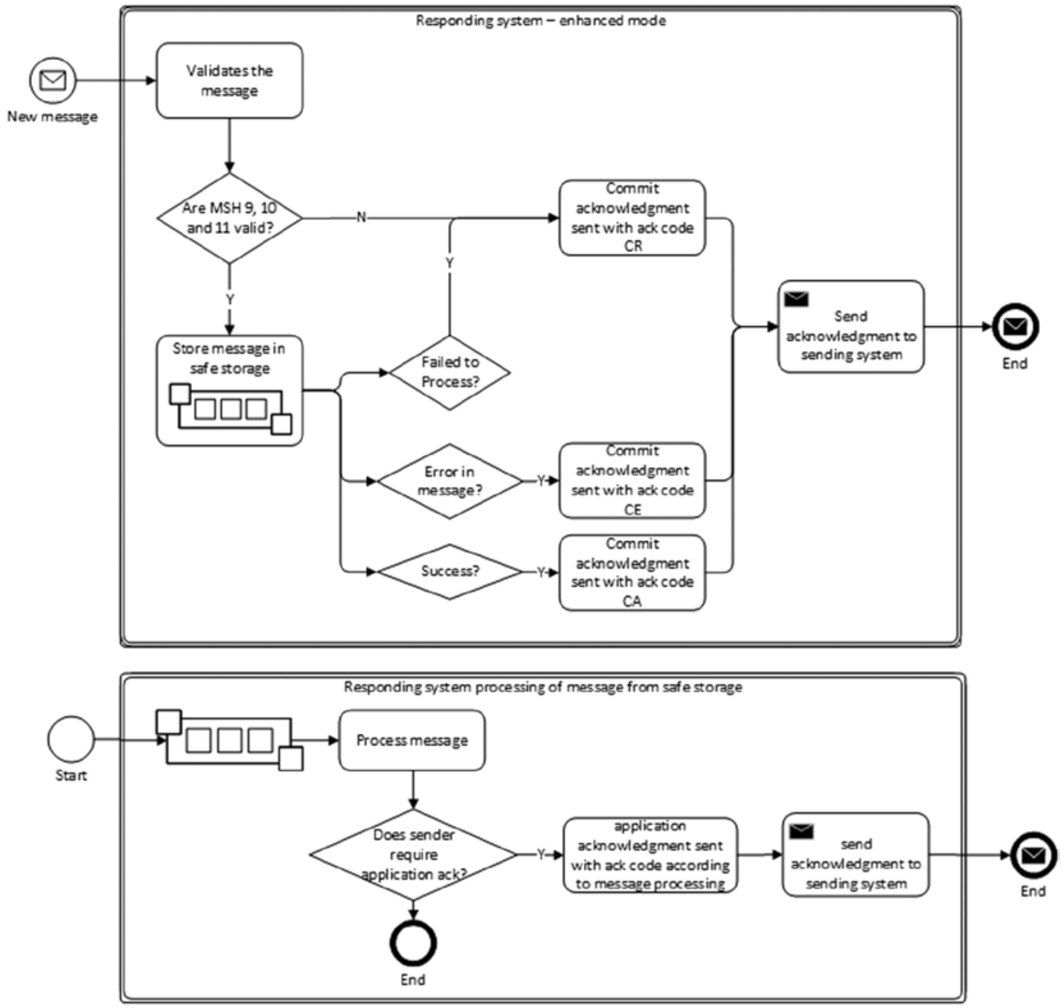

Responding system – enhanced mode
New message
Validates the message
Are MSH 9, 10 and 11 valid?
N
Y
Commit acknowledgment sent with ack code CR
Store message in safe storage
Failed to Process?
Send acknowledgment to sending system
End
Error in message?
Commit acknowledgment sent with ack code CE
Success?
Commit acknowledgment sent with ack code CA
Responding system processing of message from safe storage
Start
Process message
Does sender require application ack?
application acknowledgment sent with ack code according to message processing
send acknowledgment to sending system
End
End

Figure 4-3. *Responding system—enhanced mode*

It then checks for MSH15 in the received message and sees if the initiating system requires an Accept Acknowledgment; if it does, then a general acknowledgment is sent to the sending system with one of following acknowledgment codes:

- CA - If the message can be accepted for processing
- CR - If one of the values in MSH9, MSH11, and MSH12 is not acceptable to the receiving application
- CE - If the message cannot be accepted due to any other reason

After message processing, the receiving application checks MSH16 in the received message to see if the sending application requires application acknowledgment. If it does, then the receiving application sends an application acknowledgment. In such a case, the receiving application becomes the initiator and the sending application becomes the receiving application, and the whole process of sending an application acknowledgment become any other message exchange between two systems except that in application acknowledgment message cannot have MSH16.

■ **Note** The original acknowledgment protocol is equivalent to the enhanced acknowledgment protocol with MSH-15-accept acknowledgment type = NE and MSH-16-application acknowledgment type = AL, and with the application acknowledgment message defined so that it never requires an accept acknowledgment (MSH-15-accept acknowledgment type = NE).

Simple Message Routing

In this scenario, you will learn how to use the various HL7 Accelerator components together to build a simple message routing application. As shown in Figure 4-1, you will implement a responding system using the BizTalk HL7 Accelerator and see how two different acknowledgment modes can be achieved.

■ **Note** In order to implement this scenario, you will use HL7 version v2.5.1 ADT message type, with triggering event A01.

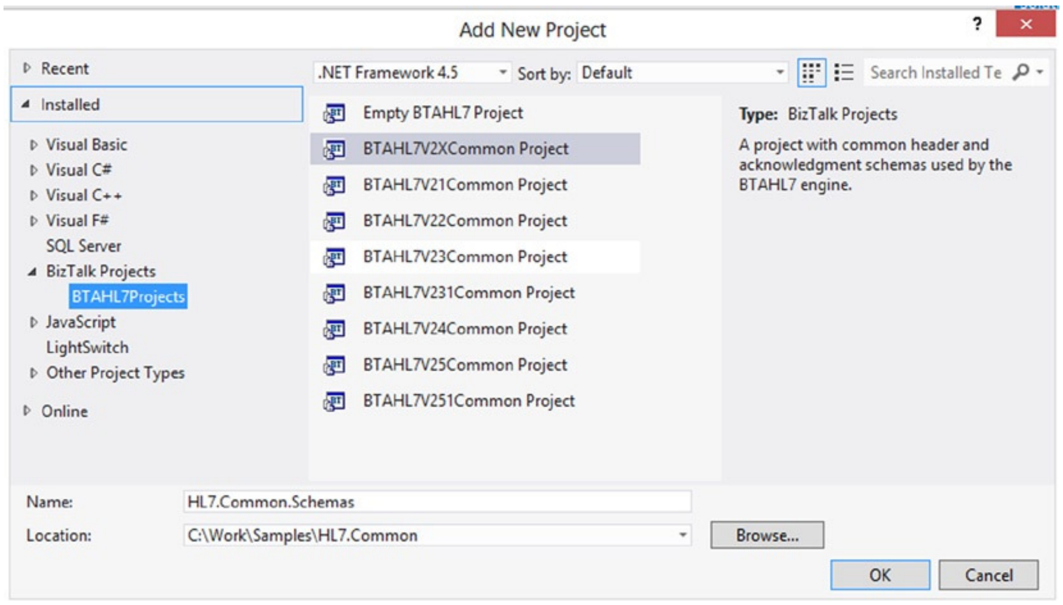
The implementation of simple message routing is same, irrespective of the acknowledgment mode you want to use. You will see how to switch from one acknowledgment mode to another using configuration.

Building the Solution

The following are the steps you will take to create the BizTalk solution for this scenario.

Common Project

1. Create a Visual Studio Blank Solution named HL7.Common.
2. Create a new BTAHL7V2XCommon project in the solution with the name HL7.Common. Schemas, as shown in Figure 4-4.



Type: BizTalk Projects

A project with common header and acknowledgment schemas used by the BTAHL7 engine.

Name: HL7.Common.Schemas

Location: C:\Work\Samples\HL7.Common

Figure 4-4. *Common project template*

The project in Solution Explorer will look like Figure 4-5.

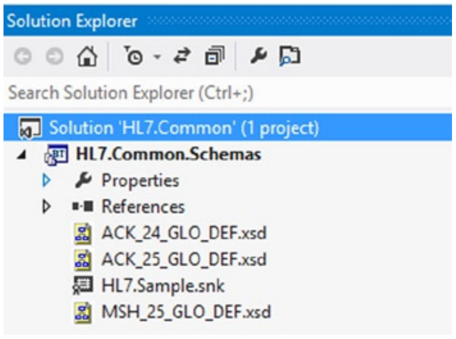


Figure 4-5. *Common project in Solution Explorer*

3. Change the BizTalk Deployment properties to deploy this project to BizTalk under the HL7. Common BizTalk application.

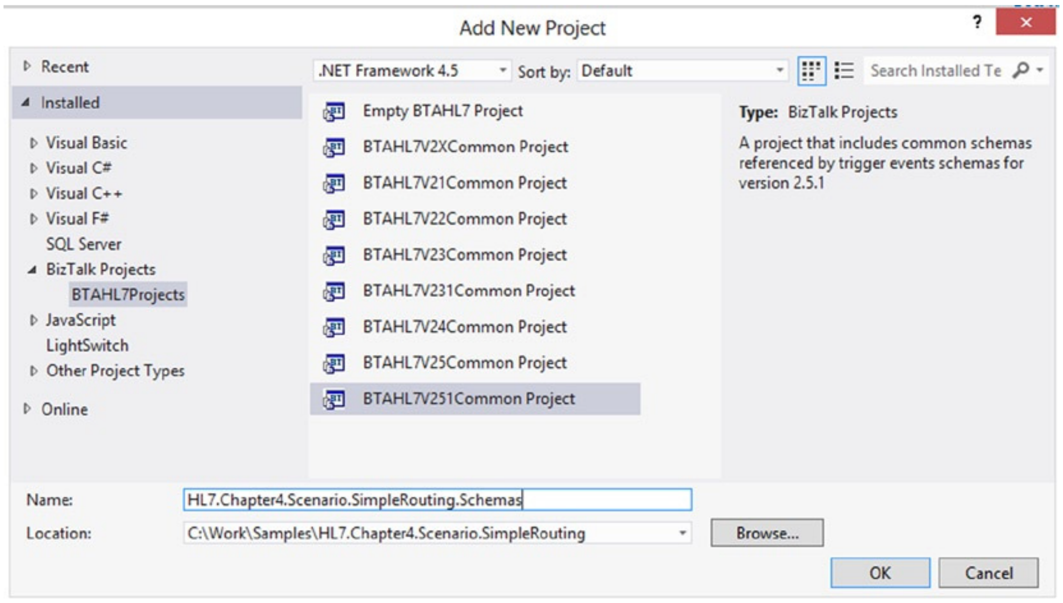
4. Sign the project using a strong name key and deploy it.

■ **Note** Message header (MSH_25_GLO_DEF.xsd) and acknowledgment (ACK_24_GLO_DEF and ACK_25_GLO_DEF) message schemas can only be deployed once in BizTalk Group for all applications.

Build HL7 Version Schemas Project

You will learn the steps to add specific HL7 message schemas to a BizTalk project. For this scenario, you will be adding version 2.5.1 ADT message type with triggering event A01.

1. Create a new Visual Studio blank solution with the name HL7.Chapter4.Scenarios.
2. Create a BTAHL7V251Common project for the solution, as shown in Figure 4-6.


Add New Project

Installed > BizTalk Projects > BTAHL7Projects

.NET Framework 4.5 | Sort by: Default

- Empty BTAHL7 Project
- BTAHL7V2XCommon Project
- BTAHL7V21Common Project
- BTAHL7V22Common Project
- BTAHL7V23Common Project
- BTAHL7V231Common Project
- BTAHL7V24Common Project
- BTAHL7V25Common Project
- BTAHL7V251Common Project

Type: BizTalk Projects

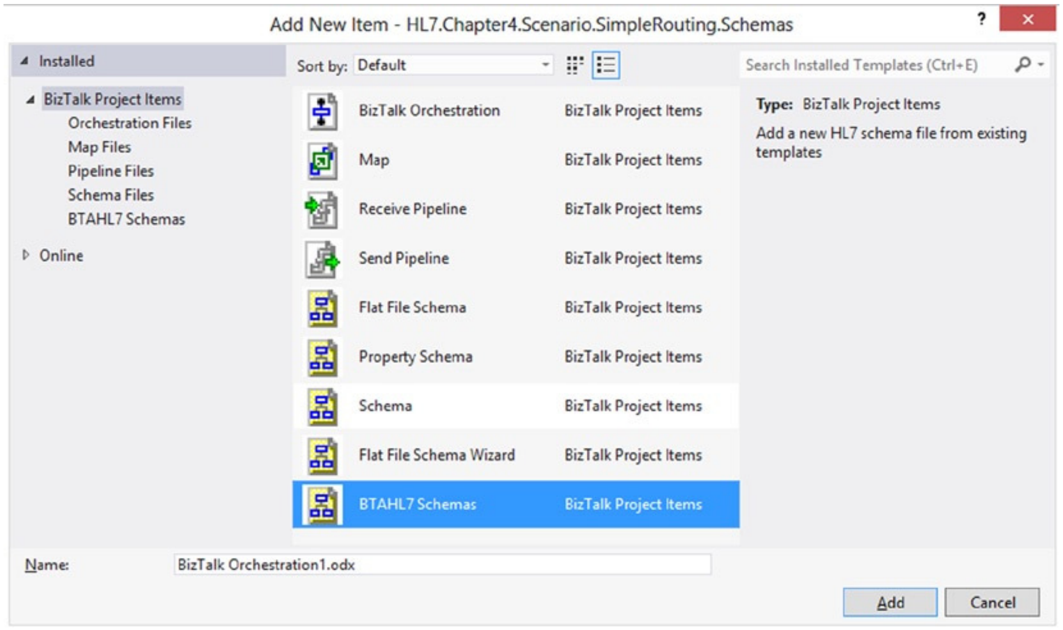
A project that includes common schemas referenced by trigger events schemas for version 2.5.1

Name: HL7.Chapter4.Scenario.SimpleRouting.Schemas

Location: C:\Work\Samples\HL7.Chapter4.Scenario.SimpleRouting

***Figure 4-6.** Segments, DataTypes, TableValues Project Template*

3. The above step will add datatypes_251, tablevalues_251, and segments_251 schemas. Now add a new item to the project via Ctrl+Shift+A and select BTAHL7 Schemas as shown in Figure 4-7.


Add New Item - HL7.Chapter4.Scenario.SimpleRouting.Schemas

Installed
- BizTalk Project Items
  - Orchestration Files
  - Map Files
  - Pipeline Files
  - Schema Files
  - BTAHL7 Schemas
- Online

Sort by: Default

Search Installed Templates (Ctrl+E)

BizTalk Orchestration	BizTalk Project Items
Map	BizTalk Project Items
Receive Pipeline	BizTalk Project Items
Send Pipeline	BizTalk Project Items
Flat File Schema	BizTalk Project Items
Property Schema	BizTalk Project Items
Schema	BizTalk Project Items
Flat File Schema Wizard	BizTalk Project Items
BTAHL7 Schemas	BizTalk Project Items

Type: BizTalk Project Items

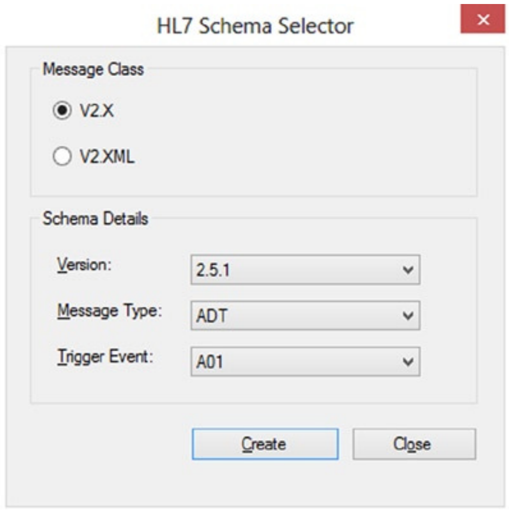
Add a new HL7 schema file from existing templates

Name: BizTalk Orchestration1.odx

Add Cancel

Figure 4-7. *Message Type Schema Template*

4. Click Add, and don't worry about naming the file as next step overrides anything you select here. Once you click Add, you get a new screen called HL7 Schema Selector where you select the message type, triggering event, and version.
5. Select the message class, version, message type, and trigger event as shown in Figure 4-8. Once you click Create, it adds a schema named ADT_A01_251_GLO_DEF.xsd


HL7 Schema Selector

Message Class

● V2.X

○ V2.XML

Schema Details

Version: 2.5.1

Message Type: ADT

Trigger Event: A01

Create Close

Figure 4-8. *HL7 Schema Type Selector*

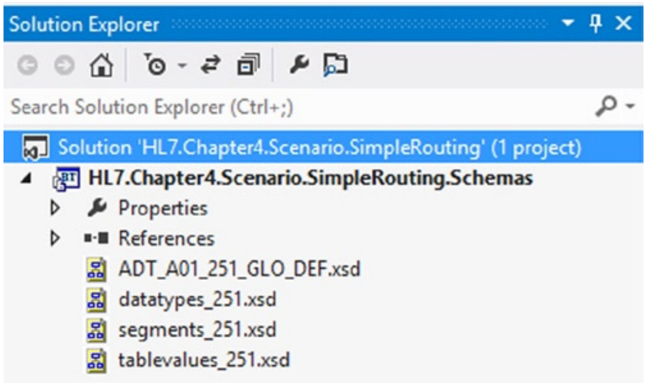
6. By default, the message schema is expected to be added to a separate project because it refers to segments_251 xsd from an assembly reference, as shown:

```
<xs:import schemaLocation="BTAHL7Schemas.segments_251"
namespace="http://microsoft.com/HealthCare/HL7/2X/2.5.1/Segments" />
```

7. Since you added the schema to same project as segments_251 schema, change the schema location to the following:

```
<xs:import schemaLocation="segments_251.xsd"
namespace="http://microsoft.com/HealthCare/HL7/2X/2.5.1/Segments" />
```

8. The project in Solution Explorer will look like Figure 4-9.



```
Solution 'HL7.Chapter4.Scenario.SimpleRouting' (1 project)
  HL7.Chapter4.Scenario.SimpleRouting.Schemas
    Properties
    References
    ADT_A01_251_GLO_DEF.xsd
    datatypes_251.xsd
    segments_251.xsd
    tablevalues_251.xsd
```

Figure 4-9. *Message Schema Project in Solution Explorer*

9. Sign, build, and deploy the project to BizTalk application HL7.Chapter4.Scenarios

This is all that is required to receive an ADT message and route it somewhere else. In this case, you will route it to a folder.

Configurations

Let's create the required receive and send ports to configure the deployed application using the following steps.

1. Add a reference to BizTalk Application 1 application in the BizTalk admin console.
2. Create a two-way MLLP receive port with the following configurations:

Property	Value
Adapter Type	MLLP
Receive Handler	A 32-bit host
Receive Pipeline	BTAHL72XReceivePipeline
Receive Pipeline->Enable Trailing Delimiter	True
Send Pipeline	BTAHL72XSendPipeline
MLLP->Use Direct Synchronous HL7 ACK	True
MLLP->Connection Name	Same as your receive location name
MLLP->Host	127.0.0.1
MLLP->Persistent Connection	True
MLLP->Port	11001 (could be any available port)
MLLP->Receive Time Out	0

3. Create one send port for successfully parsed messages to send them to a FILE folder using following properties:

Property	Value
Filter->BTAHL7Schemas.ParseError	False
Filter->BTS.MessageType	`http://microsoft.com/HealthCare/HL7/2X#ADT_A01_251_GLO_DEF`
Send Pipeline	BTAHL72XSendPipeline
Send Handler	A 32-bit host

This port will send all successfully parsed ADT messages.

4. Create one send port for failed messages with following filters:

Property	Value
Filter->BTAHL7Schemas.ParseError	True
Send Pipeline	Passthrough
Send Handler	Any host, 32-bit or 64-bit

This port will send messages that failed parsing by the HL7 Disassembler.

5. Start the application and the host instances.

Test the Scenario

For testing the scenario, you will use a sample ADT^A01 message (provided below for reference).

```
MSH|^~\&|ADTSource|MCM|ADTDestination|MCM|198808181126|SECURITY|ADT^A01|MSG00002|P|2.5.1|||AL|NE
EVN|A01|198808181123
PID|||PATID1234^5^M11||FN^LN^A^III||19610615|A||2106-3|Address line1^^City^State^Zip|GL|000-000-
0000|000-000-0000~000-000-0000||S||PATID12345001^2^M10|123456789|9-87654^NC
NK1|1|FN^LN^K|SPO|||||20011105
NK1|1|FN^LN^A|FTH
PV1|1|I|2000^2012^01||||004777^LEBAUER^SIDNEY^J.|||SUR||-||1|A0
AL1|1||^PENICILLIN||PRODUCES HIVES~RASH
AL1|2||^CAT DANDER
DG1|001|I9|1550|MAL NEO LIVER, PRIMARY|19880501103005|F||
PR1|2234|M11|111^CODE151|COMMON PROCEDURES|198809081123
ROL|45^RECORDER^ROLE MASTER LIST|AD|CP|FN^LN^ELLEN|199505011201
GT1|1122|1519|BILL^GATES^A
IN1|001|A357|1234|BCMD|||||132987
IN2|ID1551001|SSN12345678
ROL|45^RECORDER^ROLE MASTER LIST|AD|CP|KATE^ELLEN|199505011201
```

There are a couple of ways to send this message to your receive location in BizTalk. The MLLPSend utility provided by the HL7 Accelerator discussed in Chapter 3 can be used as follows:

```
C:\Program Files (x86)\Microsoft BizTalk 2013 Accelerator for HL7\SDK\MLLP Utilities>mllpsend
/IP 127.0.0.1 /P 11001 /TWOWAY /SB 11 /EB 28 /CR 13 /F C:\Work\TestFiles\demo1.hl7
```

With this method, the message acknowledgment returned by the BizTalk connection will be saved in the `demo1.hl7.RESPONSE` file in the `C:\Work\TestFiles` folder and the acknowledgment will look as follows:

```
MSH|^~\&|ADTDestination|MCM|ADTSource|MCM|20140318063809||ACK^A01^ACK|20000GSM|P|2.5.1
MSA|CA|MSG00002
```

If everything is set up correctly, you should get the above acknowledgment and the message should route to the successfully parsed send port. If you stop your send port and notice the context properties of the suspended message, there will be number context properties promoted by the HL7 Disassembler, which can also be used for routing. These properties are defined in the HL7PropertySchema in BizTalk Application 1.

The second method to send HL7 messages to your receive location is a third-party tool referred to as 7Edit. This is a very useful GUI tool as it provides easy mapping of a HL7 message to their segment and fields. You can download a free trial from www.7edit.com/home/index.php.

Key Observations

Some of the key observations we want to highlight while creating and testing these scenarios are as follows:

- There is no need to define a party to accept and parse the message successfully.
- The context properties promoted by the HL7 Disassembler, such as MSH3_1, MSH5_1, and ParseError, are useful for routing.

- The MSA.1 code CA in the acknowledgment message indicates enhanced mode acknowledgment as requested using MSH15. If you change MSH15 to NE and MSH16 to AL, then you will receive an original acknowledgment mode acknowledgment with code AA. You can try changing the values of MSH15 and MSH16 in the message to see different behavior. In some cases like where MSH15 = AL and MSH16 = AL, you will get a suspended message in BizTalk because it will now generate two acknowledgments, one with code AA and another with CA.
- Change the message to make it invalid; for example, change PID8 value to A1 and resend the message. You will see message routing to Failed Message send port with parse error = true. PID8 field has a predefined list of values which you can use, and A1 is not part of that list, so the Disassembler fails in parsing the message. If in your message MSH15 = AL and MSH16 = NE, then the acknowledgment returned will have no error details; however, if you change MSH15 = NE and MSH16 = AL, then the error is returned as part of the acknowledgment.
- Change the message type MSH9 of the message to ADT^A02 and resend the message. You will see that an error is returned that the message schema for ADT A02 is not found and the message is routed to the failed message send port.

■ **Note** In this scenario, the HL7 Accelerator is using default configuration (message body validation, acknowledgment mode).

Turning Off Message Body Validation

In many cases, the message you receive contains a lot of local customized data which does not conform to the HL7 messaging standard, and the requirement is to still accept the message and route it to the destination as required. In such cases, you need to turn off the message body validation.

Build the Solution

As part of implementation, you are going to use exact same Visual Studio solution you used in previous scenarios and expand it as required.

Configuration

In order to turn off message validation, you will perform following steps.

1. Identify the MSH3 value in the message. In your sample ADT message, it is ADTSource. Typically, this is an identifier of the message source.
2. Create a party called ADTSource from the BizTalk Admin Console, as shown in Figures 4-10 and 4-11.

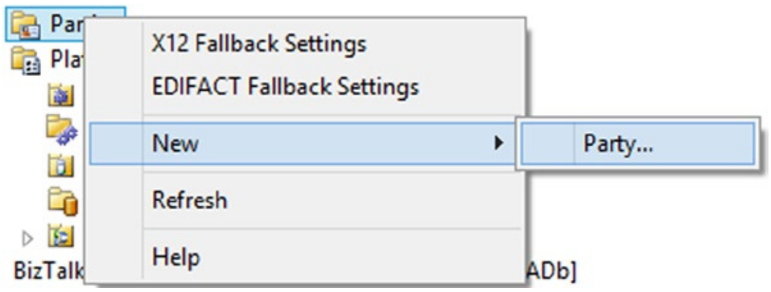
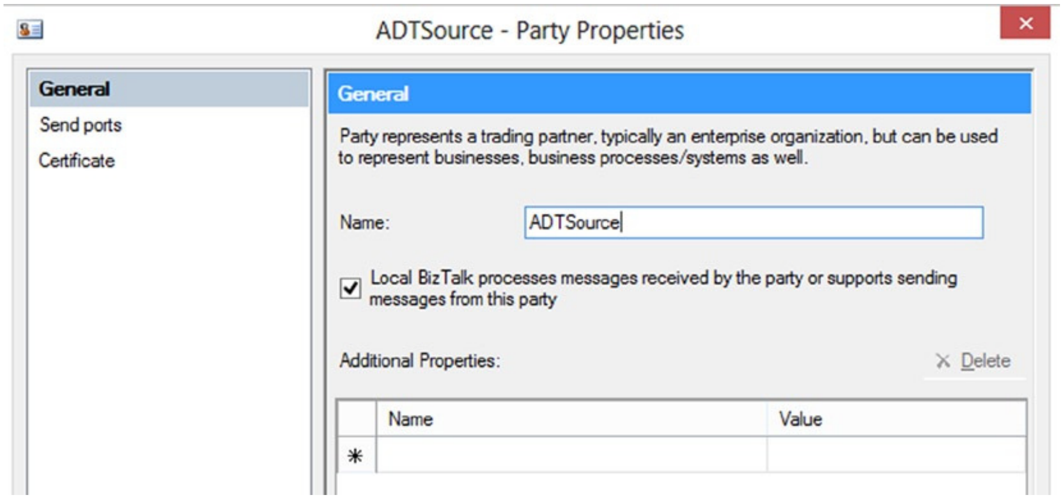


Figure 4-10. *New party creation option*



Party represents a trading partner, typically an enterprise organization, but can be used to represent businesses, business processes/systems as well.

Name: ADTSource

[✓] Local BizTalk processes messages received by the party or supports sending messages from this party

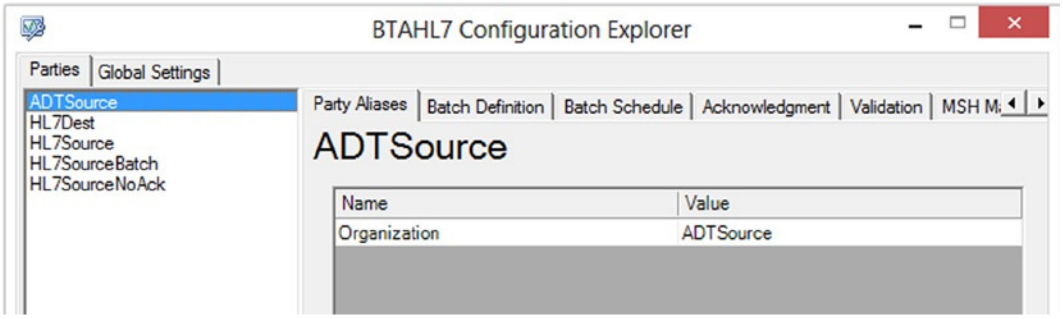
Additional Properties:

	Name	Value
*		

Figure 4-11. *Party creation*

When creating the party, only give it a name and save it; no other configuration is required.

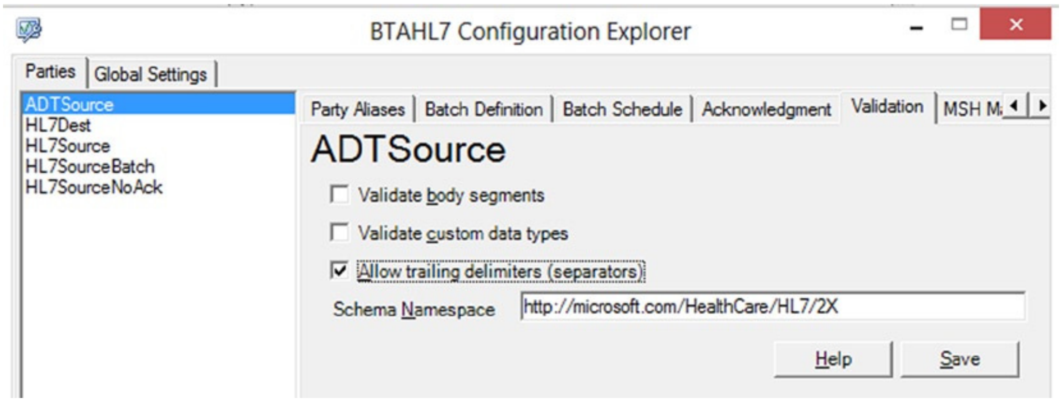
3. Go to BTAHL7 Configuration Explorer, which is installed along with HL7 Accelerator. As you open it, you should see the party ADTSource in left list box along with any other parties available, as shown in Figure 4-12.



Name	Value
Organization	ADTSource

Figure 4-12. *BTAHL7 Configuration Explorer*

4. Go to the Validation tab and uncheck “Validate body segments” and check “Allow trailing delimiters (separators)” as shown in Figure 4-13 and save the party.

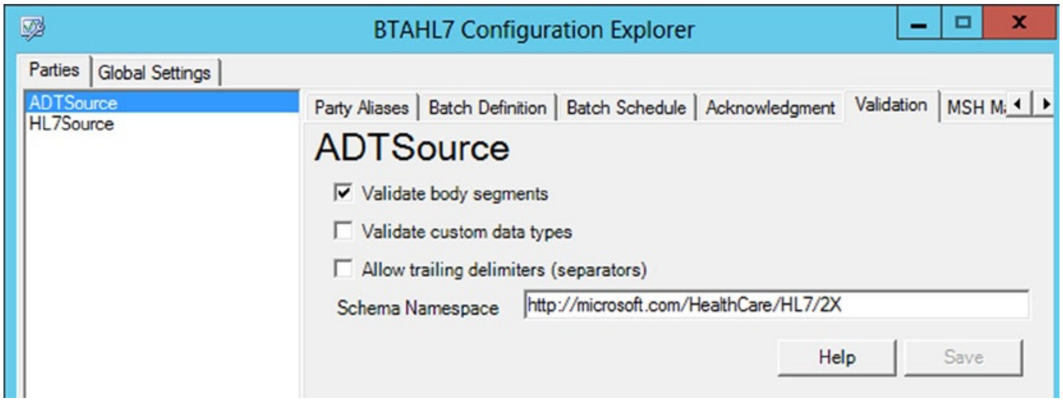


- [ ] Validate body segments
- [ ] Validate custom data types
- [✓] Allow trailing delimiters (separators)

Schema Namespace: http://microsoft.com/HealthCare/HL7/2X

Figure 4-13. *BTAHL7 Configuration Explorer Validation tab*

Note that the “Validate body segments” option is checked by default as shown below:



- [✓] Validate body segments
- [ ] Validate custom data types
- [ ] Allow trailing delimiters (separators)

Schema Namespace: http://microsoft.com/HealthCare/HL7/2X

Figure 4-14. *BTAHL7 Configuration Explorer Validation tab default values*

5. Restart the host on which you have configured your receive location.

Test the Scenario

If you now test the following sample message which has a body validation error in PID 8 (highlighted bold), you will see that no error is raised by the HL7 Disassembler and the message is parsed successfully:

```
MSH|^~\&|ADTSource|MCM|ADTDestination|MCM|198808181126|SECURITY|ADT^A01|MSG00002|P|2.5.1|||NE|AL
EVN|A01|198808181123
PID|||PATID1234^5^M11||FN^LN^A^III||19610615|A1||2106-3|Address line1^^City^State^Zip|GL|000-000-
0000|000-000-0000~000-000-0000||S||PATID12345001^2^M10|123456789|9-87654^NC
NK1|1|FN^LN^K|SPO||||20011105
NK1|1|FN^LN^A|FTH
PV1|1|I|2000^2012^01||||004777^LEBAUER^SIDNEY^J.|||SUR||-||1|A0
AL1|1||^PENICILLIN||PRODUCES HIVES~RASH
AL1|2||^CAT DANDER
DG1|001|I9|1550|MAL NEO LIVER, PRIMARY|19880501103005|F||
PR1|2234|M11|111^CODE151|COMMON PROCEDURES|198809081123
ROL|45^RECORDER^ROLE MASTER LIST|AD|CP|FN^LN^ELLEN|199505011201
GT1|1122|1519|BILL^GATES^A
IN1|001|A357|1234|BCMD|||||132987
IN2|ID1551001|SSN12345678
ROL|45^RECORDER^ROLE MASTER LIST|AD|CP|KATE^ELLEN|199505011201
```

Key Observations

1. There is no development effort for switching off message body validation.
2. If you receive a message from a different source (a different MSH3), you need a new party to be created to turn off the message validation for each of them. This can become a maintenance problem if the number of message sources is too many. You can avoid this by doing further customization using pipeline components, which we will discuss later in this chapter.
3. The host instance restart is always required after making any change to a party configuration.
4. Notice the behavior after turning on the validation again and you will see the same error returned in acknowledgment.

■ **Note** You have seen property called Allow Trailing Delimiter and you specifically checked it in the party configuration and receive pipeline properties. This property means you can accept messages that have a trailing delimiter in the message segment; in above sample, the DG1 segment has trailing delimiter. Notice the behavior if you uncheck it in the party configuration; you will get the acknowledgment with an error on the DG1 segment.

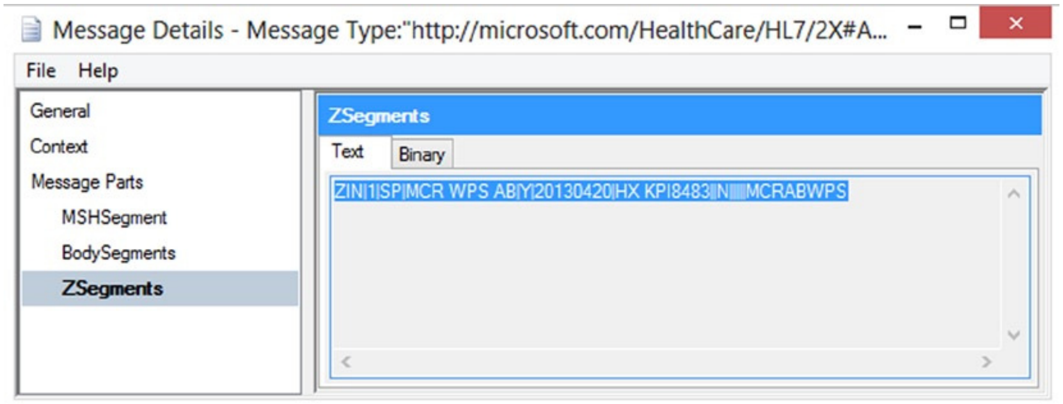
Accept Messages with Customized Fields

There are a couple of scenarios where you need to accept messages with customized fields and as a result you need to modify message schema. These scenarios are listed in the following sections

Z Segment

The HL7 standard allows customization for a message using Z segments. Using these segments you can add fields as per specific system requirements. These Z segments by default are parsed by the HL7 Disassembler as string data type as follows and as shown in Figure 4-15:

```
ZIN|1|SP|MCR WPS AB|Y|20130420|HX KPI8483||N||||MCRABWPS
```



```
ZIN|1|SP|MCR WPS AB|Y|20130420|HX KPI8483||N||||MCRABWPS
```

Figure 4-15. *Z Segments data in message*

If you add the above ZIN segment to your ADT sample and send the message, observe the Z Segments of parsed message on a suspended message instance by stopping the send port.

In many cases, these ZSegments are required to be part of BodySegments so that they can be used in BizTalk maps for transformation. In such cases, you need to change your ADT schema to include Z segment definition.

Standard Segment Field Customization

In addition to the Z segment addition to a schema, many times the source system generates extra segment fields in the message. These extra segment fields, which are not defined in HL7 standard schemas, cause the HL7 Disassembler to fail parsing even if body validation is turned off. This is a very common problem when parsing HL7 messages in BizTalk. One example of such message is the following:

```
MSH|^~\&|ADTSource|MCM|ADTDestination|MCM|198808181126|SECURITY|ADT^A01|MSG00002|P|2.5.1|||NE|AL
EVN|A01|198808181123
PID|||PATID1234^5^M11||FN^LN^A^III||19610615|A||2106-3|Address line1^^City^State^Zip|GL|000-000-
0000|000-000-0000~000-000-0000||S||PATID12345001^2^M10|123456789|9-87654^NC
NK1|1|FN^LN^K|SPO||||20011105
NK1|1|FN^LN^A|FTH
PV1|1|I|2000^2012^01||||004777^LEBAUER^SIDNEY^J.|||SUR||-||1|A0
AL1|1||^PENICILLIN||PRODUCES HIVES~RASH
AL1|2||^CAT DANDER
DG1|001|I9|1550|MAL NEO LIVER, PRIMARY|19880501103005|F||
PR1|2234|M11|111^CODE151|COMMON PROCEDURES|198809081123
ROL|45^RECORDER^ROLE MASTER LIST|AD|CP|FN^LN^ELLEN|199505011201
GT1|1122|1519|BILL^GATES^A
IN1|001|A357|1234|BCMD|||12345^11123|132987
```

```
IN2|ID1551001|SSN12345678
ROL|45^RECORDER^ROLE MASTER LIST|AD|CP|KATE^ELLEN|199505011201
ZIN|1|SP|MCR WPS AB|Y|20130420|HX KPI8483||N||||MCRABWPS
```

In this example, IN1.8 field highlighted in bold causes parsing to fail with the following error:

```
MSH|^~\&|ADTDestination|MCM|ADTSource|MCM|20140320202627||ACK^A01^ACK|0000GSM2|P|2.5.1|||NE
MSA|AE|MSG00002
ERR||IN1^1^8|102^Data type error^HL79999|E|||||||^^^^^^^^^^^
```

The error exactly indicates which field of the segment has the issue. Such errors cannot be resolved even by turning off body validation. The recommended approach to solve such an issue is to ask the message source system to fix this; however, this is not always possible. In cases where you need to accept the message with extra fields, you need to modify the message schema, as you will see shortly.

Build the Solution

In order to implement both scenarios, you will modify ADT message schema using following steps.

1. Add a new BTAHL7V251Common Project to your previous scenario solution and name it HL7.Chapter4.Scenarios.CustomizedSchemas.
2. Add the ADT A01 schema to the project as you did in the previous scenario.
3. In order to customize the schemas, you need to change the default target namespace of all the schemas you added. The quickest way is to find `http://microsoft.com/HealthCare/HL7/2X` and replace it with a new namespace like `http://hl7.chapter4.scenario.customization/schemas/hl7/2x`.
4. After the change, you should be able to open the ADT schema without any error.
5. Now that you can modify your schemas as necessary, let's look for IN1 segment in segments_251.xsd and go to field IN1_8_GroupNumber. Note that this is a simple type accepting strings. To change it as per your sample message, you need to add a new complex type in the datatypes_251.xsd schema.

   ```
   <xs:element name="CustomIN1_8" type="CustomIN1_8" />
   <xs:complexType name="CustomIN1_8"><xs:sequence><xs:element name="GroupNumber" type="ST" />
   <xs:element name="GroupNumber_1" type="ST" /></xs:sequence></xs:complexType>
   ```

6. Let’s go back to segments_251.xsd to change IN1_8_GroupNumber. Since this field is a simple type, you need to delete it and replace it with a Record type and change its data structure type to `ns0:CustomIN1_8.`
7. Now let’s add a ZIN segment to segments_251.xsd as follows:

   ```
   <xs:element name="ZIN" type="ZIN" />
   <xs:complexType name="ZIN">
       <xs:sequence>
         <xs:element name="Field1" type="ns0:ST" />
         <xs:element name="Field2" type="ns0:ST" />
         <xs:element name="Field3" type="ns0:ST" />
         <xs:element name="Field4" type="ns0:ST" />
         <xs:element name="Field5" type="ns0:ST" />
   ```

```
            <xs:element name="Field6" type="ns0:ST" />
            <xs:element name="Field7" type="ns0:ST" />
            <xs:element name="Field8" type="ns0:ST" />
            <xs:element name="Field9" type="ns0:ST" />
            <xs:element name="Field10" type="ns0:ST" />
            <xs:element name="Field11" type="ns0:ST" />
            <xs:element name="Field12" type="ns0:ST" />
            <xs:element name="Field13" type="ns0:ST" />
            <xs:element name="Field14" type="ns0:ST" />
          </xs:sequence>
        </xs:complexType>
```

8. Now add the newly defined ZIN segment to the ADT A01 schemas by adding a child record of data structure type `ns0:ZIN`.

9. Your modifications to the schemas are complete. Now deploy the application to the HL7. Chapter4.Scenarios.CustomizedSchemas BizTalk application.

■ **Note** Changing the default target namespace of the schemas allows you to deploy both your original message and modified schemas. Remember, you cannot modify the namespace for MSH and ACK message types.

Configuration

Receive and Send Ports

You will use your first scenario's receive and send port configurations to test this scenario, except one additional send port is required to receive your modified schema. You will create this send port in application HL7.Chapter4.Scenarios itself to keep all send ports together. The send port properties are as follows:

Property	Value
Filter->BTAHL7Schemas.ParseError	False
Filter->BTS.MessageType	`http://hl7.chapter4.scenario.customization/schemas/hl7/2x` #ADT_A01_251_GLO_DEF
Send Pipeline	BTAHL72XSendPipeline
Send Handler	A 32-bit host

Party

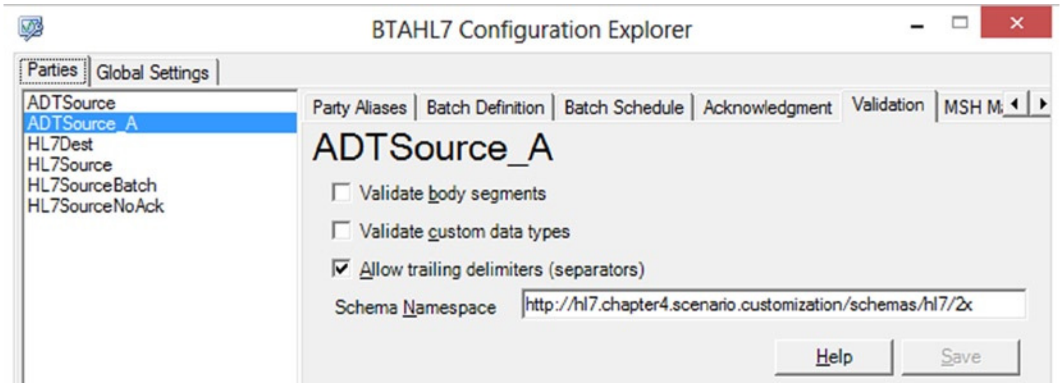
You will now create a new party named ADTSource_A. On the validation tab, make the following changes:

1. Change the schema namespace to `http://hl7.chapter4.scenario.customization/schemas/hl7/2x`.

2. Uncheck body validation.

3. Allow trailing delimiters.

The party configuration should look like Figure 4-16.



ADTSource_A

- [ ] Validate body segments
- [ ] Validate custom data types
- [✓] Allow trailing delimiters (separators)

Schema Namespace: http://hl7.chapter4.scenario.customization/schemas/hl7/2x

Figure 4-16. Validation tab: BTAHL7 Configuration Explorer

Save the party and restart the host instances.

■ **Note** The only reason you created a new party is to keep both scenarios working and to showcase the difference between the two. You can always modify your existing party to change the namespace.

Test the Scenario

In your sample message, change MSH3 to ADTSource_A to match the party name and test the message. Note now that the message has been parsed successfully with the body having a ZIN segment and IN1_8 having two fields.

Key Observations

1. Both message types with original schemas and modified schemas can coexist by changing the target namespace.
2. The same receive port can accept both messages using different MSH3 (if you change MSH3 to ADTSource, the message default schema will be used by the Disassembler; if it is ADTSource_A, then a customized schema will be used).

Customized Acknowledgment Behavior

As of now, you have used the default behavior of the HL7 Accelerator for acknowledgment, which means the acknowledgment is generated based on the MSH15 and MSH16 value in the message. In many cases, the HL7 message does not have both MSH15 and MSH16 values defined, and the sending application still wants an acknowledgment to be returned. In such cases, with the default behavior on the two-way receive port, no acknowledgment is generated by the HL7 Accelerator, and the two-way receive port instance is kept waiting for the acknowledgment.

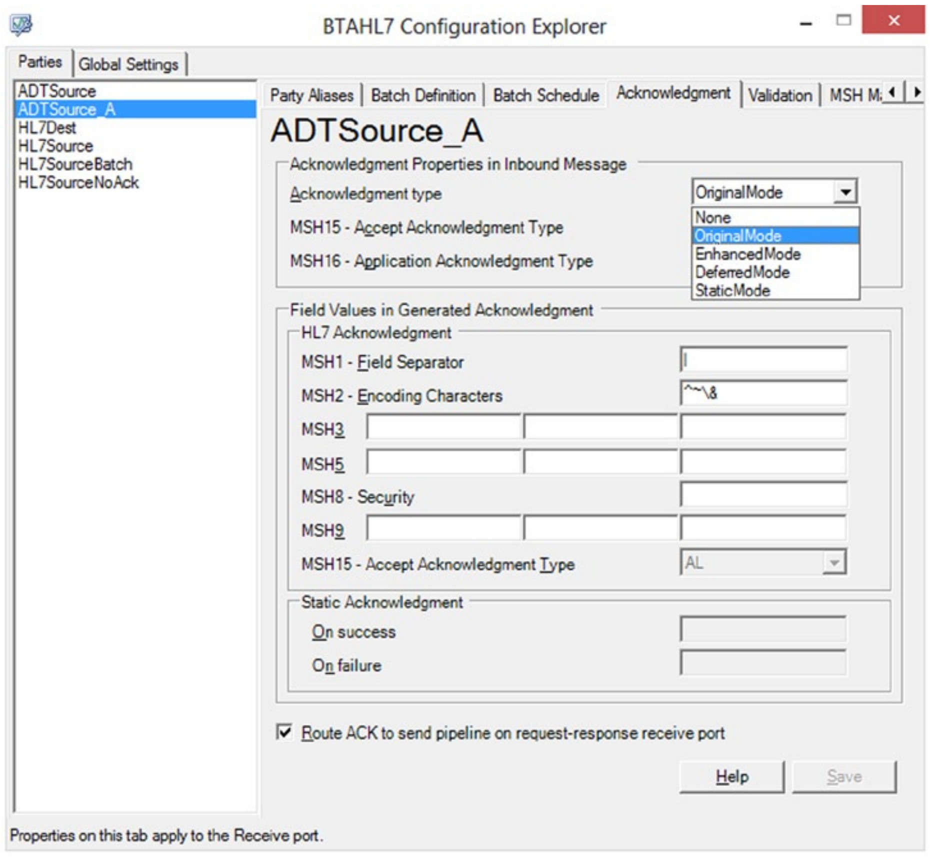
Ideally, the sending application should be sending MSH15 and MSH16, requesting acknowledgment; however at times messages don't have these field values. The HL7 Accelerator allows you to configure an acknowledgment mode in such cases to generate the acknowledgment when MSH15 and MSH16 are not supplied in message.

Build the Solution

There is no change to the solution you have built so far.

Configurations

All you need to do to make this work is change the acknowledgment mode in the party configuration from "None" to one of the modes you want, as shown in Figure 4-17.



ADTSource_A

Acknowledgment Properties in Inbound Message

Acknowledgment type	OriginalMode
MSH15 - Accept Acknowledgment Type	
MSH16 - Application Acknowledgment Type	

None
OriginalMode
EnhancedMode
DeferredMode
StaticMode

Field Values in Generated Acknowledgment

HL7 Acknowledgment

MSH1 - Field Separator	|
MSH2 - Encoding Characters	^~\&
MSH3	
MSH5	
MSH8 - Security	
MSH9	
MSH15 - Accept Acknowledgment Type	AL

Static Acknowledgment

On success	
On failure	

☑ Route ACK to send pipeline on request-response receive port

Properties on this tab apply to the Receive port.

Figure 4-17. *Acknowledgment tab: BTAHL7 Configuration Explorer*

If you use EnhancedMode, you can also override MSH15 and MSH16 depending on the requirement. You can also override other MSH segment fields in the generated acknowledgment for modes other than StaticMode.

■ **Note** All different acknowledgment modes defined on acknowledgment tab are only used when MSH15 and MSH16 values are not present in message—except for StaticMode.

Test the Scenario

If you remove the MSH15 and MSH16 values from the sample message and test it, you will see the exact behavior configured on the party configurator.

Key Observations

1. Use a different acknowledgment mode and notice the generated acknowledgment. The acknowledgment is generated as per the configuration.
2. Notice the behavior when MSH15 and MSH16 are also provided in the message and the party configuration has an acknowledgment mode defined. In this case, the acknowledgment mode provided within the party configuration is ignored unless the acknowledgment is static.

Message Transformation

Message transformation is one of the key features of BizTalk. Message transformation for a HL7 message is same as any other xml message transformation except that transformation requires an orchestration to be created because a HL7 message is disassembled into a multipart message. This is a limitation of all multipart messages because a map cannot be created and used without orchestration for such messages. In this scenario, you will learn how to consume a HL7 multipart message in orchestration and use a map to transform it.

For this scenario, you will transform the incoming message ADT^A01 on message type `http://microsoft.com/HealthCare/HL7/2X#ADT_A01_251_GLO_DEF` to ADT^A01 message of message type `http://hl7.chapter4.scenario.customization/schemas/hl7/2x#ADT_A01_251_GLO_DEF`.

Build the Solution

You will use the same solution you have been using for other scenarios to build the maps and orchestration using following steps.

1. Add a new project called HL7.Chapter4.Scenarios.Orchestrations to the solution you built and add a reference to the HL7.Common.Schemas assembly created in the first scenario.
2. Add another reference to the schema project that contains the ADT^A01 message. You also need to add a reference to the HL7 Accelerator property schema assembly called Microsoft.Solutions.BTAHL7.HL7Schemas to add filters on the receive shape.
3. Add a new multipart message type named HL7MessageType from the orchestration view with three parts to it as follows:

Message Parts	Body Part	Type
Part 1-> MSHSegment	False	MSH_25_GLO_DEF, schema type referenced from HL7.Common.Schemas assembly.
Part2->BodySegments	True	ADT_A01_251_GLO_DEF, schema type referenced from HL7.Chapter4.Scenarios.Schemas project
Part3->Z Segments	False	System.String

4. Add another multipart message type named HL7MessageOutType from the orchestration view with three parts to it as follows:

Message Parts	Body Part	Type
Part 1-> MSHSegment	False	MSH_25_GLO_DEF, schema type referenced from HL7.Common.Schemas assembly.
Part2->BodySegments	True	ADT_A01_251_GLO_DEF, schema type referenced from HL7.Chapter4.Scenarios.CustomizedSchemas project
Part3->Z Segments	False	System.String

5. Now create the input message variable named hl7MesageIn of a multipart message type and use this message to receive an ADT message from the message box with a filter on following filters:

```
BTS.MessageType = "http://microsoft.com/HealthCare/HL7/2X#ADT_A01_251_GLO_DEF" And
BTAHL7Schemas.ParseError = false
```

6. Create another message variable for output hl7MessageOut of the second multipart message type HL7MessageOutType.

7. In order to create the transformation, you also need to create two message variables of your source and destination body schema types (HL7.Chapter4.Scenarios.Schemas.ADT_A01_251_GLO_DEF and HL7.Chapter4.Scenarios.CustomizedSchemas.ADT_A01_251_GLO_DEF).

8. Create the transformation using the two message variables you created in step 7 and use the mass copy function to map everything from source to destination, as shown in Figure 4-18.

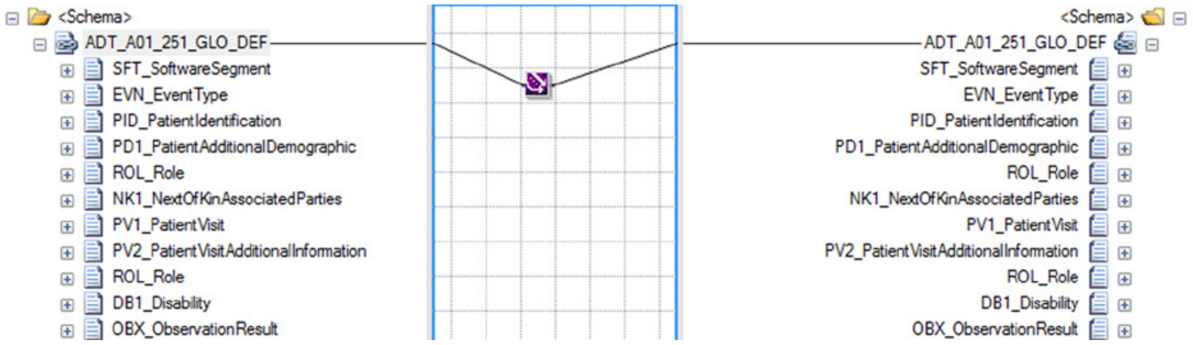


Figure 4-18. *Map*

9. In the construct shape, use the transformation to map the body and use the message assignment shape to assign the other message parts.

First assignment shape:

```
bodySegmentIn = hl7MessageIn.BodySegments;

Second assignment shape:

hl7MessageOut.BodySegments = bodySegmentOut;
hl7MessageOut.Z Segments = hl7MessageIn.Z Segments;
hl7MessageOut.MSHSegment = hl7MessageIn.MSHSegment;
hl7MessageOut(*) = hl7MessageIn(*);
```

10. Send the output message, and name the logical send port operation to SendTransformedMessage.
11. The orchestration will look like Figure 4-19.

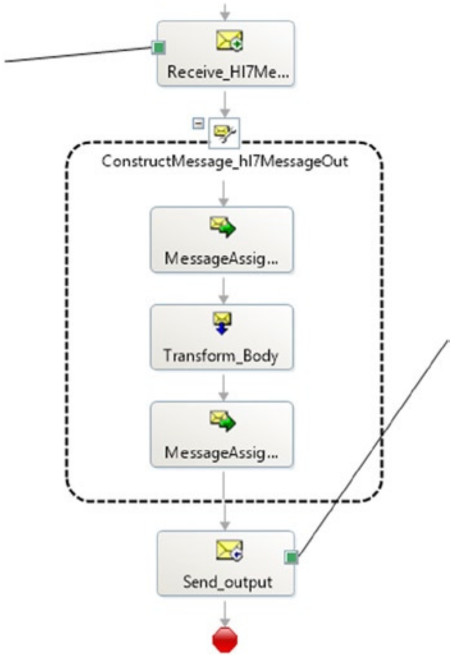


Figure 4-19. *Orchestration sample*

12. Build and deploy the solution to the HL7.Chapter4.Scenarios application.

Configurations

The configuration will be same for other scenarios except you will add another send port called Send_OrchestrationOutput to subscribe to the message output of orchestration using the `BTS.Operation == SendTransformedMessage` filter.

Test the scenario

You will be using the sample ADT message that you have been using for all scenarios with MSH3=ADTSource. For quick reference, the message is provided below:

```
MSH|^~\&|ADTSource|MCM|ADTDestination|MCM|198808181126|SECURITY|ADT^A01|MSG00002|P|2.5.1|||AL|NE
EVN|A01|198808181123
PID|||PATID1234^5^M11||FN^LN^A^III||19610615|A||2106-3|Address line1^^City^State^Zip|GL|000-000-
0000|000-000-0000~000-000-0000||S||PATID12345001^2^M10|123456789|9-87654^NC
NK1|1|FN^LN^K|SPO||||20011105
NK1|1|FN^LN^A|FTH
PV1|1|I|2000^2012^01||||004777^LEBAUER^SIDNEY^J.|||SUR||-||1|A0
AL1|1||^PENICILLIN||PRODUCES HIVES~RASH
AL1|2||^CAT DANDER
DG1|001|I9|1550|MAL NEO LIVER, PRIMARY|19880501103005|F||
PR1|2234|M11|111^CODE151|COMMON PROCEDURES|198809081123
ROL|45^RECORDER^ROLE MASTER LIST|AD|CP|FN^LN^ELLEN|199505011201
GT1|1122|1519|BILL^GATES^A
IN1|001|A357|1234|BCMD|||12345|132987
IN2|ID1551001|SSN12345678
ROL|45^RECORDER^ROLE MASTER LIST|AD|CP|KATE^ELLEN|199505011201
ZIN|1|SP|MCR WPS AB|Y|20130420|HX KPI8483||N||||MCRABWPS
```

In testing, you will see now that the message is sent to two send ports: the original message output and the transformed message output. You can cross-check the output by stopping your send ports and checking the suspended send ports message types.

Key Observations

1. Implementation of orchestration and transformation of the HL7 multipart message is simple, except the transformation can be complex due to the number of fields in the message and depending on the requirements.
2. It's not possible to use maps on the receive or send ports directly; this limitation is due to the multipart message and requires orchestration to be built.

Send an HL7 Message via a MLLP Adapter

So far in all the scenarios, you have been using the FILE send adapter to send messages; however, in a real scenario, you will mostly be using the MLLP adapter to send HL7 messages. In this scenario, you will learn how to configure both one- and two-way MLLP send adapters.

Build the Solution

You will use your existing scenario solution. No change is required.

Configuration

You will configure two send ports to configure the following two options:

- Create a one-way send port and receive acknowledgment on different receive locations.

Property	Value
Adapter Type	MLLP
Send Handler	A 32-bit host
Send Pipeline	BTAHL72XSendPipeline
Send Pipeline->Enable Trailing Delimiter	True
MLLP->Acceptable Ack Codes	All
MLLP->Connection Name	Same as your send port name (could be any string identifying the connection in performance counter)
MLLP->Host	127.0.0.1
MLLP->Persistent Connection	True
MLLP->Port	21110 (could be any available port)
MLLP->Send Time Out	0
MLLP->Solicit Response Enabled	Yes
MLLP->Submit Receive Location URI for ACK	127.0.0.1:65535 (This receive location is created in BizTalk Application 1 after the HL7 Accelerator installation. You can create any other one-way receive location on a different port to receive the ack; the key thing is that the host must be same for both send port and receive location.)
Filters	`BTAHL7Schemas.ParseError == false And BTS.MessageType == http://microsoft.com/HealthCare/HL7/2X#ADT_A01_251_GLO_DEF`

All other configurations should be left default.

- Create one two-way send port to receive the acknowledgment.

Property	Value
Adapter Type	MLLP
Send Handler	A 32-bit host
Send Pipeline	BTAHL72XSendPipeline
Send Pipeline->Enable Trailing Delimiter	True
Receive Pipeline	BTAHL72XReceivePipeline
Receive Pipeline->Enable Trailing Delimiter	True
MLLP->Acceptable Ack Codes	All
MLLP->Connection Name	Same as your send port name (could be any string identifying the connection in performance counter)
MLLP->Host	127.0.0.1
MLLP->Persistent Connection	True
MLLP->Port	21111 (could be any available port)
MLLP->Send Time Out	0
Filters	`BTAHL7Schemas.ParseError == false  And BTS.MessageType == http://microsoft.com/HealthCare/HL7/2X#ADT_A01_251_GLO_DEF`

All other configurations should be left default.

- Now create a listener on port 21111 and 21110 to receive the message and send the acknowledgment. Listeners can be created either using the MLLPReceive utility provided by the HL7 Accelerator or using the 7-Edit tool. Let's create it using MLLPReceive using following command on the command prompt:

```
mllpreceive /I 127.0.0.1 /P 21110 /SB 11 /EB 28 /CR 13 /HL7ACK C:\Work\TestFiles\Acks\
ack.txt /D C:\Work\TestFiles\MLLPReceivedFiles\21110
```

  and

```
mllpreceive /I 127.0.0.1 /P 21110 /SB 11 /EB 28 /CR 13 /HL7ACK C:\Work\TestFiles\Acks\
ack.txt /D C:\Work\TestFiles\MLLPReceivedFiles\21111
```

  The following is the acknowledgment message sent to BizTalk:

```
MSH|^~\&|TEST2|MCM|TEST|MCM|20131226132358||ACK|10000GSM|P|2.5.1|||NEMSA|AE|MSG00001ERR||
PID^1^5|102^Data type error^HL79999|E|||||||^^^^^^^^^^^ ERR||NK1^1^2|102^Data type error
^HL79999|E|||||||^^^^^^^^^^^
```

- You also need to create one send port to subscribe to the acknowledgment sent by MLLReceive to BizTalk. The send port you create is a FILE adapter with a filter on `BTS.MessageType == http://microsoft.com/HealthCare/HL7/2X#ACK_25_GLO_DEF`

Test the Scenario

Test the scenario using your sample ADT^A01 message. You will see that two messages are stored in each directory, `C:\Work\TestFiles\MLLPReceivedFiles\21110` and `C:\Work\TestFiles\MLLPReceivedFiles\21111`. Also, BizTalk receives the acknowledgment and sends it to send port you created in your last step.

Key Observations

1. Notice the use of a separate receive location in the send port to receive the acknowledgment one way send port.
2. Notice how even the acknowledgment with the error reached the acknowledgment send port. If you change Accept Acknowledgment codes to AA or CA, then BizTalk will suspend the original message if negative acknowledgment is received. You can test this behavior by making a change to the send port configurations.

Batching

In this scenario, you will learn how to batch HL7 messages before sending them. Let's say you want to batch all of messages you use to test these scenarios before sending them out. You'll configure a batch size of five message counts (when five messages are received, a batch message will be created and sent out).

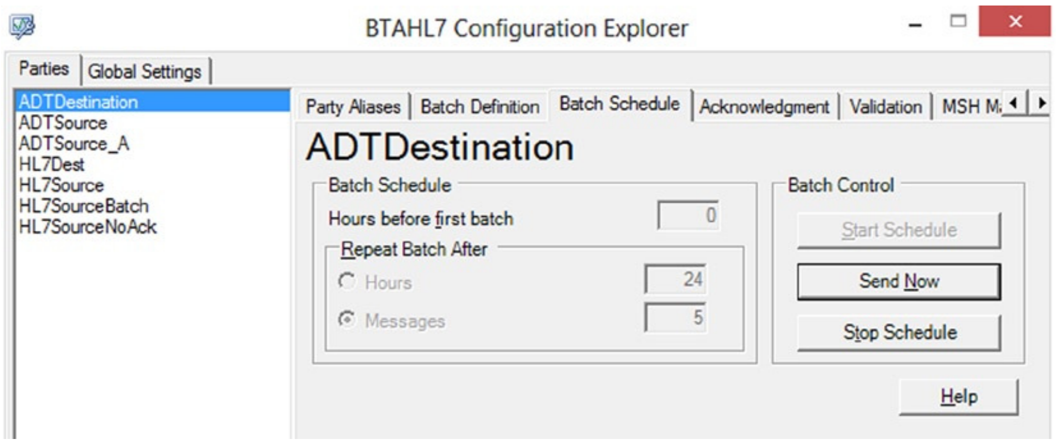
Build the Solution

There is no change required to your solution to implement batching.

Configurations

You need the following configurations to implement batching.

1. So far in your scenarios, you have not configured party for destination application (in other words, based upon a MSH5 value, batching requires you to set this up). In your message, you have been using ADTDestination as MSH5, so create a party called ADTDestination, and on the Batch Schedule tab, select Message count as 5 under the Repeat Batch After section, and click the Start Schedule button. The batch schedule should look like Figure 4-20.


BTAHL7 Configuration Explorer

Parties | Global Settings

ADTDestination
ADTSource
ADTSource_A
HL7Dest
HL7Source
HL7SourceBatch
HL7SourceNoAck

Party Aliases | Batch Definition | Batch Schedule | Acknowledgment | Validation | MSH M

ADTDestination

Batch Schedule

Hours before first batch: 0

Repeat Batch After

Hours: 24

[✓ Messages]: 5

Batch Control

Start Schedule

Send Now

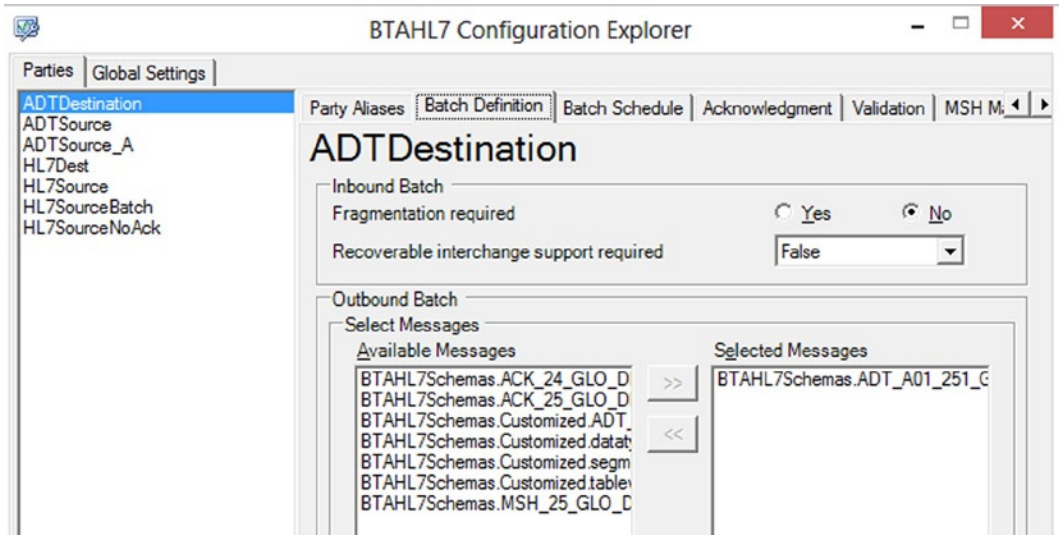
Stop Schedule

Help

Figure 4-20. *Batch schedule: BTAHL7 Configuration Explorer*

2. Make sure you have enabled the BatchControlLocation receive location and started batch orchestration in BizTalk Application 1. The receive location and orchestration is created by the HL7 Accelerator installation. After the start schedule, you should have your Batch orchestration activated and ready to receive the messages you want to batch.

3. On the Batch Definition tab of the ADTDestination party, select the ADT message schema type, as shown in Figure 4-21.


BTAHL7 Configuration Explorer

Parties | Global Settings

ADTDestination
ADTSource
ADTSource_A
HL7Dest
HL7Source
HL7SourceBatch
HL7SourceNoAck

Party Aliases | Batch Definition | Batch Schedule | Acknowledgment | Validation | MSH M

ADTDestination

Inbound Batch

Fragmentation required: Yes [✓ No]

Recoverable interchange support required: False

Outbound Batch

Select Messages

Available Messages:
BTAHL7Schemas.ACK_24_GLO_D
BTAHL7Schemas.ACK_25_GLO_D
BTAHL7Schemas.Customized.ADT_
BTAHL7Schemas.Customized.datat
BTAHL7Schemas.Customized.segm
BTAHL7Schemas.Customized.table
BTAHL7Schemas.MSH_25_GLO_D

>> <<

Selected Messages:
BTAHL7Schemas.ADT_A01_251_G

Figure 4-21. *Batch definition: BTAHL7 Configuration Explorer*

4. On the Validation tab, also uncheck the Validate body segment box and check the Allow Trailing Delimiter box.
5. Create a send port to subscribe the batched message using filter `BTAHL7Schemas.BTAHL7MessageType == OutboundBatch`.
6. Unlist all the other send ports and orchestration you have created in your scenarios so far.
7. Restart the hosts.

Test the Scenario

For testing you will use same message you have been using and you will submit the same message five times. In order to track each of them, you can change the MSH10 message control id every time you send the message. Once you have sent the five messages, one batched message will be sent out to your send port. The output folder will have a batched message containing all five messages. The message output will have FHS, BHS headers and BTS, FTS trailer segments as required by HL7 specifications.

Key Observations

1. In order to implement batching, you were not required to change your solution for this simple scenario. You achieved batching with configurations only.
2. Batching configuration was done only on the destination party (the MSH5 value).

Summary

In this chapter, you implemented basic scenarios to showcase BizTalk implementations using the HL7 Accelerator. These scenarios allow you to understand more about the HL7 Accelerator and are key to implementing any real-world scenarios, as you will see in Chapter 5.

CHAPTER 5

HL7 Advanced Topics

In the previous chapter, you were provided with several scenarios regarding the use of the HL7 Accelerator and receiving HL7 Version 2.x messages. In this chapter, you are going to create and send these messages. Specifically, you will learn how to do the following:

- Map Version 2.x to the HL7 Version 3 CDA
- Meaningful Use for Immunizations
- Use HL7 V.2.5.1 queries

Mapping Version 2.x to the HL7 CDA

CDA stand for HL7 Clinical Data Architecture. It is the standard XML format for clinical documents. The CDA is the HL7 Version 3 Standard. If you are not familiar with HL7 CDA, you can find more information at the HL7 web site (`www.hl7.org/implement/standards/product_brief.cfm?product_id=186`).

Note The HL7 Version 3 CDA documentation uses the name "Component" for each element and attribute. Since we deal with schemas and XML documents in BizTalk, we will use "element" or "attribute" throughout this topic.

The Steps

There are three steps you will use in your conversion. Figure 5-1 shows these steps.

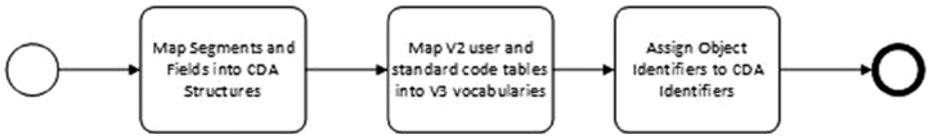

Map Segments and Fields into CDA Structures
Map V2 user and standard code tables into V3 vocabularies
Assign Object Identifiers to CDA Identifiers

Figure 5-1. *Three steps*

Before starting on your mapping, let's take a look at a code example showing the minimum elements for the HL7 Version 3 CDA; see Listing 5-1. Table 5-1 shows the key that will make it easier for you to understand the structure used throughout this topic.

Table 5-1. *Key for Listing 5-1*

<required
Fixed value
Optional
variableValue = xxxx

Listing 5-1. Minimum CDA Elements

```
<ClinicalDocument xmlns="urn:hl7-org:v3" xmlns:mif="urn:hl7-org:v3/mif"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:hl7-org:v3 CDA.xsd">
<typeId root="2.16.840.1.113883.1.3" extension="POCD_HD000040"/>

<id root="xxxx" extension="xxxx"/>

<code code="xxxx" codeSystem="xxxx" codeSystemName="xxxx" displayName="xxxx"/>

<effectiveTime value="xxxx"/>

<confidentialityCode code="x" codeSystem="xxxx"/>

<recordTarget>
  <patientRole>
    <id extension="xxxx" root="xxxx"/>
    <patient>
      <name>
        <given>xxxx</given>
        <family>xxxx</family>
        <suffix>xxxx</suffix>
      </name>
      <administrativeGenderCode code="x" codeSystem="xxxx"/>
      <birthTime value="xxxx"/>
    </patient>
    <providerOrganization>
      <id root="xxxx"/>
    </providerOrganization>
  </patientRole>
</recordTarget>

<author>
  <time value="xxxx"/>
  <assignedAuthor>
    <id extension="xxxx" root="xxxx"/>
    <assignedPerson>
      <name>
        <given>xxxx</given>
        <family>xxxx</family>
        <suffix>xxxx</suffix>
      </name>
    </assignedPerson>
    <representedOrganization>
      <id root="xxxx"/>
    </representedOrganization>
  </assignedAuthor>
</author>

<custodian>
  <assignedCustodian>
    <representedCustodianOrganization>
      <id root="xxxx"/>
    </representedCustodianOrganization>
  </assignedCustodian>
</custodian>

<documentationOf>
  <serviceEvent classCode="xxxx">
   <code code="xxx" codeSystem="xxx" codeSystemName="xxx" displayName="xxx"/>
   <effectiveTime>
     <low value="xxxx"/>
     <high value="xxxx"/>
   </effectiveTime>
   <performer typeCode="xxxx">
     <functionCode code="xxxx" codeSystem="xxxx"/>
     <time>
       <low value="xxxx"/>
       <high value="xxxx"/>
     </time>
     <assignedEntity>
       <id extension="xxxx" root="xxxx"/>
         <code code="xxxx" codeSystem="xxxx"
            codeSystemName="xxxx" displayName="xxxx"/>
          <addr>
            <streetAddressLine>xxxx</streetAddressLine>
            <city>xxxx</city>
            <state>xxxx</state>
            <postalCode>xxxx</postalCode>
            <country>xxxx</country>
          </addr>
          <telecom value="xxxx" use="xxxx"/>
          <assignedPerson>
            <name>
              <prefix>xxxx</prefix>
              <given>xxxx</given>
              <family>xxxx</family>
              <suffix>xxxx</suffix>
            </name>
          </assignedPerson>
        </assignedEntity>
      </performer>
    </serviceEvent>
  </documentationOf>
```

nonXMLBody:

```
<component>
  <nonXMLBody>
    <text mediaType="xxxx/xxxx">
        <reference value="xxxx.xxx"/>
    </text>
  </nonXMLBody>
</component>
```

structuredBody:

```
<component>
  <structuredBody>
    <section>
      <code code="xxxx" codeSystem="xxxx" codeSystemName="xxxx" displayName="xxxx"/>
      <title>xxxx</title>
      <text>
       xxxx
<content styleCode="xxxx">xxxx</content>
      xxxx</text>
    </section>
  </structuredBody>
</component>
```

■ **Note** For most of this topic we will use HL7 Version 2.5.1.

Common Message Types

Let's look at a few message types that are common to both the Version 2.x Standard and the Version 3 Standard.

ORU - Unsolicited Transmission of an Observation Message

OBX Segments that appear in an ORU message can use both the single form found in MDM messages and the name/value as in ADT messages. These are typically lab reports in a tabular format or diagnostic studies. A Narrative element can also be included.

ORU messages allow for multiple OBX Segments that will be associated with a single OBR segment. In these cases, each OBR can represent a single section of the CDA.

MDM - Medical Document Management

In MDM messages, a single OBX Segment often contains the entire document contents. The contents of this segment can be plain, rich text, or any other format.

- New lines are often represented by using repeated OBX-5 components. They can also be represented by a hexadecimal escape sequence, such as `\XOD\` or `\XOA\` or sometimes both.
- Line breaks can be inserted to support a repeat character, such as a tilde (~).

- For formats other than plain text, the OBX-2 field will often use the HL7 Version 2 ED data type.
- This data type, like the Version 3 data type, allows the contents to be base-64 encoded.
- The content of this document is represented in CDA as `<NonXMLBody>` element where `<text>` content is the same form as the HL7 Version 2 message.

Some report forms are more structured and can contain multiple OBX Segments. You can see this in imaging reports, where one OBX will record the impressions and another the recommendations. In this case, each OBX can be recorded as a separate `<section>` in a `<structuredBody>` element in the CDA document.

ADT - Patient Administration

The ADT message is one of the most common message types. These messages are used to exchange patient state within healthcare facilities. Like the ORU and MDM message types, the OBX Segment is used to represent data in a name/value pair representation.

Some Message Segments Are Common to Both

There are a number of message elements that are common to different message types. These segments include the following:

- MSH - Message Header
- EVN - Event Type
- PID - Patient Identifier
- PV1 - Patient Visit
- PV2 - Patient Visit (Additional Information)
- OBX - Observation/Result

■ **Tip** Be careful with the OBX Segment because it is used differently in certain messages types.

HL7 V2 Segments to CDA Elements

Table 5-2 shows the mapping of the V2 Segments to CDA elements.

Table 5-2. *V2 Segments to CDA Elements*

Version 2	CDA XML Element
NK1	`<participant>`
PV1	`<encompassingEncounter>`
OBX	`<observation>, <nonXMLBody>`
TXA	`<ClinicalDocument>`
PID	`<recordTarget>`
MSH	`<ClinicalDocument>`
EVN	`<ClinicalDocument>`
ORC	`<infulfillmentOf>`
OBR	`<section>, <observation>`
NTE	`<text>`
SPM	`<specimen>, <procedure>`

The Data Types

Knowledge of the Version 2 and Version 3 data types is the key to your understanding how you are able to map between both versions.

HL7 Version 2 to Version 3 Data Type Mappings

Table 5-3 shows the Version 2 data types and their Version 3 equivalents.

Table 5-3. *HL7 Version 2 to Version 3 Data Type Mappings*

Version 2	Version 3
AD	ADDR
CE, CNE, CWE	CD
CN	II + PN
CK, CX, E1	II
DT, TS	TS
ED	ED
ID	CS, CD
HD	OID, UUID
NM	REAL, PQ
PN, XPN	PN
RP	TEL
TN, XTN	TEL

Mapping the HL7 Version 2.x Data Types

In this section you will see how to map the V2.x data types to the Version 3 CDA data types.

■ **Note** All the code examples showing the CDA XML elements will show the V2.x data type as their value.

To make it easier to understand the code sample in this section, let's take a look at some sample code. In the sample below, the attribute values are in single quotes and are bolded. The value for the element is also bolded. All of these values are capitalized.

```
<value xsi:type='ED' mediaType='ED.2/ED.3' extension='ED.4'>ED.5<value>
```

AD - Address

You will start with the *AD - Address*. Mapping this data type is rather simple. Listing 5-2 shows how to do this.

Listing 5-2. Address Mapping

```
<addr use='AD.7'>
     <streetAddressLine>AD.1</streetAddressLine>
     <streetAddressLine>AD.2</streetAddressLine>
     <city>AD.3</city>
     <state>AD.4</state>
     <postalCode>AD.5</postalCode>
     <country>AD.6</country>
     <county>AD.8</county>
</addr>
```

The seventh component of the HL7 2 data type is mapped to the `<addr>` *use attribute.*

CE - Coded Element, CNE - Coded No Exceptions, and CWE - Coded With Exceptions

The *CE - Coded Element data type* supports two identification codes, a primary, which should be universal, and an alternate, which can be a local code. The CNE is an extension that adds the code system version along with the original text components. The CWE data type is the same as the Version 2.x CNE data type. Listing 5-3 shows the mapping of the CDA `<code>` element.

Listing 5-3. CE = Coded Element Code Sample

```
<code code='CE.1' displayName='CE.2' codeSystemName='CE.3' codeSystem='CE.3'
codeSystemVersion='CNE.7'>
<originalText>CNE.9</originalText>
<translation code='CE.4' displayName='CE.5' codeSystemName='CE.6' codeSystem='CE.6'
codeSystemVersion='CNE.8' />
</code>
```

CN - Composite ID Number and Name

The CN data type is used to identify the participations and roles of the person that is contained within the CDA document. Besides the CN data type, there are also the XCN and PPN data types.

Let's take a look at the XCN data type within the `OBX-15` field and how it maps to the `<assignedEntity>;` see Listing 5-4.

Listing 5-4. CN Composite ID and Name Code Sample

```
<assignedEntity>
        <id root='CK-4' extension='OBX-15.1'/>
        <assignedPerson>
                <name>
                        <prefix>OBX-15.6</prefix>
                        <given>OBX-15.3</given>
                        <given>OBX-15.4</given>
                        <family>OBX-15.2</family>
                        <suffix>OBX-15.5</suffix>
                        <suffix qualfier='AC'>OBX-15.7</suffix>
                </name>
        </assignedPerson>
</assignedEntity>
```

Looking at Listing 5-4, you will see `<id root='`**`CK-4`**'. The data type for the `root` attribute can be either the **`CK`** or **`CX`**. This data type is an identifier for a person or organization in HL7 Version 2.x messages. It may or may not contain a check digit. Its value will often come from a lookup table. Listing 5-5 shows this data type mapped to a CDA `id` element.

Listing 5-5. CDA id Element Code Sample

```
<id root='CK-4' assigningAuthorityName='CK-4.1' extension='CK.1' />
```

DT - Date, TS – TimeStamp

The Date and TimeStamp data types can be used directly in the CDA document, the only difference being the format.

ED - Encapsulated Data

The Encapsulated Data type in Version 2 and Version 3 serve the same purpose. In the Version 3 document, the type is set to `ED`. The second attribute of the value identifies the MIME type. The third identifies the MIME subtype, and the forth is either `A`, `Hex`, or `Base64,` which identifies the encoding of the data.

Listing 5-6 shows how to map the Version 2 ED data type to the Version 3 ED data type.

Listing 5-6. ED - Encapuslate Data Code Sample

```
<value xsi:type='ED' mediaType='ED.2/ED.3' extension='ED.4'>ED.5<value>
```

EI - Entity Identifier

This data type is similar to the `CK/CX` data types. The only difference is it does not include a check digit. Listing 5-7 shows an example of its use.

Listing 5-7. EI - Entity Identifier Code Sample

```
<id root='EI.3' assigningAuthorityName='EI-3.1' extension='EI.1' />
```

ID - Coded value, IS - Coded value for User Defined Tables

The Version 2 ID and IS data types are like the Version 3 CS data type. They are used to represent content with fixed coding systems.

HD - Hierarchical Descriptor

The Version 2 data type is used to represent an assigning authority. The corresponding Version 3 data types are UUID and OID.

NM - Numeric

The NM data type can be directly mapped to the Version 3 PQ data type. The value is a real number and can also be mapped to Version 3 INT and REAL data types.

PN - Person Name

The mapping for the PN data type is the same as the Version 3 CN data type, except that the ID number component is missing. Listing 5-8 shows an example.

Listing 5-8. Person Name Code Sample

```
<name>
        <prefix>CN-5</prefix>
        <given>CN-3</given>
        <given>CN-4</given>
        <family>CN-2</family>
        <suffix>CN-5</suffix>
        <suffix qualifier='AC'>CN-7</suffix>
</name>
```

RP - Reference Pointer

The Version 2 RP data type is used to access data by reference instead of using a value. Version 3 supports this type of reference. The reference mapping needs to be to some form of URL. The data type for this value is set to ED, as you will see in Listing 5-9.

Listing 5-9. Reference Pointer Code Sample

```
<value xsi:type='ED' mediaType='RP.2/RP.3'>
        <reference value='RP.1 and RP.2'/>
</value>TN - Telephone Number, XTN - Extended Telecommunications Number
```

The Version 2 TN Data Type is directly mapped to the Version 3 TEL data type.

The Version 2 XTN data type can contain an e-mail address as well as a telephone number. Listing 5-10 shows the contents of the Version 3 TEL data type.

Listing 5-10. Version 3 TEL Sample Code

```
<telecom value='tel:+CCxxxxxxxxxx;ext='xxxxx' use='XTN.2'/>
```

In Listing 5-10, the value is comprised of the characters CC, which represent the two digit country code (which is required), the ten digit telephone number, and an optional extension (ext) of up to five digits.

■ **Note** The **tel:+** is the required prefix for the telecom value.

If the XTN also contains an e-mail address, it will be prefixed with "mailto:" as shown in Listing 5-11.

Listing 5-11. Version 3 TEL with E-mail Sample Code

```
<telecom value='mailto:XTN.4' use='XTN.2'/>
```

Mapping the OBX Segment

Now that you understand how the HL7 Version 2.x data types map the Version 3 data types, let's take a look at mapping from the OBX Segment to the Observation Results section.

■ **Tip** The **OBX Segment** is a widely used segment. Its primary use, however, is for *Observation/Result* reporting information contained within report messages. At times an **OBX Segment** may have an **NTE Segment** which is used to provide notes and comments.

OBX - Observation Results

The HL7 Version 3 Observation Result Segment has many uses. It contains observations about the object of its parent segment. Looking back at Listing 5-1, the OBX is a section within the structured body component. The Version 2.x OBX consists of 25 sequences.

There can be multiple OBX Segments in a Version 2.x and a Version 3 - CDA Message. For example, in the VXU/RSP it is associated with the RXA or Immunization record. The basic format within this message is a question (OBX-3) and an answer (OBX-5).

■ **Note** Since mapping rules differ in the message types that contain OBX Segments, we will not include these rules in our examples.

OBX Forms

The OBX Segment comes in two different form types. They map closely to the CDA <section> element. These forms types are named Narrative and Name/Value.

Narrative Form Type

The Narrative includes a code in the OBX-3 that will describe the *text* in the OBX-5. The OBX-3 code maps to a <code> element in the CDA <section> element. The text in the OBX-5 appears in the <text> element in the CDA <section> element. Listing 5-12 is a code sample showing the mapping for the Narrative form type.

Listing 5-12. Narrative Form Code Sample

```
<component>
        <section>
                <code code='OBX-3'/>
                <text>OBX-5</text>
                272103_1_En
                        <assignedAuthor>
                                <id root='...' extension='OBX-16.1'/>
                                <authorPerson>
                                        <name>OBX-16.2</name>
                                </authorPerson>
                        </assignedAuthor>
                </author>
                <entry>
                        <observation>
   <code code='OBX-3' .../>
   <value xsi:type='OBX-2' value='OBX-5' unit='OBX-6/>
   <effectiveTime value='OBX-14'/>
   <interpretationCode code='OBX-8' codeSystem='2.16.840.1.113883.5.83' codeSystemName='Observation
   Interpretation'/>
   <methodCode code='OBX-17'/>
   <author>
        <assignedAuthor>
                <id root='...' extension='OBX-16.1'/>
                <authorPerson>
                        <name>OBX-16.2</name>
                </authorPerson>
        </assignedAuthor>
        </author>
        <preformer>
                <assignedEntity>
                        <id root='...' extension='OBX-15.1'/>
                        <assignedPerson>
                                <name>OBX-15.2 - OBX15.8</name>
                        </assignedPerson>
                </assignedEntity>
        </preformer>
   <referenceRange>
     <observationRange>
       <code code='OBX-10' .../>
       <value xsi:type='ILV_OBX-2' unit='OBX-6'/>
                <low value='OXB-7'/>
                <high value='OBX-7'/>
                </value>
                <observationInterpretation code='N' displayName='...'
                        codeSystem='2.16.840.1.113883.5.83' codeSystemName=
                        'ObservationInterpretation'/>
     </observationRange>
   </referenceRange>
 </observation>        </entry>
        </section>
</component>
```

Name/Value Form Type

The second **OBX** form type is the **Name/Value** form type. It includes a code in the OBX-3, but uses either a coded data type, a numeric, or a timestamp that appears in the OBX-5. These map to the CDA <observation> element.

The OBX-3 code maps to a <code> element in the CDA <section> element. The value for the OBX-5 maps to the <value> element in the CDA <observation> element. Listing 15-13 shows the mapping for the Name/Value form type.

Listing 15-13. OBX to CDA Name/Value Observation Mapping Sample

```
<observation>
   <code code='OBX-3' .../>
   <value xsi:type='OBX-2' value='OBX-5' unit='OBX-6/>
   <effectiveTime value='OBX-14'/>
   <interpretationCode code='OBX-8' codeSystem='2.16.840.1.113883.5.83' codeSystemName='Observation
   Interpretation'/>
   <methodCode code='OBX-17'/>
   <author>
        <assignedAuthor>
                <id root='...' extension='OBX-16.1'/>
                <authorPerson>
                        <name>OBX-16.2</name>
                </authorPerson>
        </assignedAuthor>
        </author>
        <preformer>
                <assignedEntity>
                        <id root='...' extension='OBX-15.1'/>
                        <assignedPerson>
                                <name>OBX-15.2 - OBX15.8</name>
                        </assignedPerson>
                </assignedEntity>
        </preformer>
   <referenceRange>
     <observationRange>
       <code code='OBX-10' .../>
       <value xsi:type='ILV_OBX-2' unit='OBX-6'/>
                <low value='OXB-7'/>
                <high value='OBX-7'/>
                </value>
                <observationInterpretation code='N' displayName='...'
                        codeSystem='2.16.840.1.113883.5.83' codeSystemName=
                        'ObservationInterpretation'/>
     </observationRange>
   </referenceRange>
 </observation>
```

Meaningful Use Certification for Immunizations

The task given was to build a BizTalk-based solution to report immunizations given to patients to comply with Meaningful Use 2 guidelines so that Urgent Care facility clients could participate in the government's Meaningful Use 2 program. This program is designed to compensate the facilities for the submission of data so the government, be it state or federal, can track both the immunization history and ongoing immunization events of the general public.

Overall, based on the certification test cases, our solution was designed to report on all areas where immunizations might occur. However, for our purposes, we will only be focusing on ***Urgent Care Immunizations***, since that is the solution we built. This results in passing a total of 3 test cases per scenario, with 7 scenarios in all, resulting in 21 test cases total. These cases cover a wide array of situations, from vaccinations of toddlers, children and adults to electing not to perform a vaccination due to immunity (perhaps the patient had the disease before or was previously immunized for it) or because the patient, their parent, or guardian refused the immunity.

With these details in mind, the solution must be able to dynamically address the scenarios without manual input from the user to direct the expected response, beyond making selections in the user interface that will then be committed to the database tables. The idea was for this process to remain transparent to the user once the setting to participate in the Meaningful Use 2 program was enabled, with the exception being any additional fields the user needs to complete (if any) in the user interface to input the data necessary for Meaningful Use 2 credit.

Design

The initial task was to diagram the process as we expected it would be, understanding that details discovered along the way would likely drive modifications of the process and thus render the initial diagram incomplete. The process, once we consulted with the Development team working on the EMR software as well, ended up following the process depicted in Figure 5-2.

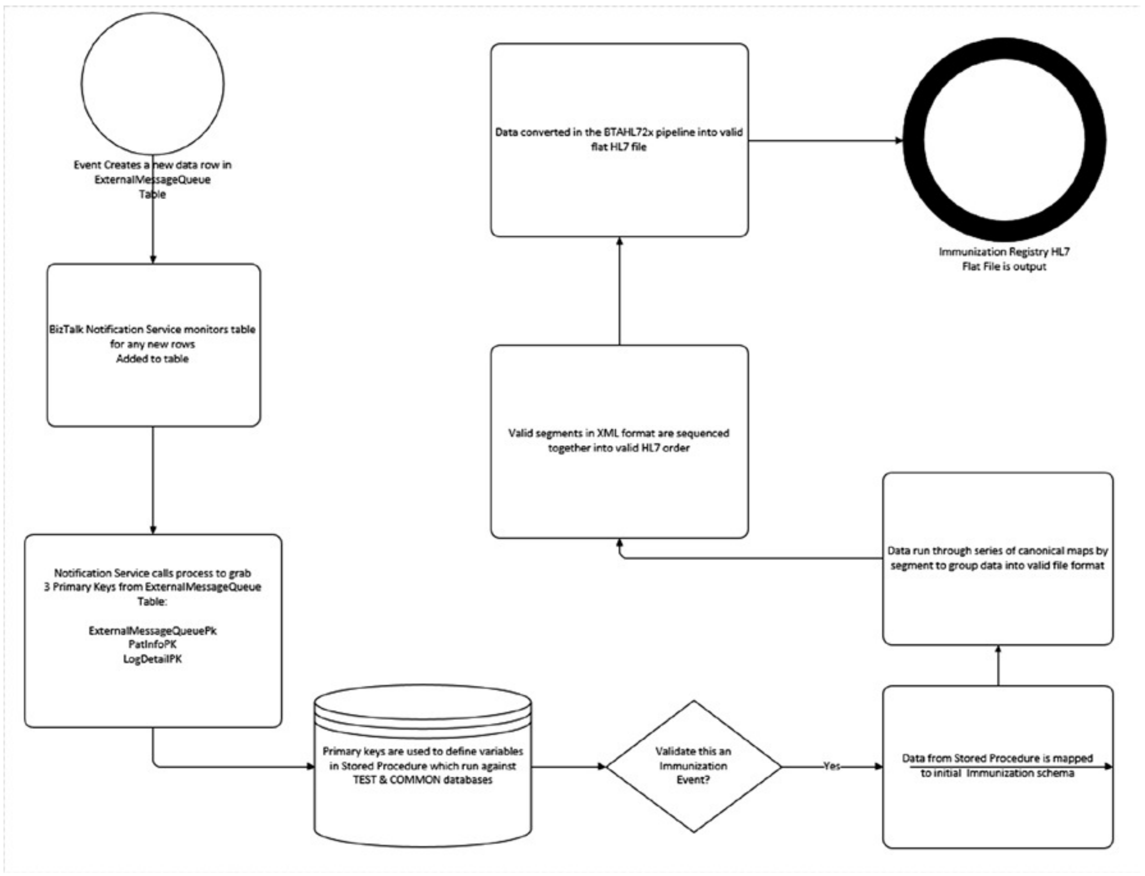

Event Creates a new data row in ExternalMessageQueue Table
BizTalk Notification Service monitors table for any new rows Added to table
Notification Service calls process to grab 3 Primary Keys from ExternalMessageQueue Table:
ExternalMessageQueuePk
PatInfoPK
LogDetailPK
Primary keys are used to define variables in Stored Procedure which run against TEST & COMMON databases
Validate this an Immunization Event?
Yes
Data from Stored Procedure is mapped to initial Immunization schema
Data run through series of canonical maps by segment to group data into valid file format
Valid segments in XML format are sequenced together into valid HL7 order
Data converted in the BTAHL72x pipeline into valid flat HL7 file
Immunization Registry HL7 Flat File is output

Figure 5-2. *Initial design*

Figure 5-2 is a high-level view of the process, but it displays most of the components that make up the Immunization Registry. At the time of design and build, no endpoint had been determined for sending the data to whichever government agency, be it state or federal, that would be compiling the database of the Immunization records.

We discovered that the easiest way to build the solution was to first identify the sections to complete by diagramming the process, then build and test them individually as we went, and finally assemble and test them as a completely assembled project.

When reviewing the project, we determined the best way to pull the data would be via a stored procedure that could access multiple databases and tables. Once this was determined, we completed the initial list of tasks we needed to build out for our project.

1. Event Notification
2. Stored Procedures
3. Mapping
4. Processing
5. Error Handling
6. Testing

Ideally we would have liked to begin with mapping, but since our stored procedures ultimately determined the starting schema, we needed to address them first.

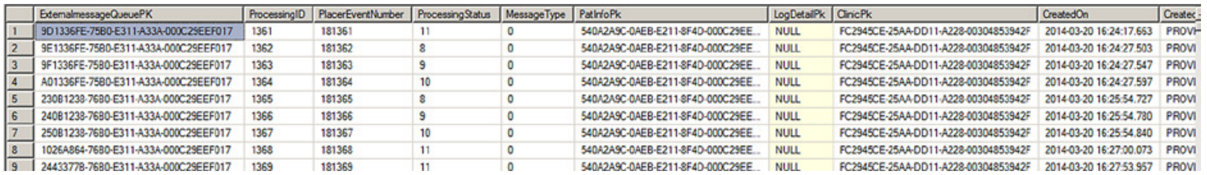
In the mapping process, there is almost inevitably a need to revisit the stored procedures and make modifications, so these two steps are very much hand in hand. Also, while *Testing* was last on our list, we tested all along the way—from validating maps and the stored procedure via their test output to validating the orchestration and error handling functions as designed. We did this for every segment because getting to the end and assembling the pieces and then testing makes it more difficult to determine the source of any errors that will inevitably arise.

Development

Development was split into two separate solutions. One handled the creation of the Immunization message and the other handled the event notification.

Event Notification

We decided to take advantage of the WCf-SQL Adapter Notification Services. We utilized this to get notifications of new records inserted into the ExternalMessageQueue table. Figure 5-3 shows the `ExternalMessageQueue` table.



	ExternalmessageQueuePK	ProcessingID	PlacerEventNumber	ProcessingStatus	MessageType	PatInfoPk	LogDetailPk	ClinicPk	CreatedOn	Createc
1	9D1336FE-75B0-E311-A33A-000C29EEF017	1361	181361	11	0	540A2A9C-0AEB-E211-8F4D-000C29EE...	NULL	FC2945CE-25AA-DD11-A228-00304853942F	2014-03-20 16:24:17.663	PROVI
2	9E1336FE-75B0-E311-A33A-000C29EEF017	1362	181362	8	0	540A2A9C-0AEB-E211-8F4D-000C29EE...	NULL	FC2945CE-25AA-DD11-A228-00304853942F	2014-03-20 16:24:27.503	PROVI
3	9F1336FE-75B0-E311-A33A-000C29EEF017	1363	181363	9	0	540A2A9C-0AEB-E211-8F4D-000C29EE...	NULL	FC2945CE-25AA-DD11-A228-00304853942F	2014-03-20 16:24:27.547	PROVI
4	A01336FE-75B0-E311-A33A-000C29EEF017	1364	181364	10	0	540A2A9C-0AEB-E211-8F4D-000C29EE...	NULL	FC2945CE-25AA-DD11-A228-00304853942F	2014-03-20 16:24:27.597	PROVI
5	230B1238-76B0-E311-A33A-000C29EEF017	1365	181365	8	0	540A2A9C-0AEB-E211-8F4D-000C29EE...	NULL	FC2945CE-25AA-DD11-A228-00304853942F	2014-03-20 16:25:54.727	PROVI
6	240B1238-76B0-E311-A33A-000C29EEF017	1366	181366	9	0	540A2A9C-0AEB-E211-8F4D-000C29EE...	NULL	FC2945CE-25AA-DD11-A228-00304853942F	2014-03-20 16:25:54.780	PROVI
7	250B1238-76B0-E311-A33A-000C29EEF017	1367	181367	10	0	540A2A9C-0AEB-E211-8F4D-000C29EE...	NULL	FC2945CE-25AA-DD11-A228-00304853942F	2014-03-20 16:25:54.840	PROVI
8	1026A864-76B0-E311-A33A-000C29EEF017	1368	181368	11	0	540A2A9C-0AEB-E211-8F4D-000C29EE...	NULL	FC2945CE-25AA-DD11-A228-00304853942F	2014-03-20 16:27:00.073	PROVI
9	2443377B-76B0-E311-A33A-000C29EEF017	1369	181369	11	0	540A2A9C-0AEB-E211-8F4D-000C29EE...	NULL	FC2945CE-25AA-DD11-A228-00304853942F	2014-03-20 16:27:53.957	PROVI

Figure 5-3. *ExternalMessageQueue table*

This table is also used by other applications. The columns of interest to us are defined in Table 5-4.

Table 5-4. *ExternalMessageQueue Fields*

Column	Description	SQL Data Type
ExternalMessageQueuePK	This column contains the value defining each row uniquely.	Uniqueidentifer
BTSProcessingStatus	This column contains the value that is updated as the process runs, showing the status of the event for that unique row at a given time.	Varchar
PatInfoPk	This value uniquely identifies a patient in the system, and is consistent across all tables in the system for that patient.	Uniqueidentifer
MessageType	This column tells a process what the row being inserted is used for it (the process) to address. The Immunization Registry looks for rows with a value of zero (0) here.	Integer
DBServerName	Name of the database server	Varchar
DBName	Name of the database	Varchar
LogDetailPk	Primary key for LogDetail table	Uniqueidentifer

The table was designed so that, upon the insertion of the row, the orchestration, which was receiving notifications from this table for an insert, would wake up and check the Processing Status column for the `NEW` value. If it found any, it would plug in the other values profiled above and run the `GetUpdateExternalMessageQueue` stored procedure. If not, it went back to a dormant status, waiting for the insertion of another row.

GetUpdateExternalMessageQueue Stored Procedure

This procedure was fired when there was a new record inserted into the ExternalMessageQueue table. It returned the values shown in ExternalMessageQueue fields shown above, which we needed to pass into the Immunization data stored procedures.

Once the stored procedure returned the values profiled above, it had the keys necessary to differentiate the records it needed from the various tables involved in the Immunization Registry event.

Stored Procedures

The stored procedures for this solution needed to touch two databases: `TEST` and `COMMON`. As the names suggest, TEST contained all the TEST data that mirrors a production environment, while `COMMON` contained the information that `TEST` and all the other databases have in common and share in everyday processes.

Immunization Data Stored Procedures

We used four stored procedures to get immunization data for a patient:

- GetImmunizationData
- GetImmunizationNextOfKin
- GetImmunizationPatientInfoVaccine
- GetImmunizationVaccineTypes

For our purposes, it made sense to use multiple stored procedures which would make finding the output data easier. It also kept like sections of data that (mostly) pull their information from a few specific tables common to their section of data.

- ***GetImmunizationData***: Patient Info contains select statements for what one would expect to be patient-specific information, largely demographic in this exercise.
- ***GetImmunizationNextOfKin***: Patient Next of Kin contains select statements that result in demographic information—if it exists in the system—for next of kin listed for the particular patient specified in the primary keys we gathered from the ExternalMessageQueue table.
- ***GetImmunizationPatientInfoVaccine***: Patient Info Vaccine contains information about the patient and vaccine administered as they pertain to the Immunization event itself, such as the clinic where it was administered, the site on the patient's body, the vaccine name, manufacturer, lot number, etc.
- ***GetImmunizationVaccineTypes***: Contains some of the more hardcore vaccine information from the system. Examples of the data gathered here include the vaccination description, CVX code, VIS (vaccine information sheet) date, etc.

So we pulled the necessary identifying values from the initial table (`ExternalMessageQueue`) and used those values to specify to the stored procedures what data we wanted from the system to create our Immunization Registry file. Up until this part, this had been done using SQL-based work. But in order to effectively manipulate the data, we needed to leverage the power of BizTalk.

Mapping

In BizTalk, when working with HL7 data, the easiest way to manipulate the data is by using the maps. Because of the scope of this project, we didn't drill down to the node/field level for every field. But what is important is both the order and use of the maps, because there is a purpose for how and why they are used as they are.

Keep in mind there are likely many ways to arrive at a solution for a project; our goal was to implement as many best practices as possible to keep the project's solutions as elegant as possible while simultaneously contributing to the most sustainable model, which is in itself a best practice. We wanted to design this solution so that the person interacting with it would not have to be a BizTalk power user, freeing up the BizTalk developers to focus on development.

Source Schemas

As mentioned, the stored procedure holds the origins of the initial schema. The following are the steps we took.

CREATING OUR SOURCE SCHEMAS

We created a new BizTalk project.

- We started off by right-clicking and choosing Add ➤ Add Generated Items ➤ Consume Adapter Service.

■ **Note** This exercise is familiar to anyone working with BizTalk who has interfaced with SQL.

- The next screen, shown in Figure 5-4, is where we choose the binding (sqlBinding from the pull-down) and profile the connection to our stored procedure which was the basis for our initial schema.

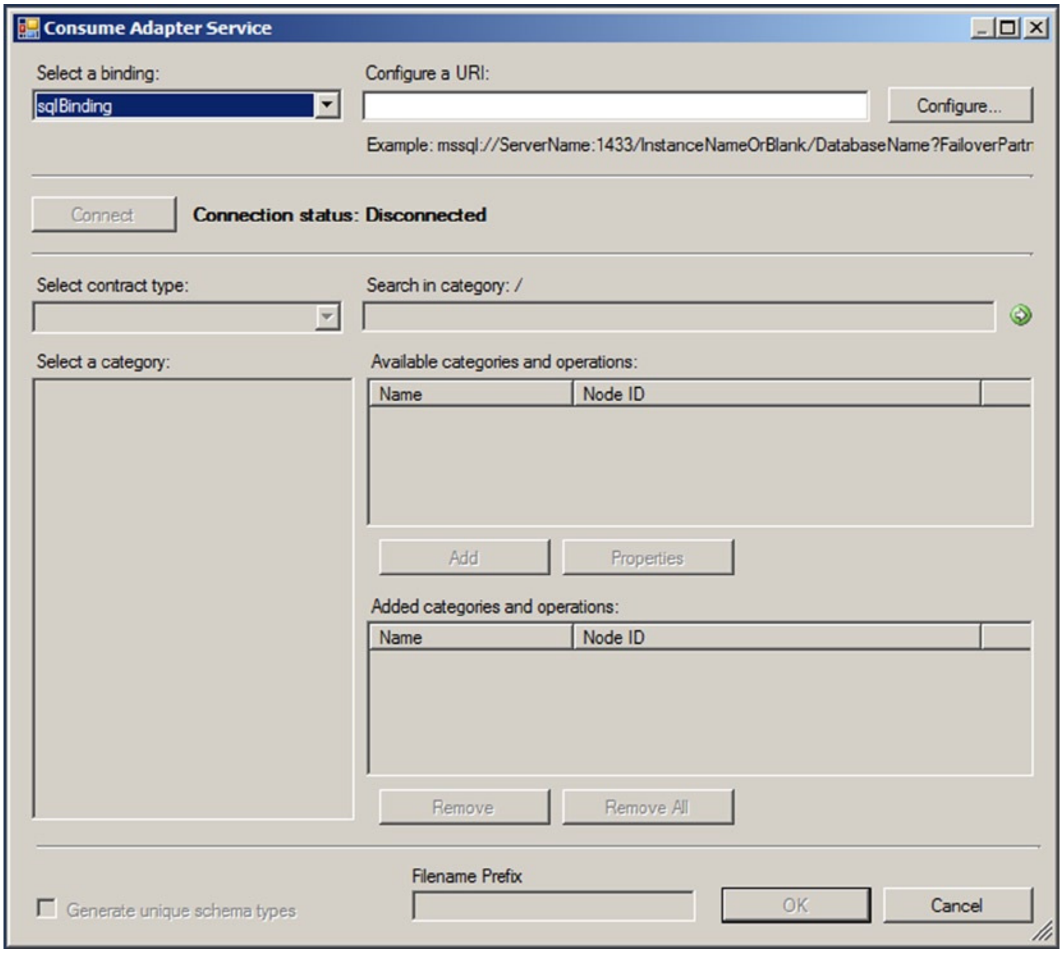
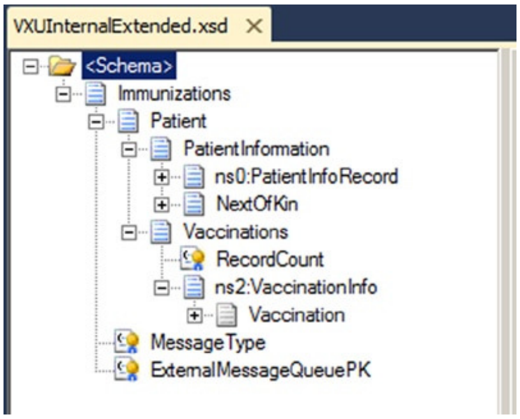


Figure 5-4. *Consume Adapter Service*

- Once the connection credentials were entered and our stored procedure was selected, clicking OK resulted in not only the initial schema being created but also the sibling schemas.
- Once created, we renamed our schema to VXUInternalExtended. This is the starting point for all of our maps; all of the data necessary to create the Immunization Registry file will need to come through this schema.

VXUInternalExtended Schema

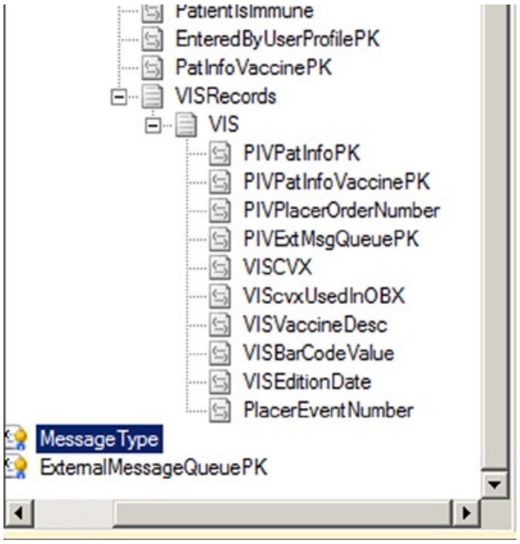
The schema itself initially would appear to have four equal sections, but over time we determined the correct configuration for purposes of looping through the data was as shown in Figure 5-5.



```
VXUInternalExtended.xsd ×
<Schema>
  Immunizations
    Patient
      PatientInformation
        ns0:PatientInfoRecord
        NextOfKin
      Vaccinations
        RecordCount
        ns2:VaccinationInfo
          Vaccination
    MessageType
    ExternalMessageQueuePK
```

Figure 5-5. *VXUInternalExtended schema*

The reasons for this structure make sense; one would want to keep Patient info and the info of their Next of Kin together, as well as the vaccination information. While it cannot be shown in Figure 5-6, the VISRecords node is a child of the Vaccination record.



```
      PatientIsImmune
      EnteredByUserProfilePK
      PatInfoVaccinePK
      VISRecords
        VIS
          PIVPatInfoPK
          PIVPatInfoVaccinePK
          PIVPlacerOrderNumber
          PIVExtMsgQueuePK
          VISCVX
          VIScvxUsedInOBX
          VISVaccineDesc
          VISBarCodeValue
          VISEditionDate
          PlacerEventNumber
MessageType
ExternalMessageQueuePK
```

Figure 5-6. *VIS Records structure*

This allows for looping when necessary, as is the case when a given vaccination is a multi-part vaccination and contains multiple vaccine records for a single vaccination event. In that case, regardless of the fact that it was a single vaccination event, the spec dictates each vaccine be profiled in the Immunization Registry output file.

In the initial schema we promoted some nodes as Distinguished Fields.

- RecordCount, which will be needed at reassembly time.
- MessageType, which defines the type of message.
- ExternalMessageQueuePK, which is the unique identifier for the message.

This project also makes use of canonical schemas, which serves as a basis to manipulate the data further. If the data needed to be looped through or manipulated in any fashion, it is a best practice to perform these tasks in the canonical maps. The goal is to have data in its final format and then if necessary, as it was in this project, stitch it all back together into the valid HL7 format.

For our purposes, we broke each of the HL7 Segments out into its own canonical map and then mapped from our initial stored proc-based schema to each canonical. Once these mappings were complete, we could use a single input file which would simulate the data coming from the stored proc-based VXUInternalExtended schema to test the single segment output. This would result in a more manageable initial test which would make tweaking the map easier.

Because of the size of the segments, we ended up with four initial "stored proc schema-to-canonical" maps:

- ImmunizationsExtended_To_PID_PD1_NK1
- ImmunizationsExtended_To_RXA_Segment (which also included the RXR Segment)
- ImmunizationsExtended_To_ORC_Segment
- ImmunizationsExtended_To_OBX_Temp

The ImmunizationsExtended_To_OBX_Temp map included the _Temp designation because the sheer size and complexity of the catch-all OBX Segment suggested we would end up utilizing additional maps in the assembly process.

In these maps, the heavy lifting occurs to manipulate the data to the formats required by the HL7 2.5.1 specification. Whether it is date formatting or logic to tell the map to only create the field if data exists, it is all here. While business rules could have been utilized for some of this, since our initial project was a base output designed for Meaningful Use 2 certification, we elected to exercise the map-based logic for this portion.

Keeping in mind that the map is executed from the top down, we were careful in the placement of our functoids. Also, it is a best practice to utilize the tabs at the bottom of the map to group the mappings so a minimum is shown on the screen at a given time. Some of the HL7 Segments get very busy, and this cuts down on the traffic on the screen, enabling edits to be more easily executed when modifying the maps. Figure 5-7 (the RXA/RXR map) is a very clean looking map which is easier to understand.

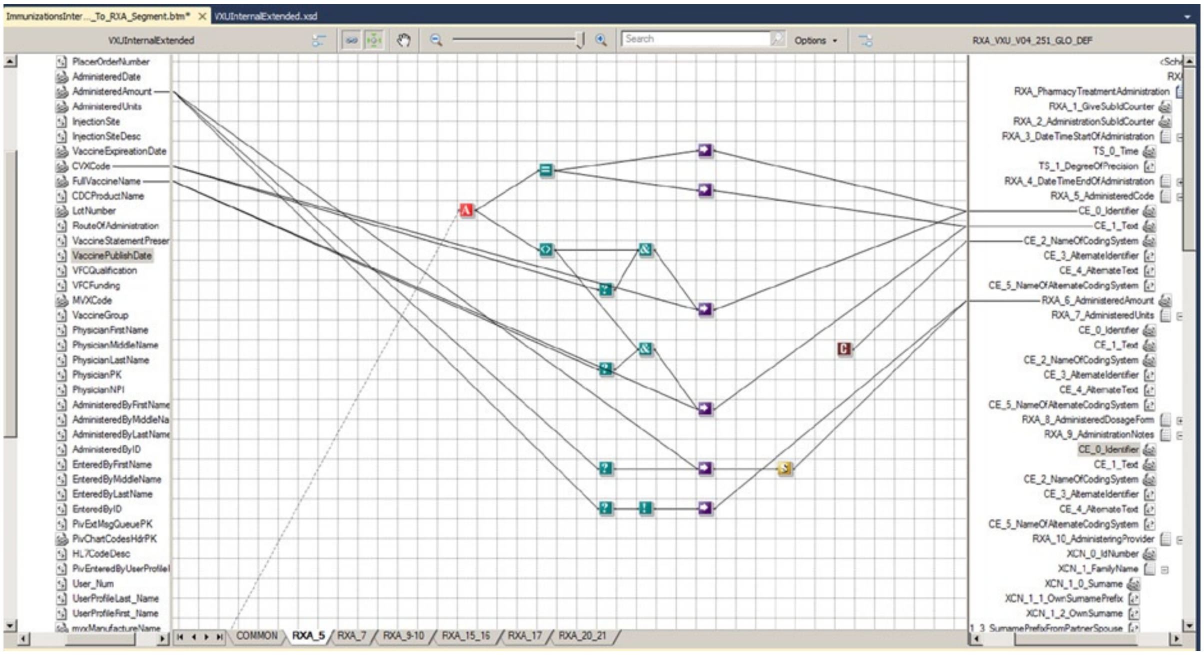


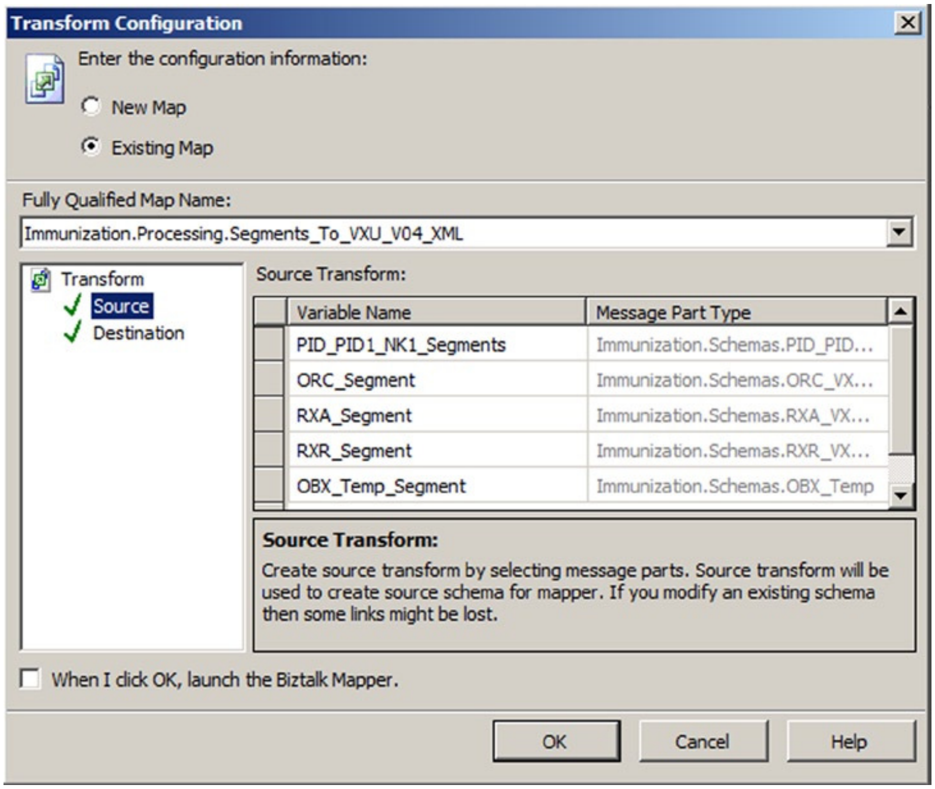
Figure 5-7. Easy-to-understand map

Once these maps were completed, we manipulated the data into a format acceptable for use by HL7. We knew this by checking the examples provided by NIST which show the expected pipe-delimited output.

Our output was still in XML, but we could verify that the fields were being populated and confirm the format of what was being pushed into each field. The main purpose of using a canonical is to limit the dirty work of data manipulation to the maps using the canonicals as destinations, so the data being pushed to the HL7 Accelerator is clean and in the right order, only relying on the Accelerator for final, minor changes and the final HL7 format.

Once the maps to the canonical versions were completed, the next map was to the final XML, non-HL7 version of the file. The orchestration drove the usage of the individual canonical schemas to the destination schema (`VXU_V04_251_GLO_DEF.xsd`) in the final map to stage them for assembly in the final map.

The transformation configurations for assembling the multiple canonical schemas to the final, HL7-friendly schema (`VXU_V04_251_GLO_DEF.xsd`) are shown in Figure 5-8 and 5-9.



Transform Configuration

Enter the configuration information:

New Map [✓ Existing Map]

Fully Qualified Map Name: Immunization.Processing.Segments_To_VXU_V04_XML

Transform
✓ Source
✓ Destination

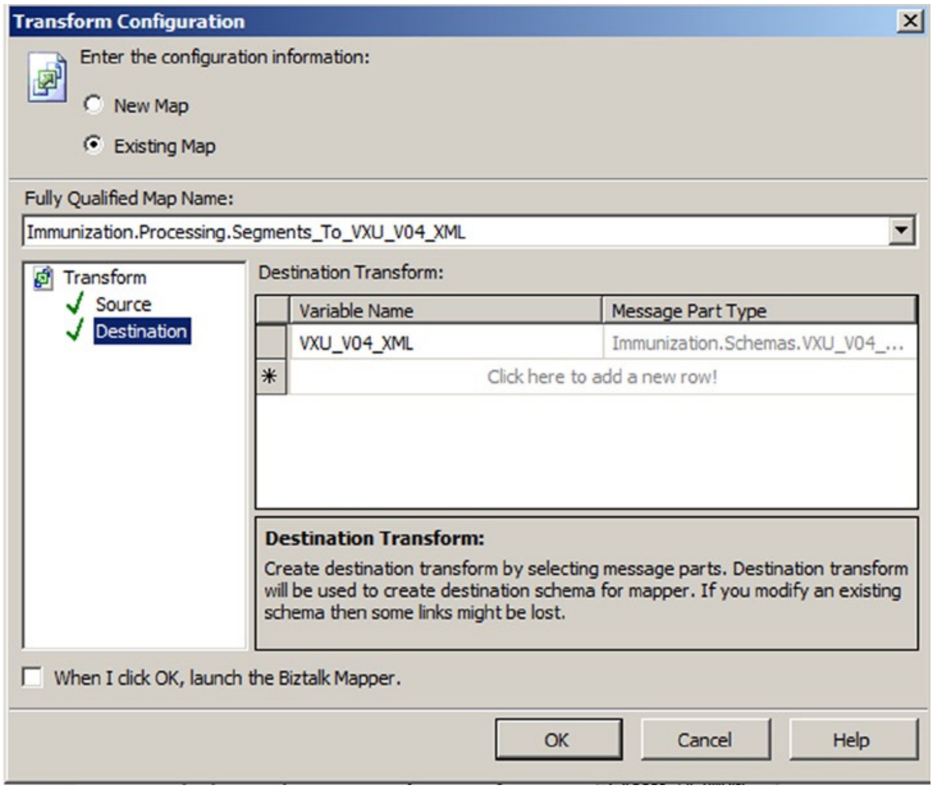
Source Transform:

Variable Name	Message Part Type
PID_PID1_NK1_Segments	Immunization.Schemas.PID_PID...
ORC_Segment	Immunization.Schemas.ORC_VX...
RXA_Segment	Immunization.Schemas.RXA_VX...
RXR_Segment	Immunization.Schemas.RXR_VX...
OBX_Temp_Segment	Immunization.Schemas.OBX_Temp

Source Transform:
Create source transform by selecting message parts. Source transform will be used to create source schema for mapper. If you modify an existing schema then some links might be lost.

☐ When I click OK, launch the Biztalk Mapper.

Figure 5-8. *Source messages*



Fully Qualified Map Name: Immunization.Processing.Segments_To_VXU_V04_XML

Variable Name	Message Part Type
VXU_V04_XML	Immunization.Schemas.VXU_V04_...
Click here to add a new row!	

Destination Transform:
Create destination transform by selecting message parts. Destination transform will be used to create destination schema for mapper. If you modify an existing schema then some links might be lost.

☐ When I click OK, launch the Biztalk Mapper.

Figure 5-9. *Destination message*

Keep in mind that this transformation was the initial mechanism to assemble the segments of data from the canonical schemas into the final version the data assumed prior to its trip through the HL7 Accelerator. With this in mind, it was important to test the output heavily so that we had an accurate expectation of the data and order of the data the HL7 Accelerator finally assembled.

The initial collapsed appearance of the schemas involved in this transformation shows some of the organization to the schemas. However, notice the tabs at the bottom to keep the segments separated and easier to work with, as well as one that allows the necessary looping even on this final transformation to be isolated as well (see Figure 5-10).

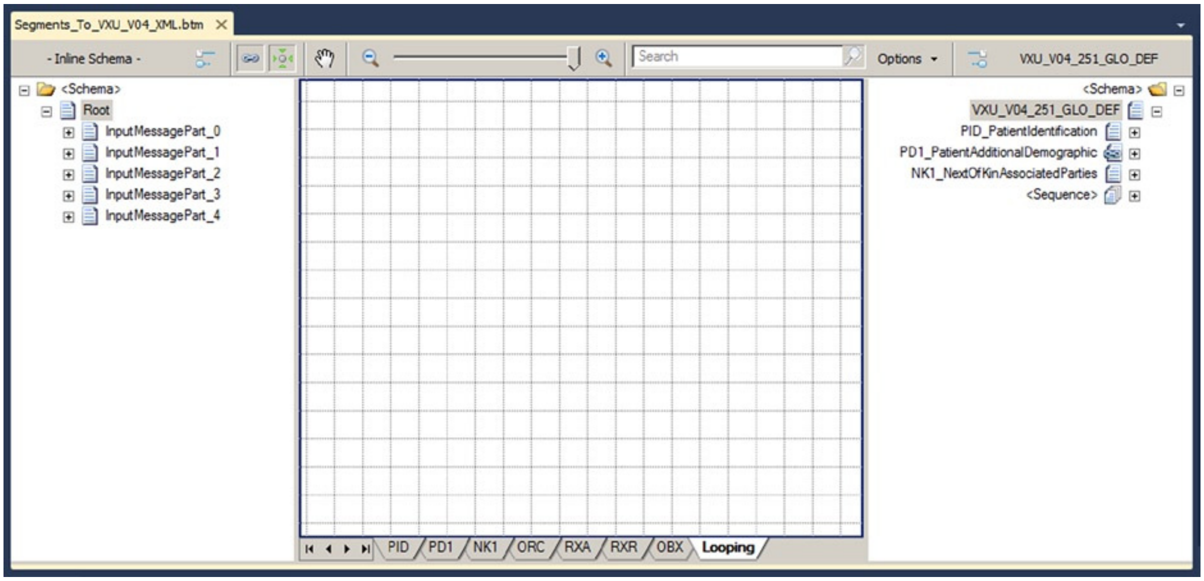


Figure 5-10. *Segments map*

A partial expansion of the schemas shows more of the configurations of the schemas in this transformation. Figure 5-11 shows a portion of the Target schema.

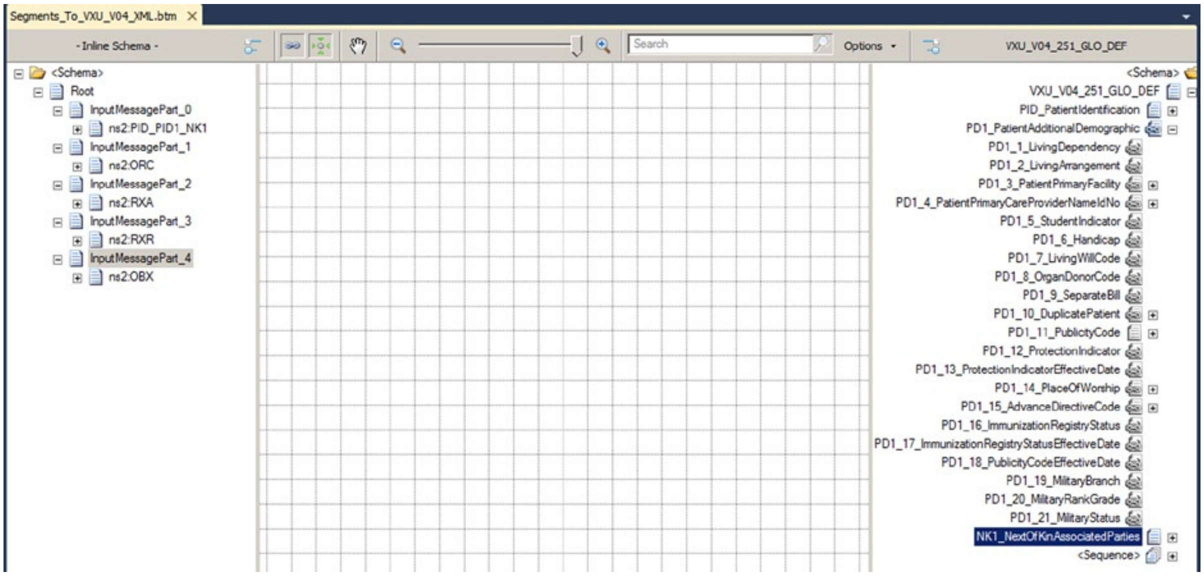
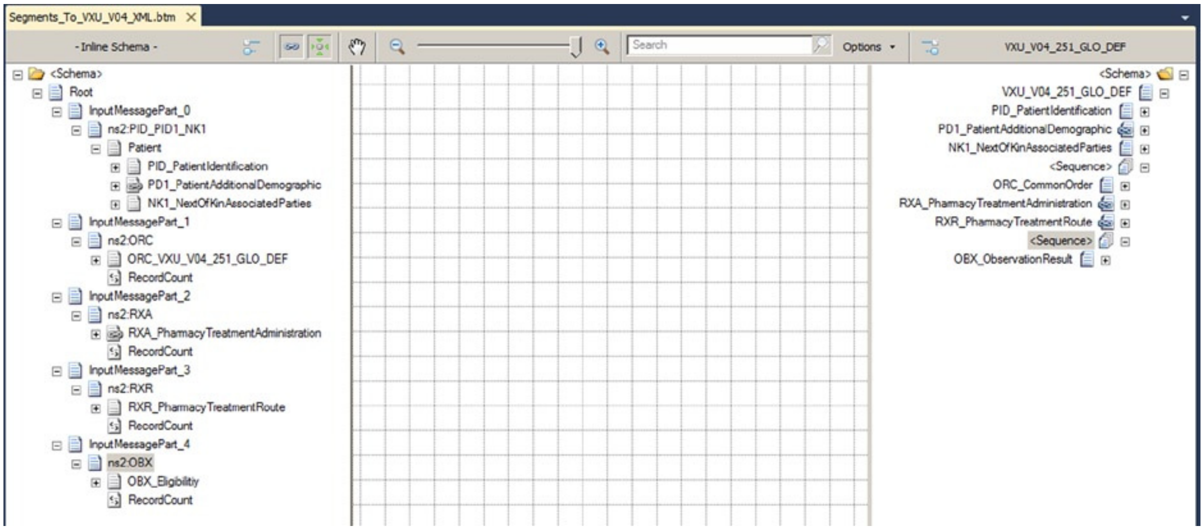


Figure 5-11. *Segment expansion*

In the final node shown above, Sequence, sit the nodes for the ORC, RXA, and RXR Segments, with another Sequence node, under which exists the final segment, which in this case is the OBX Segment.

As we show some additional detail in Figure 5-12 by expanding the sub-nodes of the Inline schema, notice the RecordCount field for each repeating segment. This is critical to getting the final order correct in the final output.



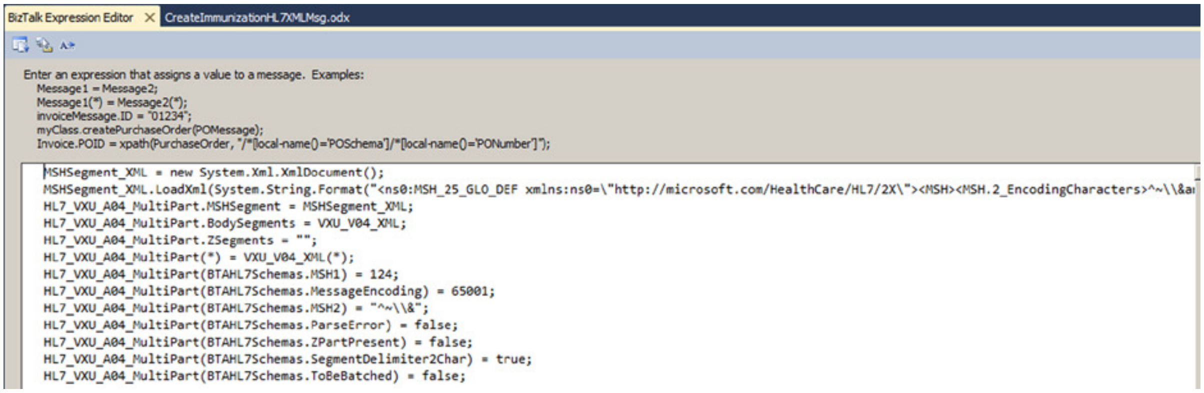
```
<Schema>
  Root
    InputMessagePart_0
      ns2:PID_PID1_NK1
        Patient
          PID_PatientIdentification
          PD1_PatientAdditionalDemographic
          NK1_NextOfKinAssociatedParties
    InputMessagePart_1
      ns2:ORC
        ORC_VXU_V04_251_GLO_DEF
        RecordCount
    InputMessagePart_2
      ns2:RXA
        RXA_PharmacyTreatmentAdministration
        RecordCount
    InputMessagePart_3
      ns2:RXR
        RXR_PharmacyTreatmentRoute
        RecordCount
    InputMessagePart_4
      ns2:OBX
        OBX_Eligibility
        RecordCount
```

```
<Schema>
  VXU_V04_251_GLO_DEF
    PID_PatientIdentification
    PD1_PatientAdditionalDemographic
    NK1_NextOfKinAssociatedParties
    <Sequence>
      ORC_CommonOrder
      RXA_PharmacyTreatmentAdministration
      RXR_PharmacyTreatmentRoute
      <Sequence>
        OBX_ObservationResult
```

Figure 5-12. *Inline schema*

Also take note of the nested/parent-child structure of these schemas; this is done for looping, which is necessary because the nature of the data involved requires looping through multiple results, all of which are necessary for the given Immunization Registry event.

Looking through these schemas one might have noticed the MSH Segment was missing from the maps; it is created in the orchestration after the assembly of the canonical data into the integrated, almost-complete HL7 message by utilizing the expression shown in Figure 5-13.



Enter an expression that assigns a value to a message. Examples:
Message1 = Message2;
Message1(*) = Message2(*);
invoiceMessage.ID = "01234";
myClass.createPurchaseOrder(POMessage);
Invoice.POID = xpath(PurchaseOrder, "/*[local-name()='POSchema']/*[local-name()='PONumber']");

```
MSHSegment_XML = new System.Xml.XmlDocument();
MSHSegment_XML.LoadXml(System.String.Format("<ns0:MSH_25_GLO_DEF xmlns:ns0=\"http://microsoft.com/HealthCare/HL7/2X\"><MSH><MSH.2_EncodingCharacters>^~\\&a
HL7_VXU_A04_MultiPart.MSHSegment = MSHSegment_XML;
HL7_VXU_A04_MultiPart.BodySegments = VXU_V04_XML;
HL7_VXU_A04_MultiPart.ZSegments = "";
HL7_VXU_A04_MultiPart(*) = VXU_V04_XML(*);
HL7_VXU_A04_MultiPart(BTAHL7Schemas.MSH1) = 124;
HL7_VXU_A04_MultiPart(BTAHL7Schemas.MessageEncoding) = 65001;
HL7_VXU_A04_MultiPart(BTAHL7Schemas.MSH2) = "^~\\&";
HL7_VXU_A04_MultiPart(BTAHL7Schemas.ParseError) = false;
HL7_VXU_A04_MultiPart(BTAHL7Schemas.ZPartPresent) = false;
HL7_VXU_A04_MultiPart(BTAHL7Schemas.SegmentDelimiter2Char) = true;
HL7_VXU_A04_MultiPart(BTAHL7Schemas.ToBeBatched) = false;
```

Figure 5-13. *MSH Segments*

This expression defines the static values utilized in the MSH Segment. The rest was input via the HL7 Accelerator BTAHL7 Configuration Explorer, enabling them to be driven by the party setup and allowing for dynamic assignment setup to be automatically populated based on criteria dictated by the endpoint to which the information is being sent. For example, if these were going to state agencies, we could assign a state-based set of criteria to drive what values were populated in the dynamic fields by state.

Processing

Once the maps were determined, the *orchestrations* needed to be created to utilize the maps and the schemas contained in the maps. Since the orchestrations contained the error handling, it seemed logical to include them in this section as well. Ideally we would have liked to put everything into a single orchestration, but for this solution we separated the mechanisms in this manner.

The initial process initiated the Immunization Registry file creation. The orchestration assembled the raw data for the Immunization Registry file. The final orchestration created the true HL7 version of the Immunization Registry file.

■ **Note** Since our solution for initiating the process was very software specific, we will focus on the orchestration that picked up the input from the Event Notification application.

Orchestrations

We used three orchestrations for this part of the solution.

- GetData
- CreateImmunizationHL7XMLMsg
- CreateHL7ImmunizationMsg

GetData

The orchestration that began the true Immunization Registry creation process was the `GetData` orchestration, so named because its overall purpose was to take the Distinguished Fields handed off to it and plug them into four separate stored procedures to pull the data from the multiple tables and databases necessary for the Immunization Registry file.

In review, the variables that were defined in the "Immunization Data Stored Procedures" section are defined again below:

- ***ExternalMessageQueuePK***: This column holds the values defining each row uniquely.
- ***ProcessingStatus***: This column holds the values that are updated as the process runs, showing the status of the event for that unique row at a given time.
- ***PatInfoPk***: This value uniquely identifies a patient in the system, and is consistent across all tables in the system for that patient.
- ***MessageType***: This column tells a process if the row being inserted is for it (the process) to address. The Immunizations Registry looks for rows with a value of zero here.

The initial step in this orchestration took the values in the initial receive shape and used the filter in the SetMsgQueue_Request shape, validating that only the values in rows with a `MessageType = 0` were being utilized, since these identify the row as being a request for an Immunization Registry file. Any other rows whose MessageType was not equal to zero were ignored and left unmodified, which will become pertinent shortly.

This initial part of the orchestration took shape as shown in Figure 5-14.

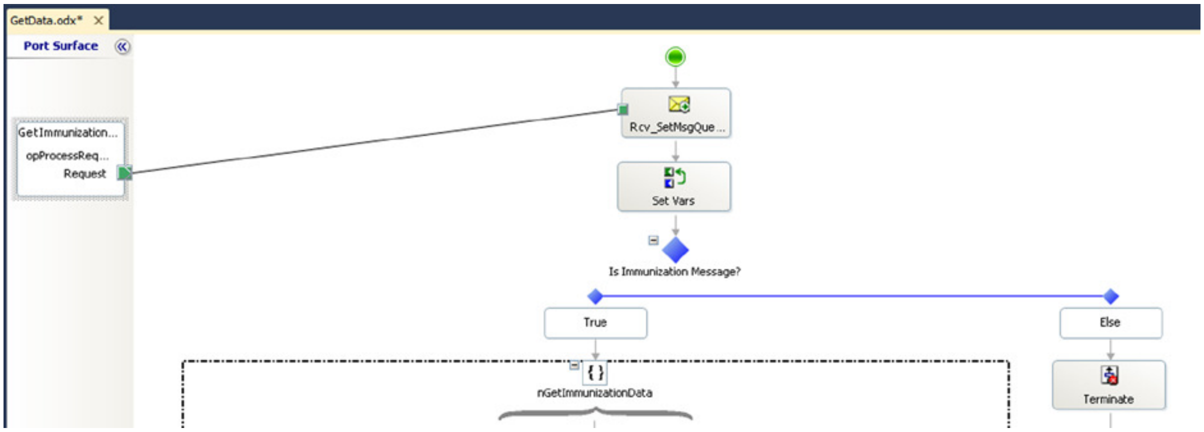
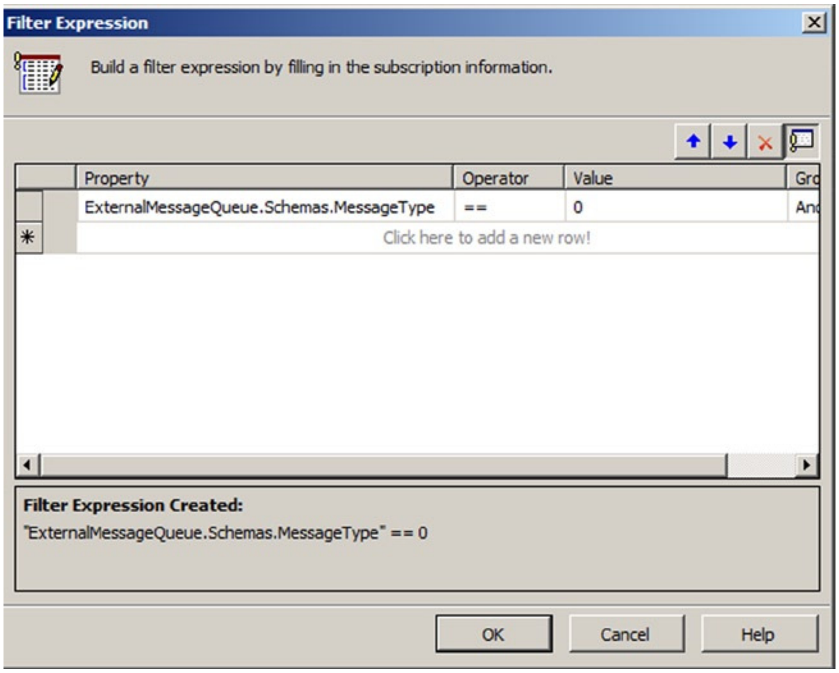


Figure 5-14. Initial part of orchestration

The filter for the SetMsgQueue_Request Receive shape was configured as shown in Figure 5-15.

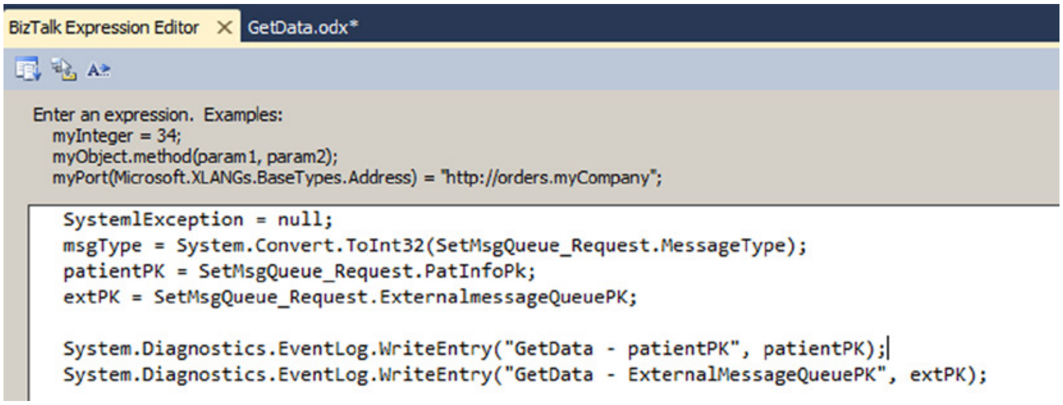

Filter Expression
Build a filter expression by filling in the subscription information.
Click here to add a new row!
OK
Cancel
Help

Property	Operator	Value	Gro
ExternalMessageQueue.Schemas.MessageType	==	0	And

Filter Expression Created:
"ExternalMessageQueue.Schemas.MessageType" == 0

Figure 5-15. Filter expression

Once these values were provided to the GetData orchestration, they were assigned to variables in the Expression Editor shape for use further down in the orchestration. The expression we are used are shown in Figure 5-16.



```
Enter an expression.  Examples:
   myInteger = 34;
   myObject.method(param1, param2);
   myPort(Microsoft.XLANGs.BaseTypes.Address) = "http://orders.myCompany";
```

```
SystemlException = null;
msgType = System.Convert.ToInt32(SetMsgQueue_Request.MessageType);
patientPK = SetMsgQueue_Request.PatInfoPk;
extPK = SetMsgQueue_Request.ExternalmessageQueuePK;

System.Diagnostics.EventLog.WriteEntry("GetData - patientPK", patientPK);
System.Diagnostics.EventLog.WriteEntry("GetData - ExternalMessageQueuePK", extPK);
```

Figure 5-16. *Assigning to local variables*

Once these steps were executed, we had all the information to answer the first question in our orchestration, which was whether or not what is being provided to us is truly an Immunization Registry message request. The orchestration looked for that zero (`0`) value in the `MessageType` and as long as it was found, the process continued. If not, it would terminate. The step was configured as shown in Figure 5-17.

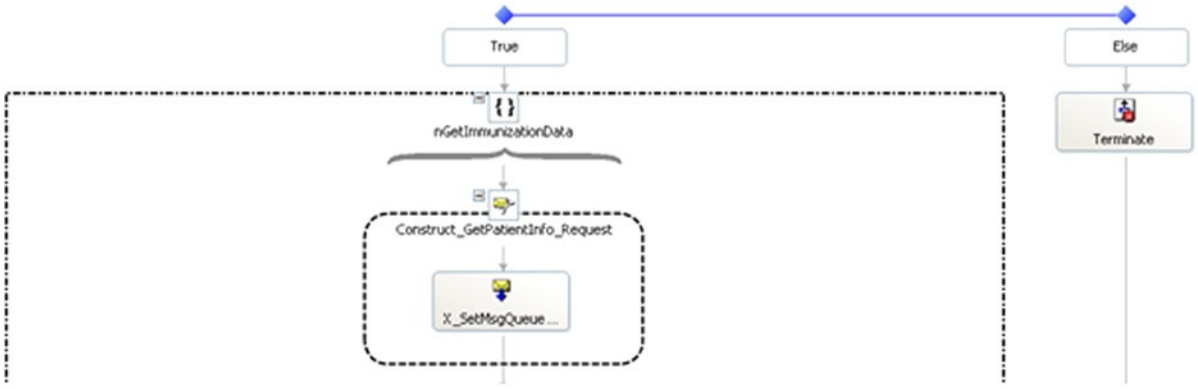


Figure 5-17. *Terminate*

GetImmunizationPatientInfo

The first step in our orchestration was to create the GetImmunizationPatientInfo request message. We used the map shown in Figure 5-18.

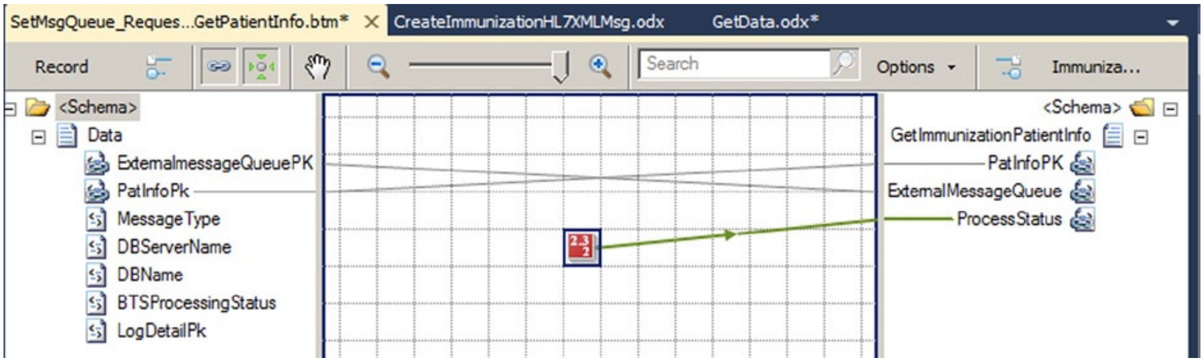


Figure 5-18. *Construct_GetPatientInfo_Request Map*

In this map, the ProcessStatus was set using an integer functoid set to `0`. This data was used in calling the first stored procedure.

The first stored procedure to utilize the variables defined in the map above was the `BizTalk.GetImmunizationPatientInfo` stored procedure, which was used to pull patient-specific data for the message being created. Since most of this data mapped to the initial segments (PID and PD1 segments) of the HL7 file, it was easier to align the sections by design to make mapping earlier, hence this being the first of the four stored procedures. Figure 5-19 shows the orchestration scope for this process.

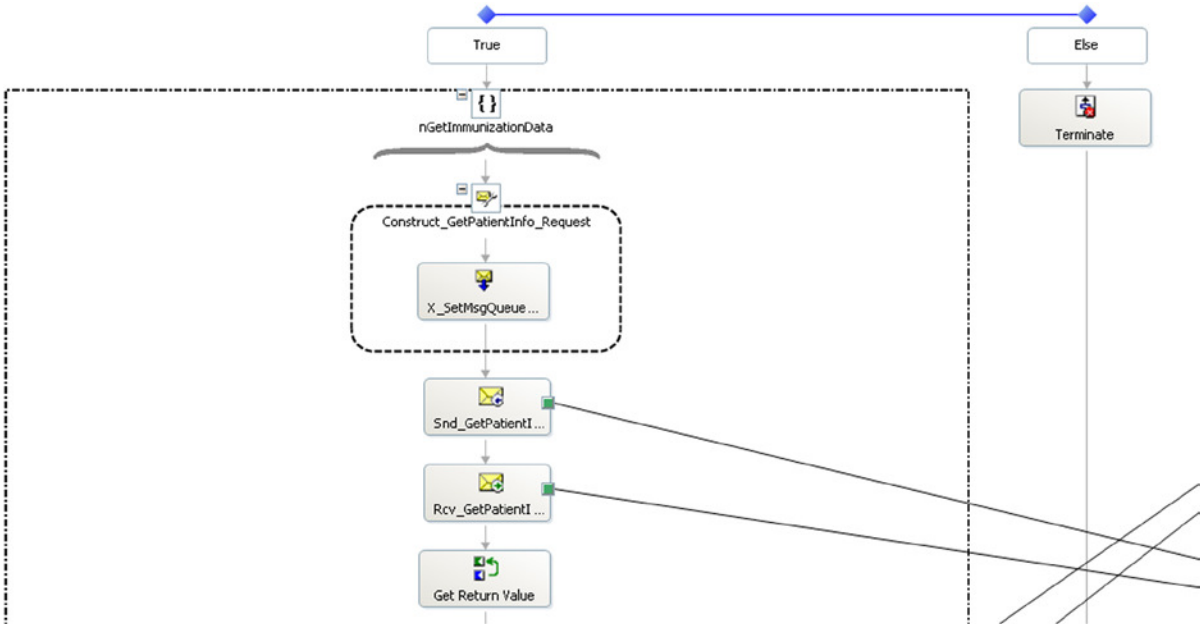
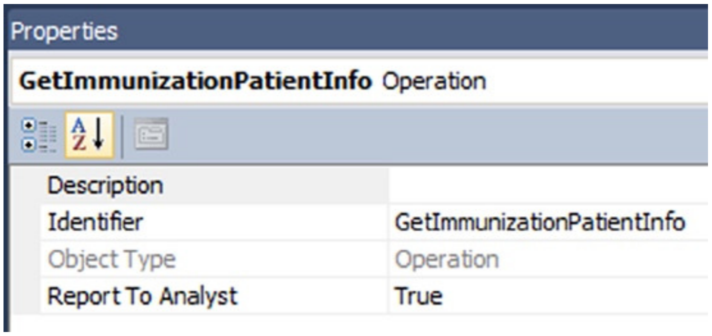


Figure 5-19. *GetImmunizationData scope*

The data was returned from the `GetImmunizationPatientInfo` request/response port, whose configuration is shown in Figure 5-20.



Properties

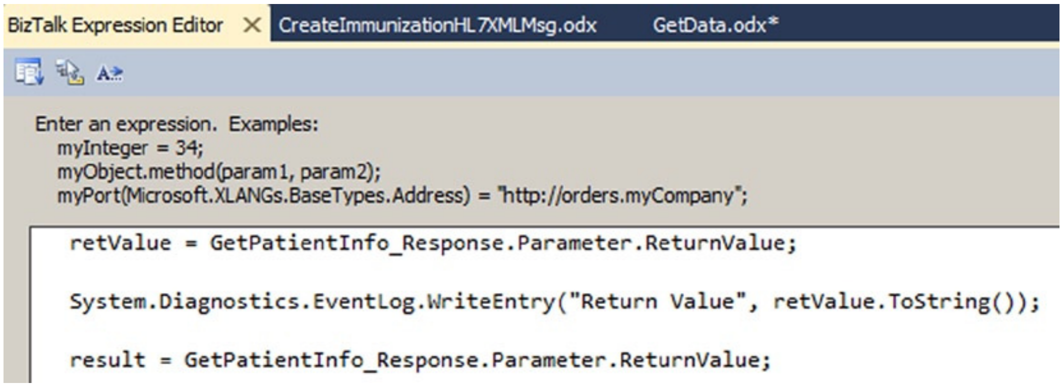
GetImmunizationPatientInfo Operation

Description	
Identifier	GetImmunizationPatientInfo
Object Type	Operation
Report To Analyst	True

Figure 5-20. *GetImmunizationPatientInfo request/response port*

Response Message Properties

When the values were returned from the stored procedure, the `Get Return Value` Expression shape contained the code to write the step to the EventLog, as shown in Figure 5-21.



```
Enter an expression.  Examples:
   myInteger = 34;
   myObject.method(param1, param2);
   myPort(Microsoft.XLANGs.BaseTypes.Address) = "http://orders.myCompany";
```

```
retValue = GetPatientInfo_Response.Parameter.ReturnValue;

System.Diagnostics.EventLog.WriteEntry("Return Value", retValue.ToString());

result = GetPatientInfo_Response.Parameter.ReturnValue;
```

Figure 5-21. *Response values*

The three stored procedures that follow are all duplications of this theme, with their ports and code modified to reflect the stored procedure being called. At times in the process, additional checks were performed to determine whether or not data was being returned before the process continued. This step was performed because if at any time there existed no data for a given stored procedure, there would be no reason to continue since the overall message being created would be incomplete. An incomplete HL7 file should not be accepted by anyone expecting valid HL7 data.

The Stored Procedures

Let's now look at the other stored procedures involved.

BizTalk.GetImmunizationNextOfKin

The second stored procedure called in this orchestration was the `BizTalk.GetImmunizationNextOfKin` stored procedure. Its data was mapped primarily to the NK1 Segment, which would be next in line in the HL7 file, and some RXR & RXA fields. The data was requested and returned the same as the first stored procedure. We again see a step to confirm that data was returned in this step.

BizTalk.GetImmunizationPatientInfoVaccine

Without exiting the branch in the orchestration, we next called the stored procedure named `BizTalk.GetImmunizationPatientInfoVaccine`, whose data was subsequently mapped to the `RXA`, `RXR, and ORC` Segments of the HL7 file. This was followed by the now standard method to update the EventLog with this step's success or failure.

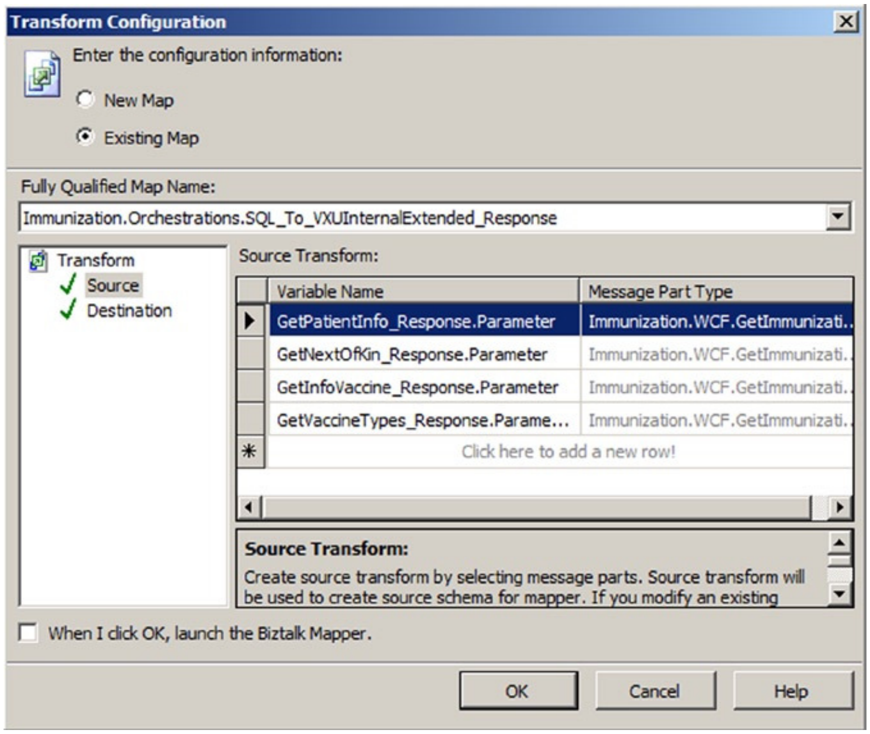
After this, the orchestration performed another check to determine if data was returned. Assuming the last stored procedure returned data successfully, we moved to the last part of the orchestration where we gathered and assembled the data in an .xml format.

BizTalk.GetImmunizationVaccineTypes

The final stored procedure called was `BizTalk.GetImmunizationVaccineTypes`, whose data almost exclusively ended up mapping the fields in the OBX Segments. A little later we will go into detail about these segments because they greatly affect some of the work we had to do in order to get them to output correctly. As was the case with the previous stored procedures, the step that followed the data being returned was a step to log the event's success or failure in the EventLog.

Creating the Final Map

The last true message building step in this orchestration was to essentially assemble the data mapped from the four stored procedures into one cohesive XML-producing map. This step, `Construct_VXUInternalExtendedResponse`, was configured as shown in Figure 5-22 and Figure 5-23.



Fully Qualified Map Name: Immunization.Orchestrations.SQL_To_VXUInternalExtended_Response

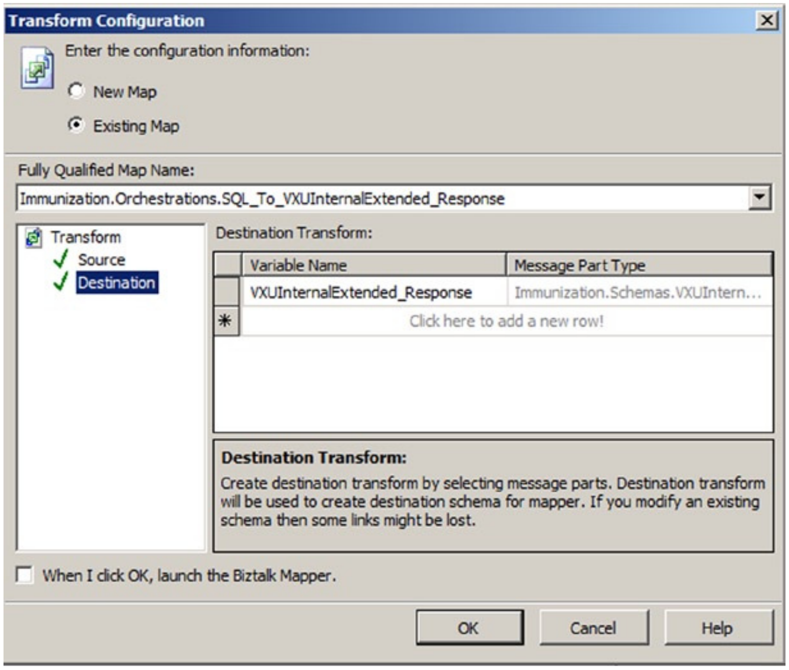
Source Transform:

Variable Name	Message Part Type
GetPatientInfo_Response.Parameter	Immunization.WCF.GetImmunizati...
GetNextOfKin_Response.Parameter	Immunization.WCF.GetImmunizati...
GetInfoVaccine_Response.Parameter	Immunization.WCF.GetImmunizati...
GetVaccineTypes_Response.Parame...	Immunization.WCF.GetImmunizati...
Click here to add a new row!	

Source Transform:
Create source transform by selecting message parts. Source transform will be used to create source schema for mapper. If you modify an existing

☐ When I click OK, launch the Biztalk Mapper.

Figure 5-22. Construct_VXUInternalExtendedResponse source



Fully Qualified Map Name: Immunization.Orchestrations.SQL_To_VXUInternalExtended_Response

Destination Transform:

Variable Name	Message Part Type
VXUInternalExtended_Response	Immunization.Schemas.VXUIntern...
Click here to add a new row!	

Destination Transform:
Create destination transform by selecting message parts. Destination transform will be used to create destination schema for mapper. If you modify an existing schema then some links might be lost.

☐ When I click OK, launch the Biztalk Mapper.

Figure 5-23. Construct_VXUInternalExtendedResponse destination

The destination schema in this map was our canonical schema; the purpose for using this was to allow a map where the transformations of data as needed would be done to prepare it for the final HL7 map XML format. The schema appeared to be largely a 1:1 map, so the mappings were mostly straight across. This also made it an ideal place for the transformations since the mapping was otherwise simple.

The map that came out of this transformation, which provided the base XML for mapping to our final HL7 schema, was very large. For that reason, we utilized the tabs at the bottom to group the mappings—in this case by stored procedure results—as a matter of best practice. The resulting initial map was still very complex, as you can see in Figure 5-24.

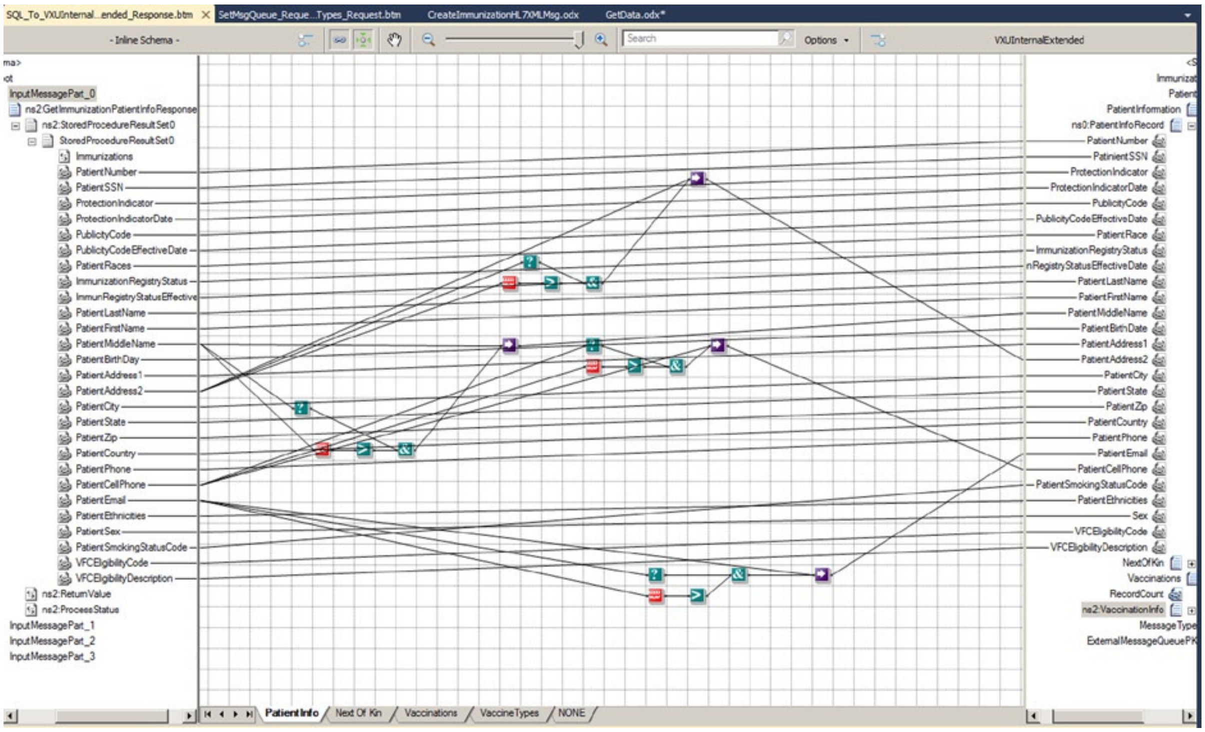


Figure 5-24. *SQL_To_ Construct_VXUInternalExtendedResponse map*

Notice the PatientInfo tab at the bottom, as well as the functoids checking for existence of data before any transformation is performed. What is shown above is only one quarter of the processes we ran. The NextOfKin map was next, as shown in Figure 5-25.

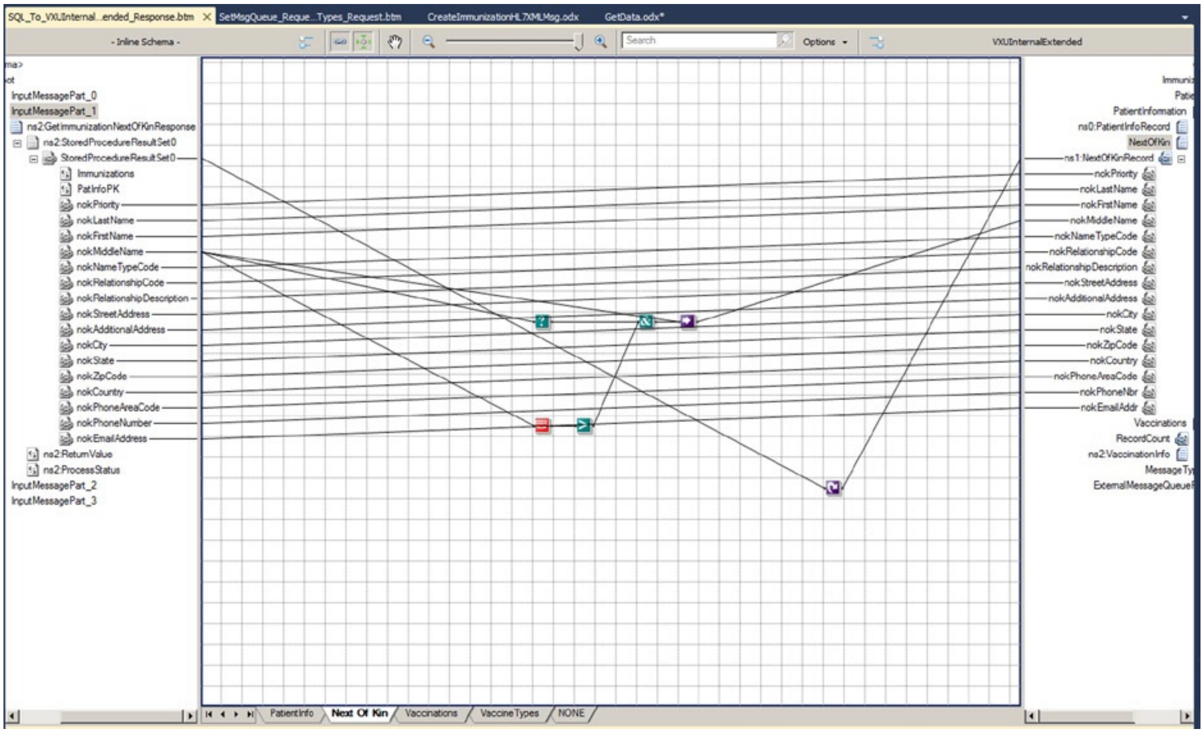


Figure 5-25. *NextOfKin map*

The remaining maps contained few transformations of data, just straight across data copies since the schema both contain the same fields. The remaining two sections of the canonical schema grouped data as it would be utilized by future maps, so they were named `VaccinationInfo` and `Vaccination`, as you can see in Figure 5-26.

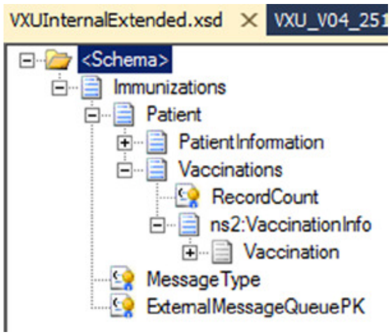


Figure 5-26. *Vaccinations and Vaccination messages*

■ **Note** The MessageType and ExternalMessageQueuePK nodes carried information through the maps, which was critical to being able to report status of the process back to the original table for monitoring by the BizTalk Administrator.

The final tab on this map contained a transformation to carry forward values for the ExternalMessageQueuePK, which was necessary for updating the ExternalMessageQueue table as progress was made.

1. A hard-coding for the MessageType value to be modified to reflect the status in the process at this step.
2. A loop over the RecordCount from the initial stored procedure to keep track of how many times the process has run.

Once this process completed, the modified XML file was sent to the next orchestration, which eventually molded it into the HL7 file, the CreateImmunizationHL7XMLMsg.odx orchestration.

The remainder of the orchestration was designed to catch any exceptions thrown by the GetImmunizationdata process and update the ProcessingStatus so the overall status of the process could be both updated and followed as it ran. Those parts of the orchestration were configured as shown in Figure 5-27.

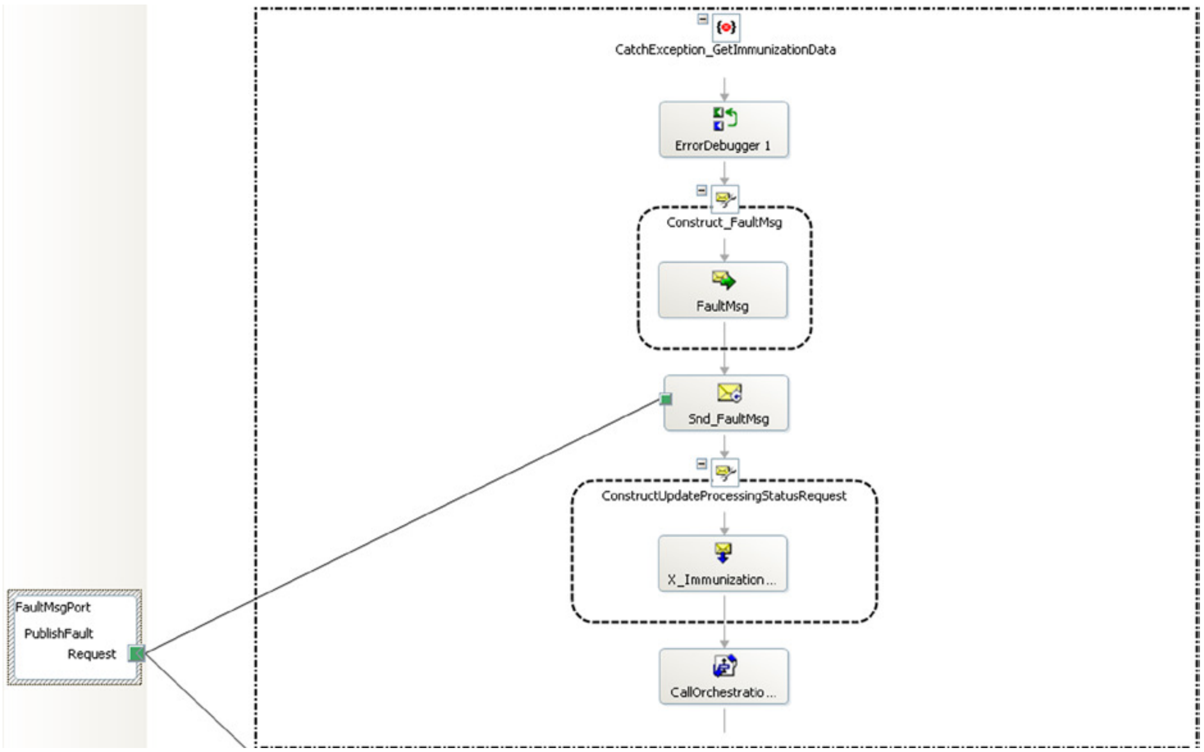


Figure 5-27. *CatchException_GetImmunizationData*

■ **Note** At the end of each of these sections of the orchestration, a call was made to another stored procedure. In this case, it was one that updated the ProcessingStatus in the ExternalMessageQueue table, part of an error logging and system notification process we decided upon when this solution was designed.

Error Handling

The final Exception Catch section was configured as shown in Figure 5-28; following the steps from the previous section, it performed much the same function. They both sent their data to the FaulMsgPort, where they were consumed and published in the error reporting process. This section caught the exception thrown.

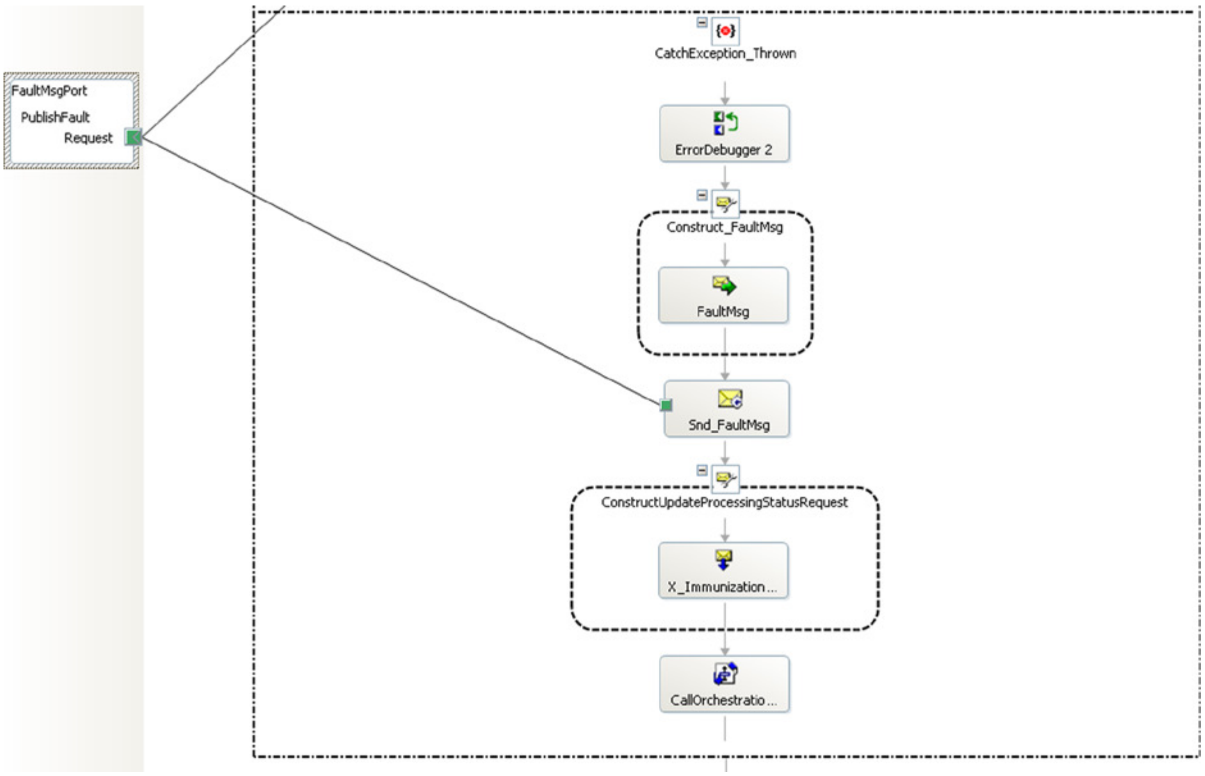


Figure 5-28. *CatchExceptionThrown*

CreateImmunizationHL7XMLMsg

While the `GetData` orchestration built the initial XML from the data gathered by the stored procedures, the `CreateImmunizationHL7XMLMsg` orchestration was where we made it a true HL7 file. The nature of this file provided two challenges over other HL7 files: there was looping over records to provide multiple iterations of segments, as well as groupings of those segments in a very specific manner.

The first step of this orchestration took in the canonical-formatted file, set a variable (`ExtMsgQueuePK = VXU_Extended.ExternalMessageQueuePK`), and told the orchestration how to make an entry in the Eventlog.

The next section of the orchestration was where the transformations started to happen from the canonical XML format to what was assembled into the final HL7 file. The steps in the orchestration are shown in Figure 5-29.

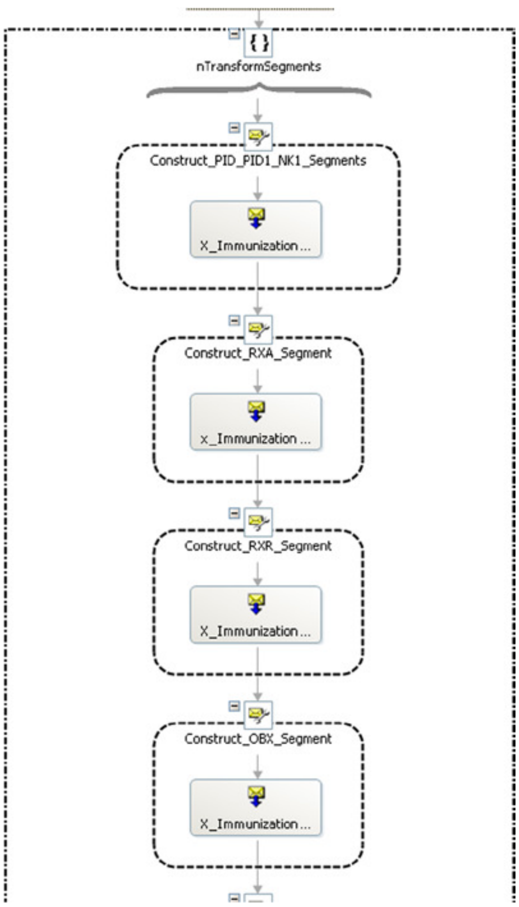


Figure 5-29. *Transform segments*

Each step took data in smaller bites, assembling it as we did before, but this time it went from the canonical schema to the true HL7 fields. As each of these maps were completed, they were tested using the TestMap functionality built into BizTalk. What we ended up with were segments of data ready for assembly into the HL7 format. The maps can get very complex, which is another reason to employ the best practice of separating the data into groups and assembling it once the data has been put into the final format.

An example of one of the more complex maps is the OBX Segment. Because of its multipurpose use, as well as the need for looping through the data to create multiple iterations of the OBX Segment containing every changing data, the map was very complex, as you can see in Figure 5-30.

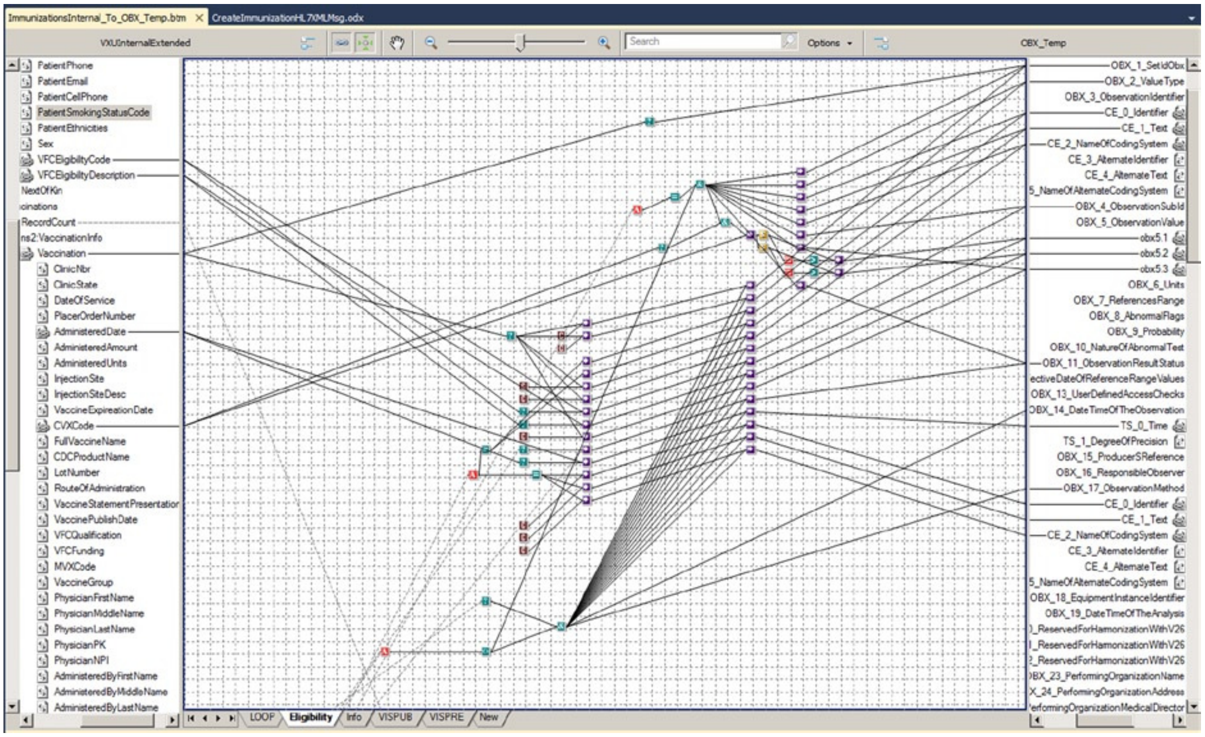


Figure 5-30. *One tab of the canonical to OBX Segment map*

Notice the multiple tabs at the bottom of the map; there is an actual pattern to the HL7 Immunization file's OBX Segments, whether there is one or there are four, ten or twenty. The pattern is as follows:

- ***OBX-1***: This version of the OBX contains eligibility information of the person being vaccinated.
- ***OBX-2***: This segment defines the type of vaccine being administered.
- ***OBX-3***: This version of the segment tells when the vaccine information given to the patient was published.
- ***OBX-4***: This last type of OBX Segment defines when the vaccine information document was presented to the patient.

As mentioned, sometimes in the test cases there were as many as ten OBX segments used to define vaccination information for a single vaccination event. How would this be for a single vaccination? Some vaccinations contain multiple vaccines; in the case of the vaccine ***DTaP***, it would produce 10 OBX Segments. The first would tell the eligibility of the patient, and the subsequent nine segments would profile each of the three vaccines that make up the multi-vaccine vaccination. So the final makeup of the OBX portion of this Immunization Registry message would look like Listing 5-14.

Listing 5-14. OBX Segments

```
OBX|1|CE|64994-7^Vaccine funding program eligibility category^LN|1|V03^VFC eligible -
Uninsured^HL70064||||||F|||20120701|||VXC40^Eligibility captured at the immunization level^CDCPHINVS
OBX|2|CE|30956-7^vaccine type^LN|2|107^DTaP^CVX||||||F
OBX|3|TS|29768-9^Date vaccine information statement published^LN|2|20070517||||||F
OBX|4|TS|29769-7^Date vaccine information statement presented^LN|2|20120814||||||F
OBX|5|CE|30956-7^vaccine type^LN|3|89^Polio^CVX||||||F
OBX|6|TS|29768-9^Date vaccine information statement published^LN|3|20111108||||||F
OBX|7|TS|29769-7^Date vaccine information statement presented^LN|3|20120814||||||F
OBX|8|CE|30956-7^vaccine type^LN|4|45^Hep B, unspecified formulation^CVX||||||F
OBX|9|TS|29768-9^Date vaccine information statement published^LN|4|20120202||||||F
OBX|10|TS|29769-7^Date vaccine information statement presented^LN|4|20120814||||||F
```

Notice the OBX-2, OBX-5, and OBX-8 share identical formats if not much of the same data. The rest of the pattern has OBX-3, OBX-6, and OBX-9 being much the same, and OBX-4, OBX-7, and OBX-10 matching in format as well. This is shown to illustrate the grouping at the bottom of the map ("canonical to OBX Segment map"). This also makes it easier to loop through the data that is returned from the stored procedures.

This also serves to illustrate the level of complexity with this solution. In addition to the OBX, the RXA, RXR, and ORC Segments also came out as multiples that had to be grouped correctly.

Once each of these maps was created and the data was awaiting assembly, the next section of the orchestration performed this assembly. However, with the level of complexity involved, we needed to utilize an XSLT style sheet to perform some of the looping. It was created by first using the BizTalk mapper tool, and then modifying the resulting XSLT file manually to enable both the necessary looping as well as the ordering of the segments, since the groupings matter.

Final Steps

The section of the orchestration shown in Figure 5-31 shows the creation of the HL7-formatted XML, as well as the step where this file was output to a folder for review. This step will likely be eliminated in Production, but it provided valuable data when troubleshooting.

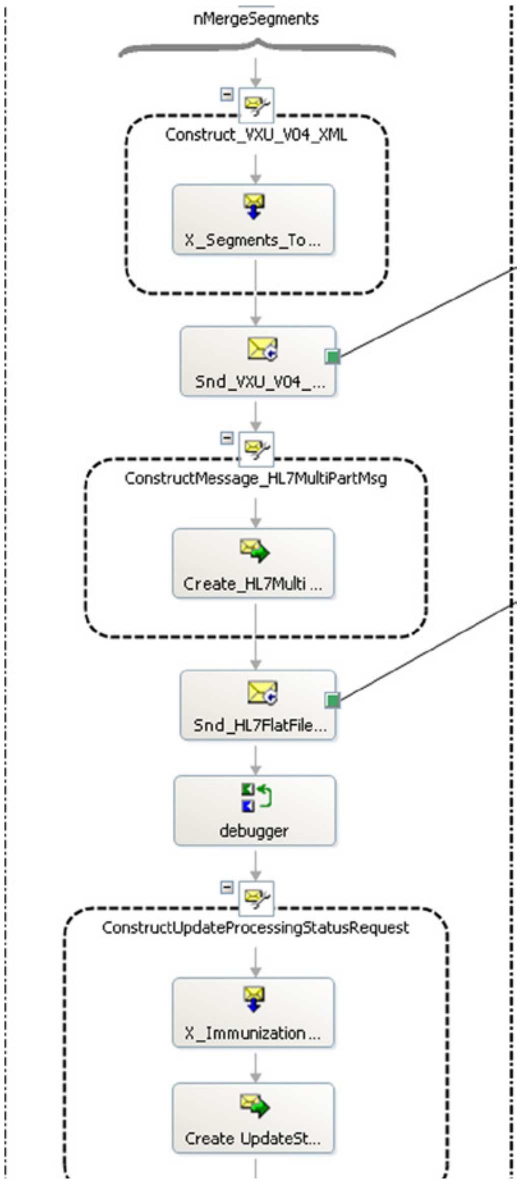
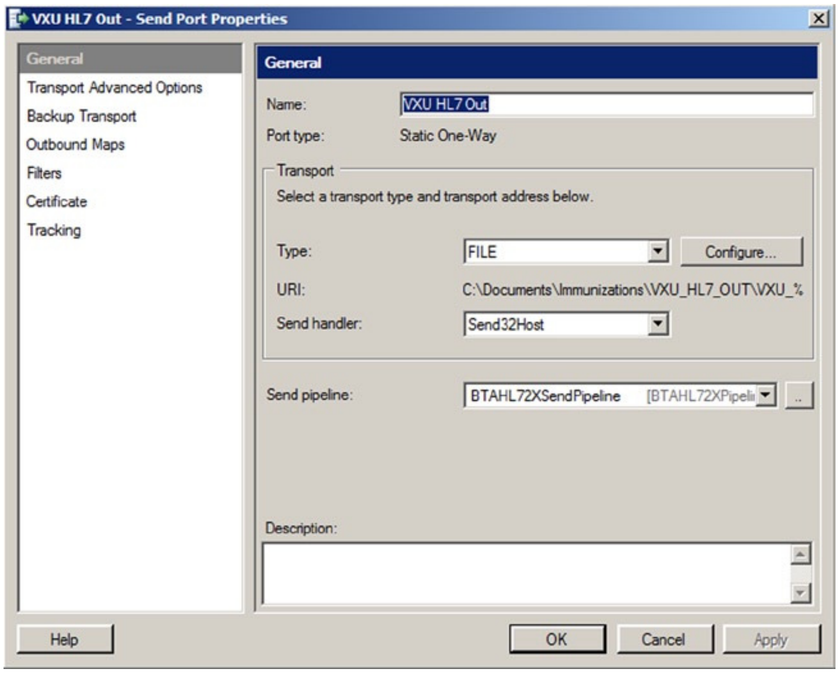


Figure 5-31. *Merge segments*

The section labeled `ConstructMessage_HL7MultiPartMsg` was where the ExpressionEditor was used to define the non-variable sections of the MSH Segment that was previously shown in this chapter; the rest was filled in by the BizTalk Administration Console via Trading Partner data. The Business Rules Engine could also be used, but if we end up sending this file to multiple endpoints, the Trading Partner functionality will enable this to stay manageable.

At this point, what we had was a large amount of HL7-formatted XML data. Getting this into the correct pipe-delimited format was performed in the ***VXU HL7 Out Send port***, via use of the send pipeline `BTAHL72XSendPipeline`, which was configured in this case on the send port itself, as shown in Figure 5-32.


VXU HL7 Out - Send Port Properties

General

Name: VXU HL7 Out

Port type: Static One-Way

Transport

Select a transport type and transport address below.

Type: FILE

URI: C:\Documents\Immunizations\VXU_HL7_OUT\VXU_%

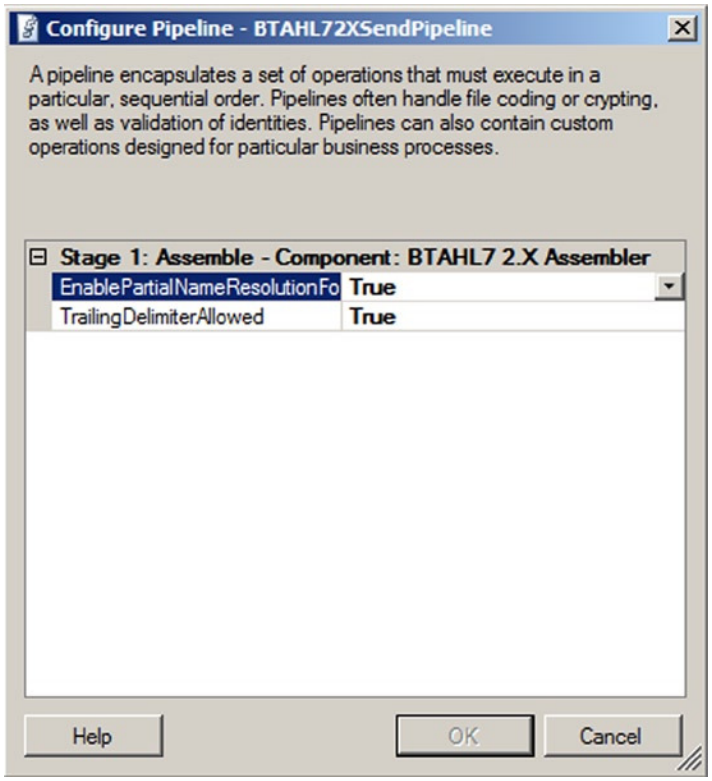
Send handler: Send32Host

Send pipeline: BTAHL72XSendPipeline [BTAHL72XPipeli

Description:

Figure 5-32. *Send port configuration*

The resulting output came out in sequence as a pipe-delimited file. Further configuration of the pipeline itself allowed the file to be created with trailing empty fields, which is acceptable to the HL7 testing body. Since the port was created using a binding file via the BizTalk Administration Console, that is where we configured the pipeline, as shown in Figure 5-33.


Configure Pipeline - BTAHL72XSendPipeline

A pipeline encapsulates a set of operations that must execute in a particular, sequential order. Pipelines often handle file coding or crypting, as well as validation of identities. Pipelines can also contain custom operations designed for particular business processes.

Stage 1: Assemble - Component: BTAHL7 2.X Assembler

EnablePartialNameResolutionFo	True
TrailingDelimiterAllowed	True

Figure 5-33. *Pipeline configuration section, accessed by ellipses next to the pipeline name*

The balance of the orchestration consisted of exception handling. The first of the CatchExceptions was for segment transformation, as shown in Figure 5-34.

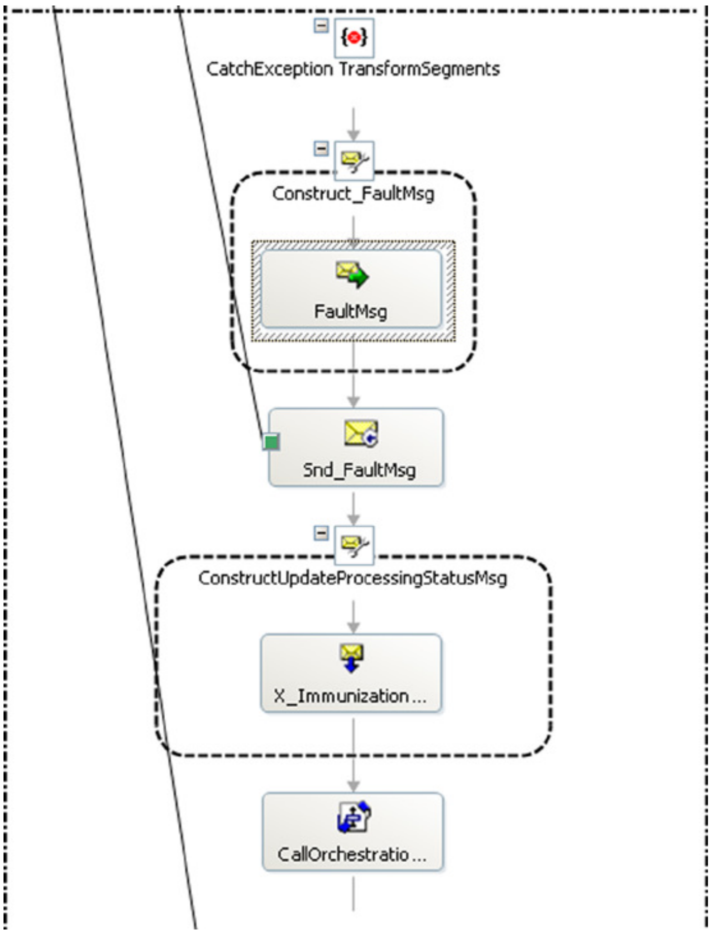


Figure 5-34. *Exception handling*

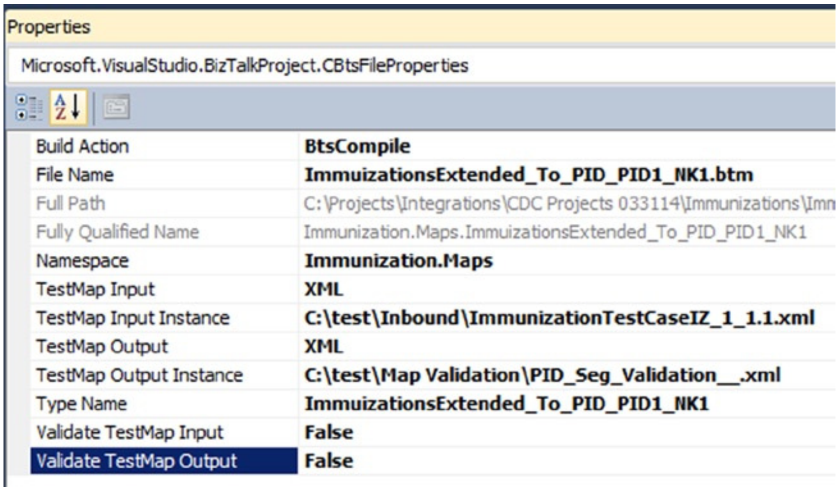
The Basic Configuration Is the Same

Construct the fault message, send the fault message, update the Status Update message, and then call the stored procedure to commit the changes to the fault message info to ExternalMessageQueue table. Again, while not profiled in detail, this table is what is monitored by the BizTalk solution for changes; any changes in the table result in it being scanned for rows of data that meet the criteria that define that row as an Immunization Registry message event.

Testing

It goes without saying that testing at every point along the way in this project was critical. Waiting until final assembly to try to ascertain where the problem lay is an invitation to a long day; the best practice, as was evident through this solution's development, was to test incrementally each and every step of the way. That included after building a schema, a map, even a section of an orchestration—even if it meant including outputting to a hard file that would eventually be removed from the process.

For example, once the maps were finished, they were tested incrementally, utilizing the TestMap functionality found in the properties section of the map's profile (see Figure 5-35). In configuring the testing of the map, we initially turned off the validation as we were simply looking to validate the basic output. We made manual tweaks to the maps based on output, but if we turned on validation and the map failed, it would not produce an output against which we could visually compare.


Properties

Microsoft.VisualStudio.BizTalkProject.CBtsFileProperties

Build Action	BtsCompile
File Name	ImmuizationsExtended_To_PID_PID1_NK1.btm
Full Path	C:\Projects\Integrations\CDC Projects 033114\Immunizations\Imm
Fully Qualified Name	Immunization.Maps.ImmuizationsExtended_To_PID_PID1_NK1
Namespace	Immunization.Maps
TestMap Input	XML
TestMap Input Instance	C:\test\Inbound\ImmunizationTestCaseIZ_1_1.1.xml
TestMap Output	XML
TestMap Output Instance	C:\test\Map Validation\PID_Seg_Validation__.xml
Type Name	ImmuizationsExtended_To_PID_PID1_NK1
Validate TestMap Input	False
Validate TestMap Output	False

Figure 5-35. *Validate TestMap Output settings*

As seen in Figure 5-36, the TestMap Input and TestMap Output were profiled before right-clicking on the map name and choosing Test Map.

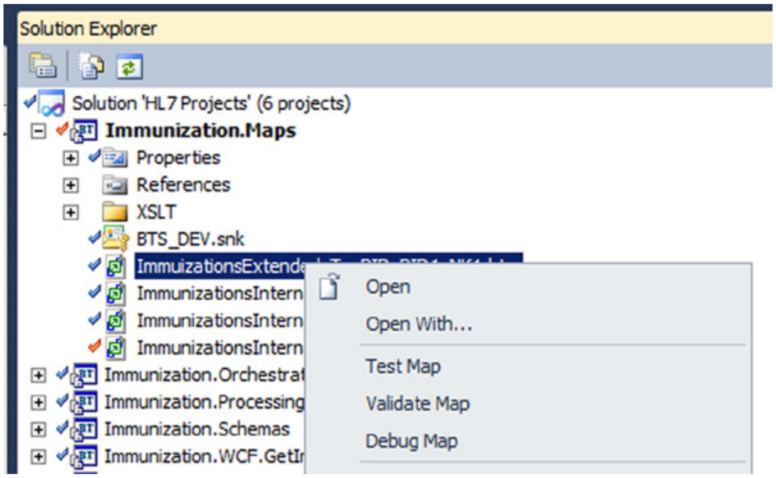


Figure 5-36. *Test map*

As we ran through the map validation process, we determined that making corrections using this process was much quicker than if we relied upon the built-in validation.

Maps and schemas are tested easily enough via the properties in each. With an orchestration, in order to check what was being created and passed outside the map, it really proved best to create a send port with a file output, usually XML.

Once the project was migrated to the BizTalk Administration Console, testing needed to be performed again, preferably with an end-to-end process. The advantages of testing through the console included access to the Windows Event Viewer/EventLog, the BizTalk Group Hub, and in our case even the ESB Management Console. Each of these can provide errors which can be worked through either locally or, if need be, via the BizTalk Use Groups.

■ **Note** This real-life scenario was written and contributed by Tom Banaski. Tom is a BizTalk developer specializing in healthcare data integrations. He was first exposed to BizTalk when it was shown in comparison to a competitor's product. For the last 13 years, Tom has worked with pharmacy and healthcare claims, and has been focusing on data integration for the last 7 years. For the last 2 years, Tom has been working with BizTalk developing EDI & HL7 solutions for Practice Velocity, Inc. One solution is for the government's Meaningful Use program. He has also worked on non-standard BizTalk solutions for proprietary client requests.

Using HL7 V.2.5.1 Queries

Besides sending HL7 Version 2.5.1 messages, there are occasions when we need to request patient and clinical data from other systems. Let's take a look at a few message types.

Query by Parameter Message Types

The Query by Parameter message types are the most common queries. Table 5-5 shows the Query by Parameter message types and their events.

Table 5-5. *QBP: Query by Parameter Message Types*

Event	Description	Msg Structure
Q11	QBP - Query by parameter requesting an RSP segment pattern response	QBP_Q11
Q13	QBP - Query by parameter requesting an RTB - tabular response	QBP_Q13
Q15	QBP - Query by parameter requesting an RDY display response	QBP_Q15
Q21	QBP - Get person demographics	QBP_Q21
Q22	QBP - Find candidates	QBP_Q21
Q23	QBP - Get corresponding identifiers	QBP_Q21
Q24	QBP - Allocate identifiers	QBP_Q21
Q25	QBP - Personnel Information by Segment Query	QBP_Q21
Q31	DBP - Dispense History	QBP_Q11
Z73	Information about Phone Calls	QBP_Z73
Z75	Tabular Patient List	QBP_Q13

(*continued*)

Table 5-5. (*continued*)

Event	Description	Msg Structure
Z77	Tabular Patient List	QBP_Q13
Z79	Dispense Information	QBP_Q15
Z81	Dispense History	QBP_Q11
Z85	Pharmacy Information Comprehensive	QBP_Q11
Z87	Dispense Information	QBP_Q15
Z89	Lab Results History	QBP_Q11
Z91	Who Am I	QBP_Q13
Z93	Tabular Dispense History	QBP_Q13
Z95	Tabular Dispense History	QBP_Q13
Z97	Dispense History	QBP_Q15
Z99	Who Am I	QBP_Q13

We will take a look at the Query/Response transaction, ***QBP^Q11*** - Query by Simple Parameter and ***RSP^K11*** - Segment Pattern Response.

QBP^Q11 - Query by Simple Parameter

This query message, which supports a Segment Pattern Response, contains these segments:

- MSH - Message Header
- QDP - Query Parameter Definition
- RCP - Response Control Parameters
- DSC - Continuation Pointer

This transaction is used to query for clinical data which can include Lab Results, Continuity of Care Document (CCD), Pharmacy Dispense Information, and Alerts. Supported query parameters are Local Patient identifier and demographic information, a date range, and the maximum number of records to be returned (specified in the RCP Segment).

QPD Parameters

Table 5-6 shows the required query parameters.

Table 5-6. *QPD Required Query Parameters*

Seq	Description	Data Type	Expected Value (examples)
QPD.1	Message Query Name	CE	If LABS then map value Z10^LABS^HL70471. If RAD then map value Z10^RAD^HL70471. If CCD then map value Z20^CCD^HL70471. If ALERT then map value Z30^ALERT^HL70471. If RDR then map value Z81^DispenseHistory^HL70471.
QPD.2	Query Tag	ST	Unique Identifier
QPD.3	User Parameters	Varies	Patient MRN

Listing 5-15 shows an example query that would return the Pharmacy Dispense Information (RDR) message.

Listing 5-15. Example of Query by Parameter Query Transaction with Pharmacy Dispense Information (RDR) as the Expected Response

```
MSH|^~\&|PCR|Gen Hosp|PIMS||199811201400-0800||QBP^Z81^QBP_Q11|ACK9901|P|2.4||||||||
QPD|Z81^Dispense History^HL70471|Q001|555444222111^^^MPI^MR||19980531|19990531|
RCP|I|999^RD|
```

RSP^K11 - Segment Pattern Response to Query by Parameter

The Query Response message is based upon using the HL7 V2.5.1 ***ORU*** Transaction. The ***RSP^K11*** supports a Segment Pattern Response to the ***QBP^Q11*** and contains the MSH, MSA, ERR, QAK, and QPD variable content segments, and the DSC. A standard or site-defined response may use this trigger event or may specify a unique trigger event value in its Conformance Statement. If a unique trigger event value is chosen for a site-defined response, that value must begin with Z.

Listing 5-16 is an example of what the response might look like for the query in Listing 15-15.

Listing 5-16. Response Example for Query

```
MSH|^~\&|PIMS|Gen hosp|PCR||199811201400-0800||RSP^Z82^RSP_Z82|8858|P|2.4||||||||MSA|AA|ACK9901|
QAK|Q001|OK| Z81^Dispense History^HL70471|4|
QPD|Z81^Dispense History^HL7nnnn|Q001|555444222111^^^MPI^MR||19980531|19990531|
PID|||555444222111^^^MPI^MR||Everyman^Adam||19600614|M||C|2222 HOME STREET^^Oakland^CA^94612||^^^^^5
55^5552004|^^^^^555^5552004||||343132266|||N|||||||||
ORC|RE||89968665|||||199805121345-0700|||77^Hippocrates^Harold^H^III^DR^MD||^^^^^555^ 5552104|||||
RXE|1^BID^^19980529|00378112001^Verapamil Hydrochloride 120 mg TAB^NDC |120||mgm||||||||||||||||||
|||||||
RXD|1|00378112001^Verapamil Hydrochloride 120 mg TAB^NDC |199805291115-
0700|100|||1331665|3||||||||||||||||
RXR|PO||||
ORC|RE||89968665|||||199805291030-0700|||77^Hippocrates^Harold^H^III^DR^MD||^^^^^555^555-5001|||||
RXE|1^^D100^^20020731^^^TAKE 1 TABLET DAILY --GENERIC FOR CALAN SR|00182196901^VERAPAMIL HCL ER TAB
180MG ER^NDC |100||180MG|TABLET SA|||G|||0|BC3126631^CHU^Y^L||213220929|0|0|19980821|||
RXD|1|00182196901^VERAPAMIL HCL ER TAB 180MG ER^NDC |19980821|100|||213220929|0|TAKE 1 TABLET DAILY
--GENERIC FOR CALAN SR||||||||||||
RXR|PO||||
ORC|RE||235134037||||||199809221330-0700|||8877^Hippocrates^Harold^H^III^DR^MD||^^^^^555^555-
5001||||||RXD|1|00172409660^BACLOFEN 10MG TABS^NDC|199809221415-0700|10|||235134037|5|AS
DIRECTED||||||||||||
RXR|PO||||
ORC|RE||235134030||||||199810121030-0700|||77^Hippocrates^Harold^H^III^DR^MD||^^^^^555^555-5001||||||
RXD|1|00054384163^THEOPHYLLINE 80MG/15ML SOLN^NDC|199810121145-0700|10|||235134030|5|AS
DIRECTED||||||||||||
RXR|PO
```

Summary

In this chapter, you took a look at three different topics. You learned about mapping Version 2.x to the HL7 CDA and how you could leverage the HL7 V.2 data types. You were provided with a real-life scenario for creating meaningful use Immunization messages. And you also learned about the basics of using the HL7 V.2.5.1 Request/Response Query by Parameter.

In the next chapter, you will learn about implementing best practices for using the HL7 Accelerator.

CHAPTER 6

Future Directions

In the previous chapter, you learned about advanced topics with the HL7 Accelerator, including HL7-Version-2-to-Version-3 conversion, Meaningful Use implementation, and using HL7 Version 2.x queries.

In this chapter, you are going to look into the near future and learn about HL7 FHIR, the newest HL7 standard that is currently in Draft Standard for Trial (DSTU).

FHIR® in BizTalk

The HL7 organization is developing a new standard called Fast Health Interoperable Resources, commonly referred to as FHIR and pronounced "fire." It is the next generation standards framework, and it combines the best features of HL7 Version 2, Version 3, and the CDA product lines. FHIR will leverage the latest web standards and apply a tight focus on implementability. In this section, you are going to learn about Release 1.1 of FHIR.

Note The source for some of the information contained within this chapter is `hl7.org/fhir`.

What is FHIR®

The following is an excerpt from *Introducing HL7 FHIR*.

> *FHIR solutions are built from a set of modular components called "Resources". These resources can easily be assembled into working systems that solve real world clinical and administrative problems at a fraction of the price of existing alternatives. FHIR is suitable for use in a wide variety of contexts – mobile phone apps, cloud communications, EHR-based data sharing, server communication in large institutional healthcare providers, and much more.*

Source: `www.hl7.org/implement/standards/fhir/summary.html`

Why Is FHIR Better?

There are so many improvements over the existing standards. Some of these improvements are the following:

- Provides for a ***strong focus on implementation***, which makes it fast and easy to implement.
- There are ***multiple implementation libraries*** and they include examples that will kick-start development.
- The ***FHIR specification is free for use***. There are no restrictions.
- Provides ***out-of-the-box interoperability***. The base resources can be used as is. And they can also be adapted for local requirements.
- Provides for an ***evolutionary development path with HL7 Version 2 and CDA standards***. They can coexist and leverage each other.
- Has a ***strong foundation in web standards, such as*** XML, JSON, HTTP, Atom, OAuth, and more.
- Provides support for ***RESTful*** architectures.
- Provides for ***seamless exchange of information*** using messages or documents.
- Concise and ***easily understood specifications.***

You can read more about these and other improvements on the FHIR web site at `www.hl7.org/implement/standards/fhir/summary.html`.

What Problems Does FHIR Solve?

If you are familiar with the HL7 Version 3 Product Suite, the HL7/ASTM Continuity of Care Document (CCD), and the CDA, then you know about the complexity of working with them in BizTalk.

FHIR aims to satisfy the needs covered in the previous primary HL7 interoperability standards. These include Version 2, Version 3, and the CDA. Let's look at *few of the key similarities* for each.

HL7 Version 2

You can read more about the HL7 Version 2 to FHIR comparison on the FHIR web site at `www.hl7.org/implement/standards/fhir/comparison-v2.html`.

- FHIR provides support for event-driven messaging.
- FHIR provides for granularity.
- FHIR provides for update behavior.
- FHIR provides for optionality and profiles.

HL7 Version 3 (and ISO 21090)

You can read the full HL7 Version 3 comparison on the FHIR web site at `www.hl7.org/implement/standards/fhir/comparison-v3.html`.

- FHIR uses codes for attributes.
- FHIR supports NULL flavors.
- FHIR supports context conduction.

CDA

The complete comparison for the CDA can be found on the FHIR web site at `www.hl7.org/implement/standards/fhir/comparison-cda.html`.

- FHIR is clinical document-focused.
- FHIR supports both machine and human readability.

■ **Tip** The documentation for FHIR on the HL7 F\HIR web site is very compressive and very easy to understand. If you want to learn more about FHIR and see the full comparison to the previous HL7 standards, it is highly recommended that you browse through the documentation at `http://hl7.org/implement/standards/fhir/`.

What Does This Mean for BizTalk?

The HL7 FHIR Workgroup has removed the complexity of working with HL7 messaging. Besides easy to read and understand documentation, there are FHIR schemas, samples of each message type, and a complete REST API. Let's start off with a look at the FHIR schemas.

FHIR Schemas

FHIR provides schemas that pertain to the following resources types. Each resource type has its own schema.

■ **Note** You will see the term "resource" used throughout these sections. Resources are not schemas.

Clinical Resource Types

The following are some of the common clinical resources types.

General

- AdverseReaction: Records an unexpected reaction suspected to be related to the exposure of the reaction subject to a substance.
- AllergyIntolerance: Indicates the patient has a susceptibility to an adverse reaction upon exposure to a specified substance.
- CarePlan: Describes the intention of how one or more practitioners intend to deliver care for a particular patient for a period of time, possibly limited to care for a specific condition or set of conditions.
- FamilyHistory: Significant health events and conditions for people related to the subject relevant in the context of care for the subject.
- Procedure: An action that is performed on a patient. This can be a physical thing like an operation, or less invasive event like counseling or hypnotherapy.
- Questionnaire: A structured set of questions and their answers. The questionnaire may contain questions, answers, or both. The questions are ordered and grouped into coherent subsets, corresponding to the structure of the grouping of the underlying questions.

Medications

- Medication: Primarily used for identification and definition of medication, but also covers ingredients and packaging.
- MedicationPrescription: An order for both the supply of the medication and the instructions for administration of the medicine to a patient.
- MedicationAdministration: Describes the event of a patient being given a dose of a medication. This may be as simple as swallowing a tablet or it may be a long-running infusion. Related resources tie this event to the authorizing prescription, and the specific encounter between patient and health care practitioner.
- Immunization: Immunization event information.
- ImmunizationRecommendation: A patient's point-of-time immunization status and recommendation with optional supporting justification.

Administrative

The following are a few of the common administrative resource types.

Attribution

- Patient: Demographic and additional administrative information about the person receiving care. This includes social services, dietary services, and the tracking of personal health and exercise data.
- RelatedPerson: Information about a person that is involved in the care for a patient, but who is not the target of healthcare, nor has a formal responsibility in the care process.
- Practitioner: A person who is directly or indirectly involved in the provisioning of healthcare.
- Organization: A formally or informally recognized grouping of people or organizations formed for the purpose of achieving some form of collective action. Includes companies, institutions, corporations, departments, community groups, healthcare practice groups, etc.

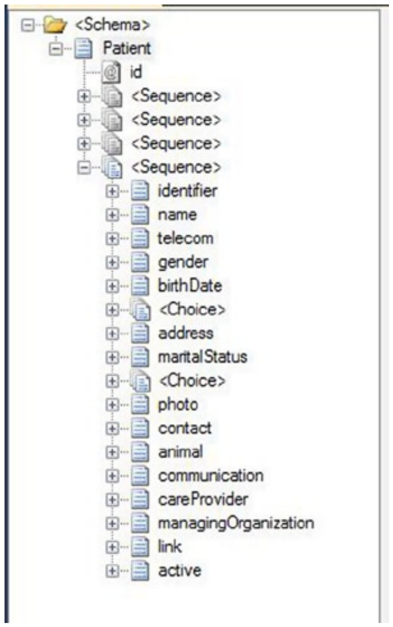
Workflow Management

- Encounter: An interaction between a patient and healthcare provider(s) for the purpose of providing healthcare service(s) or assessing the health status of a patient.
- Alert: Prospective warnings of potential issues when providing care to the patient.
- Supply: A request for something, and provision of what is supplied.
- Order: A request to perform an action.
- OrderResponse: A response to an order.

This is but a small sampling of the resources available with FHIR. Let's take a look at how the FHIR resources relate to the schemas in BizTalk.

Patient Resource

Let's take a closer look at the Patient resource (schema). Figure 6-1 shows the Patient schema.



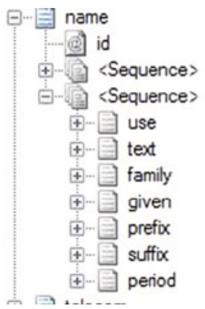
```
<Schema>
  Patient
    id
    <Sequence>
    <Sequence>
    <Sequence>
    <Sequence>
      identifier
      name
      telecom
      gender
      birthDate
      <Choice>
      address
      maritalStatus
      <Choice>
      photo
      contact
      animal
      communication
      careProvider
      managingOrganization
      link
      active
```

Figure 6-1. *Patient schema*

In Figure 6-1, note that there is an id attribute and four sequence groups. Let's take a closer look at the last sequence group, which is expanded.

■ **Note** All of the BizTalk FHIR schemas share the same structure. The only difference being the content contained within the last sequence group.

As you can see, the sequence group consists of multiple records. If you expand the name record, as shown in Figure 6-2, you will see more sequence groups.



```
name
  id
  <Sequence>
  <Sequence>
    use
    text
    family
    given
    prefix
    suffix
    period
```

Figure 6-2. *Name record*

The last sequence group is expanded. If you expand the records within this sequence group, as shown in Figure 6-3, you will see that each record has an id attribute, sequence group, and value attribute.

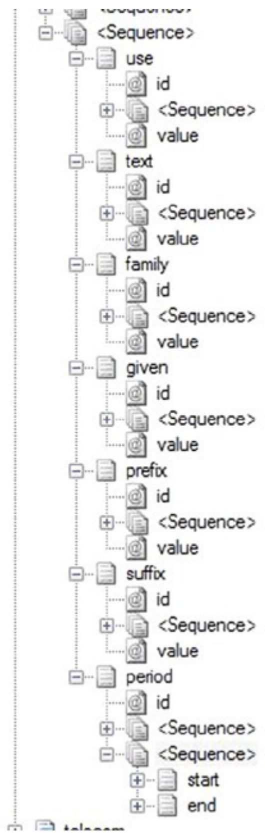


Figure 6-3. *Expanded view of name sequence group*

The value attribute field in each record stores the value for the record; for example, the family record value attribute contains the patient's last name. This attribute field can have a restriction based upon the FHIR data types. In the case of the patient's last name, it would be the FHIR string data type.

■ **Note** FHIR data types contain two categories; simple/ primitive types and complex types. You can learn more about the FHIR data types by visiting `http://hl7.org/implement/standards/fhir/datatypes.html`.

Now that you have learned a little about FHIR resources, let's take a look at the fhir-atom schema and how it is used within BizTalk.

The FHIR-ATOM Schema

This schema is what BizTalk will be receiving and sending to FHIR servers and client applications. The message can either be JSON or XML. Figure 6-4 shows the `fhir-atom.xsd`. The root node is always named “feed”.

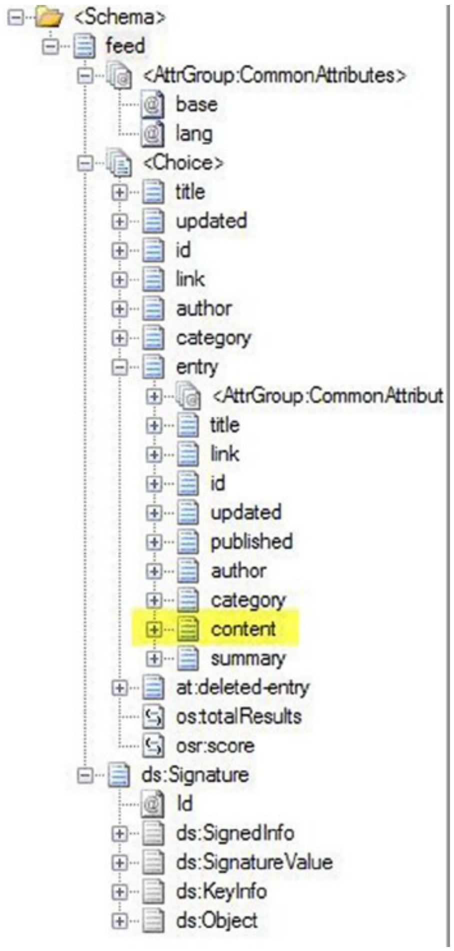


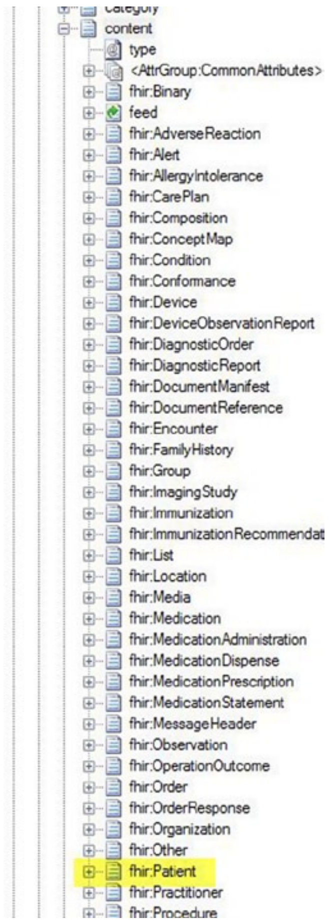
Figure 6-4. *fhir-atom schema*

Looking at Figure 6-4, you will notice that the payload is in the Choice group. The cardinality for this group is minimum of three and unbounded for the maximum. If you expand the content record, you will see all the resources defined in the FHIR Resource Index.

■ **Tip** The complete FHIR Resource Index can be viewed on the FHIR web site at `www.hl7.org/implement/standards/fhir/resourcelist.html`.

Content Record

Figure 6-5 shows the Patient resource, highlighting the content record.



```
category
content
  type
  <AttrGroup:CommonAttributes>
  fhir:Binary
  feed
  fhir:AdverseReaction
  fhir:Alert
  fhir:AllergyIntolerance
  fhir:CarePlan
  fhir:Composition
  fhir:ConceptMap
  fhir:Condition
  fhir:Conformance
  fhir:Device
  fhir:DeviceObservationReport
  fhir:DiagnosticOrder
  fhir:DiagnosticReport
  fhir:DocumentManifest
  fhir:DocumentReference
  fhir:Encounter
  fhir:FamilyHistory
  fhir:Group
  fhir:ImagingStudy
  fhir:Immunization
  fhir:ImmunizationRecommendat
  fhir:List
  fhir:Location
  fhir:Media
  fhir:Medication
  fhir:MedicationAdministration
  fhir:MedicationDispense
  fhir:MedicationPrescription
  fhir:MedicationStatement
  fhir:MessageHeader
  fhir:Observation
  fhir:OperationOutcome
  fhir:Order
  fhir:OrderResponse
  fhir:Organization
  fhir:Other
  fhir:Patient
  fhir:Practitioner
  fhir:Procedure
```

Figure 6-5. *Content record expanded*

The `fhir:Patient` record shown in Figure 6-5 is the Patient schema. You could map a Patient record directly to this record.

Moving On

Now that you know a little about the structure of the FHIR schemas in BizTalk, let's create a simple BizTalk project for the FHIR schemas. This will give you a chance to see the schemas first hand.

STEPS FOR CREATING A BIZTALK FHIR SCHEMA PROJECT

1. Create one Visual Studio solution named Fhir.POC with one BizTalk project.
2. The project, which will be named Fhir.Schemas, will contain the FHIR schemas.
   a. Download the Schema Zip file from www.hl7.org/implement/standards/fhir/downloads.html.

■ **Tip** While you are at it, you can also download the FHIR Book (e-book), FHIR Specification, All Valuesets, and both XML and JSON examples (having a local copy will make it easier to work with). If you want to see how to work with FHIR in .Net, download the Reference Implementations

---

   b. Extract the schemas from the zip file into the project.
   c. You will need to make a few modifications to `fhir-base.xsd` in order to build the schema project.
      i. First, remove `<xs:include schemalocation="fhir-all.xsd"/>`Next, change `<xs:complextype name="Resource.Inline">` to the following:

```
<xs:complextype name="Resource.Inline">
<xs:sequence>
<xs:any processcontents="lax"/>
</xs:sequence>
</xs:complextype>
```

3. Sign and build the schema project.

■ **Tip** If you decide you want to develop a complete FHIR application, you can use a public FHIR test server. The easiest one to work with is Spark at http://spark.furore.com/.

The Spark server was developed by Furore (http://fhir.furore.com/). The FHIR-ATOM Schema can also be mapped to both HL7 Version 2.x and Version 3 Schemas. This will provide you with backward compatibility.

It is my understanding that Microsoft Healthcare is starting to invest in the use of FHIR with BizTalk, along with Azure, Office365, Healthvault, and CRM.

Summary

In this chapter, you were introduced to HL7 FHIR. You have learned how the FHIR schemas are structured. You also created a BizTalk project for the FHIR schemas. In the next chapter, you will learn all about *Best Practices* for using HL7 with BizTalk.

CHAPTER 7

Best Practices for HL7 with BizTalk

In previous chapter, you learned about the future directions for HL7 with BizTalk. In this chapter, you will learn about the best practices for working with HL7 in BizTalk, which help improve the design and maintenance of applications. You will learn about best practices in the following areas:

- Managing HL7 Message Schemas
- Dynamic Data Validation
- Managing Trading Partners
- Message Exchange Pattern
- Ordered Delivery
- Optimization for Performance

Managing HL7 Message Schemas

BizTalk provides all HL7 message schemas as part of the HL7 Accelerator installation. The number of schemas is dependent on the following three properties:

- Message Type
- Message Trigger Event
- HL7 Messaging Standard Version

For each of these properties, BizTalk has one message schema (XSD) defined. In addition to these schemas, there are other schemas like the message header (MSH) and acknowledgment (ACK) message schemas. Managing so many schemas in a BizTalk environment can be a very painful task, especially when you need to deal with different versions, or different trigger events of one or more message types.

In order to manage these schemas in a more controlled manner and avoid deployment of all the schemas to a BizTalk environment, you can follow the best practices covered in this chapter, which will not only reduce the number of schemas deployed in your environment but will also help in maintaining them easily.

Deploying MSH and ACK Message Schemas to Their Own BizTalk Application

MSH and ACK message schemas are required to be deployed only once in entire BizTalk environment; in other words, there can't be two MSH or ACK message schemas for messages to parse successfully by the HL7 Disassembler. *It is a recommended practice to deploy and maintain these schemas in their own solution and BizTalk application.* These schemas are hardly ever touched after first deployment. Keeping them separate will allow you to not worry about these while deploying and maintaining other message schemas.

Using One HL7 Message Standard Version

The BizTalk HL7 Accelerator supports HL7 versions from HL7 V2.1 to V2.6. HL7 messaging standards with higher versions are backward compatible with lower versions, so Version 2.6 is compatible with Version 2.1. So if you are receiving messages of different versions in a BizTalk environment then *it is recommended to only use higher versions of a schema in BizTalk and use a pipeline component to change the version in the message header before passing the message to the HL7 Disassembler.*

■ **Note** When creating HL7 messages, you should use the schema of the specific version you want to create the message for, unless you are sure about the differences between the latest version and the version you want to create the message for.

Using Minimum Required Message Trigger Event Schemas

Similar to different schema versions, there are also have many message trigger events for a particular message type. BizTalk has a different schema for each trigger event of a particular message type; however, the message structure for each of these schemas is nearly same. So all ADT message trigger events (A01- A62) can be managed with one or two message schemas deployed. *It is recommended to install and deploy only the minimum required message schemas and use a pipeline component to change the trigger event before parsing the message.* This reduces the number of schemas to be deployed to a very low number, highly improving the maintenance of these schemas.

Changing Namespace for Customized Message Schemas

It's highly unlikely that messages received will be in the exact format as defined by the HL7 messaging standard, and some customization to BizTalk message schemas is always required. *It is recommended to change the target namespace of these customized schemas and use a trading partner to use these schemas.* This allows you to isolate the changes from the original message schemas and their namespace. Please refer to Chapter 4 to customize a message schema.

Dynamic Data Validation

Dynamic data validation is about validating HL7 message content with dynamic data (the data that changes frequently). Although the HL7 Accelerator provides the ability to validate message content and format it using a message schema definition, this method of validation has following limitations:

- This is a static validation because the schema definition once deployed can't be changed dynamically at runtime. Any change to the schema requires redeployment of the message schema.
- Redeployment of artefacts affects time-to-market: it slows down the process of delivering the new requirements.
- Many times HL7 messages received are not as per the HL7 messaging standard. Also, changing the message content is not feasible. In such cases, schema validation produces many more errors, as anticipated.

These limitations are huge in a large and changing environment as there is much pressure to deliver the ever-changing requirements quickly. To overcome these limitations, it is recommended to use the approach outlined in the following sections.

Resolving Message Structure Validation Issues

The HL7 Accelerator always performs message structure validation irrespective of whether message body validation is turned on or not. Message structure validation validates the segment and the field's structure, whether the expected number of fields or subfields is present or not. For example, the OBX5 field by definition is a simple text (ST) format data type, and if in a message the OBX5 field has subfields in it, then the message structure validation will fail with a data type error. These data type errors can be difficult to find at times, and one of the best ways is to test the messages. In order to resolve these issues, *it is recommended to resolve all such message structure validation by either making the message source change the message according to the definition or change the message schema to accommodate fields.*

Using a Database to Maintain Dynamic Data

In order to maintain data that changes frequently, *it is highly recommended to use a database to maintain such data, which allows easy updates to the data at runtime without redeploying the artefacts*. The dynamic data can also include HL7 codes such as LOINC codes, SNOMED codes etc., against which you need to validate the message content.

Using Business Rules Engine (BRE) for Message Content Validation

This ensures that message content is valid as per your requirements by implementing business rule engine (BRE) policies. BRE policies can be invoked from pipeline components or from orchestration to perform message content validation. There are number of advantages of using BRE policies for message body validation:

- BRE policies can use a database to refer to dynamic data that changes frequently, which allows you to add or change the data at runtime without any redeployment.
- Validation will be based upon specific requirements rather than entire message content.

In order to implement content validation using BRE, you need to turn off message body validation from the HL7 Party Configuration Explorer and create a BRE policy referring to the database. Please refer to `http://msdn.microsoft.com/en-us/library/aa560118.aspx` to create a BRE policy for accessing a database. The BRE policy can be invoked from a pipeline component to return the validation errors as part of the message acknowledgment or from an orchestration depending on the requirement.

Managing Trading Partners (Parties)

In a large environment, you may receive or send HL7 messages to and from many different trading partners, with each of them having a different MSH3 or MSH5 in the HL7 message. As you learned in Chapters 3 and 4, you need to set up a BizTalk trading partner (party) for many functionalities of the HL7 Accelerator with respect to MSH3 and MSH5 values. Setting up a trading partner for each and every sending or receiving application can be very unmanageable especially if the environment is large. You can have smaller number of trading partners and manage them easily by using the following best practice.

Group Trading Partners

Generally a trading partner is created for each different MSH3 (Sending Application) value in message you receive, so if you receive HL7 messages having three different MSH3 values, you need to create three trading partners to apply the configuration you need. If each of these sending applications sends messages of the same type or which require similar customization and configuration, *it is recommended to create only one trading partner and have one pipeline component to change the MSH3 value in the component before passing the message to the HL7 Disassembler.* This way you can group the trading partners and reduce them significantly.

The same applies to trading partners created for the receiving application (MSH5). For example, if you need to batch outgoing HL7 messages for different destinations, you must create a trading partner for each MSH5 to create the required batch schedule. *It is recommended to create one trading partner with one schedule and use the same trading partner in MSH5 when sending the message.*

Message Exchange Patterns

There are specific message exchange patterns for HL7 messaging. The following are the key patterns you use in HL7 in BizTalk.

Fire-And-Forget Pattern

In the fire-and-forget pattern, the sending application keeps sending the messages without waiting for the message acknowledgment. This pattern performs better; however, it has the disadvantage that messages can't be traced back in case of any issues.

Request-Response Pattern

In Chapter 2, we discussed two message processing exchange patterns, original mode and enhanced mode.
Each mode differs in the way it sends acknowledgment to the sending application; however, the similarity is that in both the sending application waits for acknowledgment before sending the next message. The message exchange pattern is referred to as a request-response exchange.

It is recommended to use the request-response message exchange pattern, which ensures that you can trace the message acknowledgment and does not send the next message until you receive acknowledgment. If you are the receiver, you should use the request-response pattern with enhanced mode; in other words, use an intermediary storage to store messages, such as MSMQ or SQL Server, and return the acknowledgment before processing the messages. Storing messages in MSMQ or any other queuing system also helps in achieving ordered delivery of the messages.

■ **Note** HL7 has transmission patterns. These patterns include specialized patterns called query, batch, polling, and event replay. These HL7 patterns are equivalent to the MEP.

Example

In order to demonstrate the above best practices with an example, consider the following scenario:

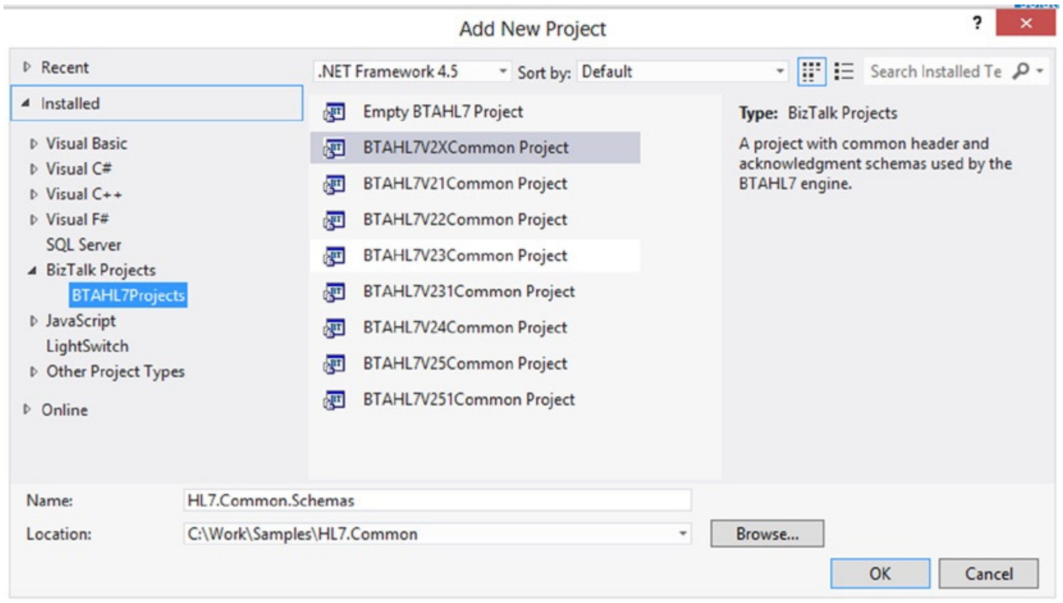
- BizTalk receives an ADT version 2.3 message with different trigger events.
- BizTalk validates the message and returns the acknowledgment.
- BizTalk stores the message in intermediate storage, such as MSMQ.

The steps to implement this scenario are covered in the following sections. Some of these steps are similar to those provided in the Chapter 4 scenario.

Building a Common Project

The first step in implementing the scenario is to create a common project that contains a message header (MSH) and acknowledgment (ACK) schema. This will only need to be done once in the BizTalk environment for all HL7-based applications. You can skip this step if you already have MSH and ACK messages deployed to BizTalk.

1. Create a Visual Studio blank solution and name it HL7.Common.
2. Create a new BTAHL7V2XCommon project in the solution named HL7.Common.Schemas, as shown in Figure 7-1.



Type: BizTalk Projects

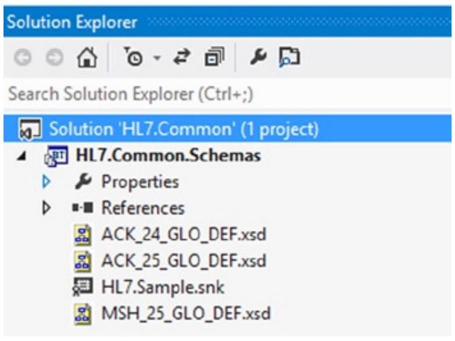
A project with common header and acknowledgment schemas used by the BTAHL7 engine.

Name: HL7.Common.Schemas

Location: C:\Work\Samples\HL7.Common

Figure 7-1. *Common project template*

3. The project in Solution Explorer will look like Figure 7-2.


Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'HL7.Common' (1 project)
- HL7.Common.Schemas
  - Properties
  - References
  - ACK_24_GLO_DEF.xsd
  - ACK_25_GLO_DEF.xsd
  - HL7.Sample.snk
  - MSH_25_GLO_DEF.xsd

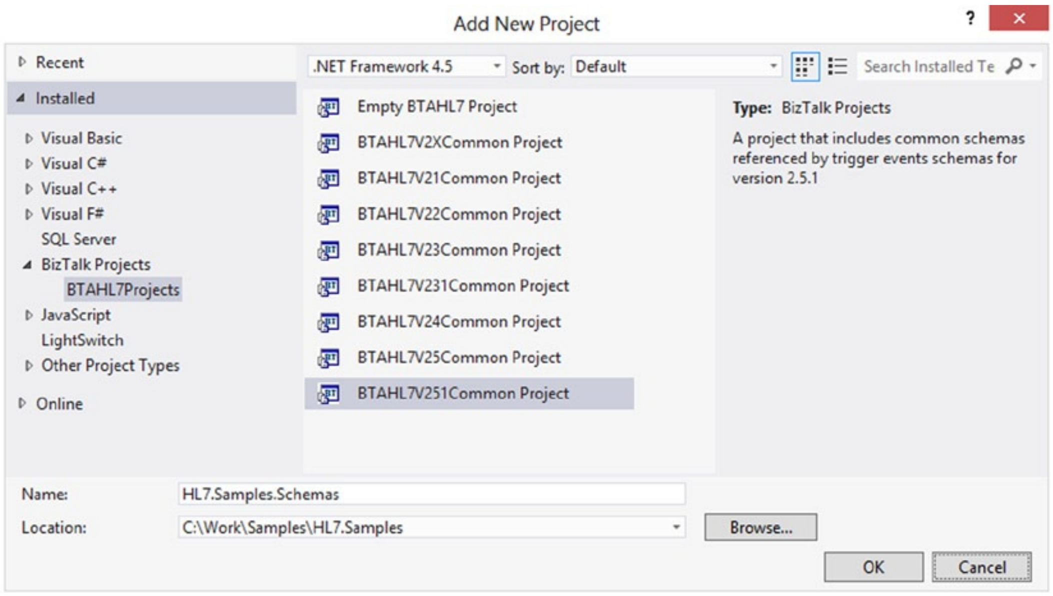
Figure 7-2. *Common project in Solution Explorer*

4. Change the BizTalk Deployment properties to deploy this project to BizTalk under the HL7.Common BizTalk application.
5. Sign the project using a strong name key and deploy it.

Building a Message Schema (ADT) Project

The next step is to create BizTalk project to add ADT message type schemas and deploy it. You will use ADT V2.5.1 schemas even though you are going to receive a V2.3 message. This will help you maintain less BizTalk schemas.

1. Create a new Visual Studio blank solution called HL7.Samples.
2. Create a BTAHL7V251Common project to the solution, as shown in Figure 7-3.



.NET Framework 4.5 Sort by: Default

- Empty BTAHL7 Project
- BTAHL7V2XCommon Project
- BTAHL7V21Common Project
- BTAHL7V22Common Project
- BTAHL7V23Common Project
- BTAHL7V231Common Project
- BTAHL7V24Common Project
- BTAHL7V25Common Project
- BTAHL7V251Common Project

Type: BizTalk Projects

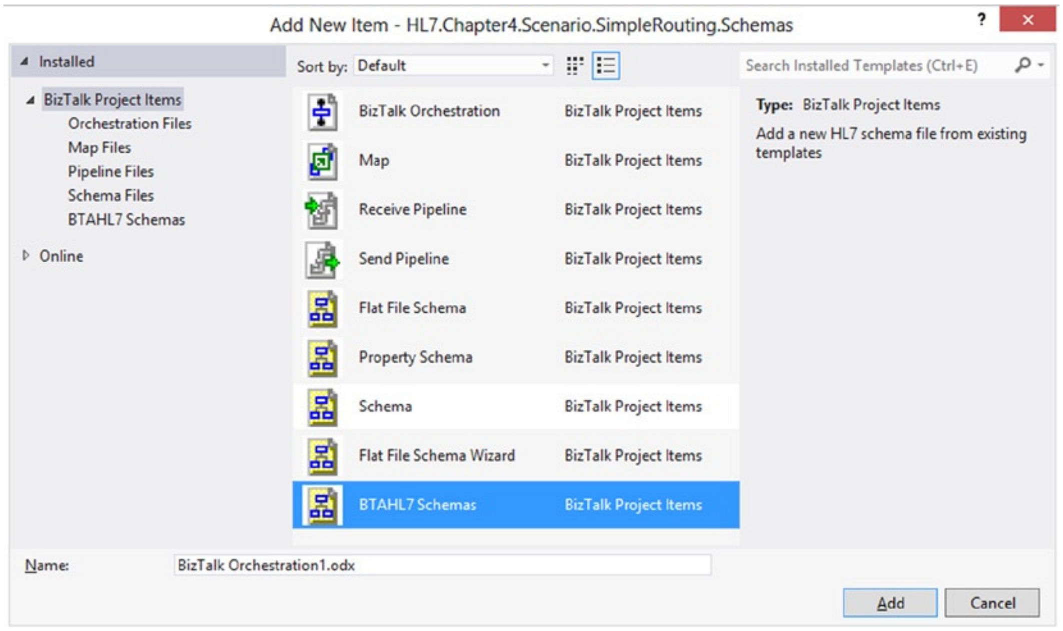
A project that includes common schemas referenced by trigger events schemas for version 2.5.1

Name: HL7.Samples.Schemas

Location: C:\Work\Samples\HL7.Samples

Figure 7-3. *Segments, DataTypes, TableValues project template*

3. The above step will add the datatypes_251, tablevalues_251, and segments_251 schemas. Now add a new item to project by pressing Ctrl+Shift+A and select BTAHL7 Schemas, as shown in Figure 7-4.


Add New Item - HL7.Chapter4.Scenario.SimpleRouting.Schemas

BizTalk Orchestration	BizTalk Project Items
Map	BizTalk Project Items
Receive Pipeline	BizTalk Project Items
Send Pipeline	BizTalk Project Items
Flat File Schema	BizTalk Project Items
Property Schema	BizTalk Project Items
Schema	BizTalk Project Items
Flat File Schema Wizard	BizTalk Project Items
BTAHL7 Schemas	BizTalk Project Items

Type: BizTalk Project Items

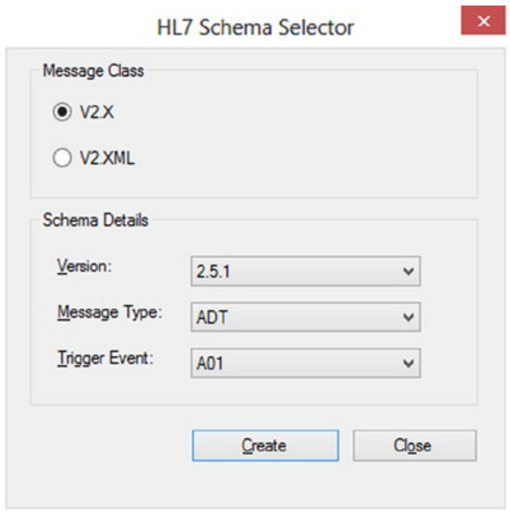
Add a new HL7 schema file from existing templates

Name: BizTalk Orchestration1.odx

Figure 7-4. *Message Type schema template*

4. Click Add, and don't worry about naming the file as next step overrides anything you select here. Once you click Add, you will get a new screen called HL7 Schema Selector to select the message type, triggering event, and version.

5. Select the message class, version, message type, and trigger event, as shown in Figure 7-5. Once you click Create, it adds schema `ADT_A01_251_GLO_DEF.xsd`.


HL7 Schema Selector

Message Class

● V2.X

○ V2.XML

Schema Details

Version:	2.5.1
Message Type:	ADT
Trigger Event:	A01

Figure 7-5. *HL7 schema type selector*

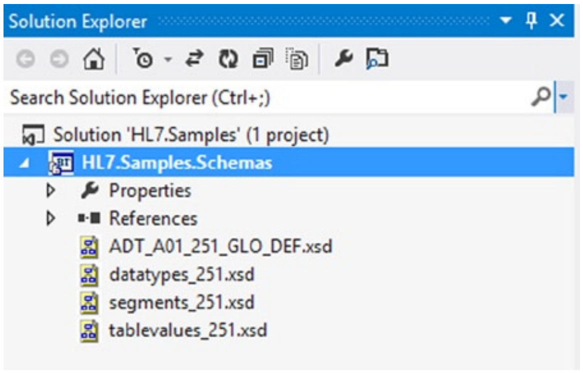
6. By default, the message schema is expected to be added to a separate project because it refers to `segments_251. xsd` from an assembly reference, as shown below from the schema file:

```
<xs:import schemaLocation="BTAHL7Schemas.segments_251"
namespace="http://microsoft.com/HealthCare/HL7/2X/2.5.1/Segments" />
```

7. Since you added the schema to same project as the segments_251 schema, change the `schemaLocation` as follows:

```
<xs:import schemaLocation="segments_251.xsd"
namespace="http://microsoft.com/HealthCare/HL7/2X/2.5.1/Segments" />
```

8. The project in Solution Explorer will look like Figure 7-6.



Solution 'HL7.Samples' (1 project)
- HL7.Samples.Schemas
  - Properties
  - References
  - ADT_A01_251_GLO_DEF.xsd
  - datatypes_251.xsd
  - segments_251.xsd
  - tablevalues_251.xsd

Figure 7-6. *Message schema project in Solution Explorer*

9. Change the default namespace from `http://microsoft.com/HealthCare/HL7/2X` with namespace `http://hl7.samples/schemas/hl7/2x` in all .xsd files.

10. Change the BizTalk deployment application to HL7.Samples in the project properties.

Building a Custom Receive Pipeline Component

This pipeline component changes incoming MSH segment fields to standardize the message before disassembling the message. It changes three MSH segment fields as follows:

1. The pipeline component changes MSH3 and MSH5 to HL7_Samples corresponding to the HL7 party you will define.

2. The pipeline component changes MSH8 to ADT^A01 always because A01 is the schema you have deployed.

3. The pipeline component changes MSH12 to 2.5.1 to standardize the message version.

The pipeline component code is shown in Listing 7-1.

Listing 7-1. Pipeline Code

```
public Microsoft.BizTalk.Message.Interop.IBaseMessage Execute(IPipelineContext pContext,
IBaseMessage pInMsg)
{
    if (pContext == null || pInMsg == null)
        throw new ArgumentNullException();

    IBaseMessageContext msgContext = pInMsg.Context;

    if (msgContext == null)
        throw new ArgumentNullException("pInMsg.Context");

    //wrap the stream into readonly seekable stream
    if (!pInMsg.BodyPart.GetOriginalDataStream().CanSeek)
    {
        ReadOnlySeekableStream seekableStream = new
        ReadOnlySeekableStream(pInMsg.BodyPart.GetOriginalDataStream());
        // Set new stream for the body part data of the input message.
        //This new stream will then used for further processing.
        // We need to do this because input stream may not support
        //seeking, so we wrap it with a seekable stream.
        pInMsg.BodyPart.Data = seekableStream;
        pContext.ResourceTracker.AddResource(seekableStream);
    }

    Stream stream = pInMsg.BodyPart.GetOriginalDataStream();
    //preserve stream position
    long position = stream.Position;
    string message;
    StreamReader sr = new StreamReader(stream);
    message = sr.ReadToEnd();

    stream.Position = position;

    //1. replace MSH3 and MSH5 with a Party defined e.g. HL7_Samples
    //2. change MSH12 to 2.5.1
    //3. Change MSH8.2 to A01

    string[] segments = message.Split('\r');
    string[] mshSegment = segments[0].Split('|');

    //These are hard coded in pipeline for demo, can be used from //pipeline properties
    mshSegment[2] = "HL7_Samples";
    mshSegment[4] = mshSegment[2];

    mshSegment[8] = "ADT^A01";
    mshSegment[11] = "2.5.1";

    //aggregate the message segments back
    segments[0] = mshSegment.Aggregate((s1, s2) => s1 + "|" + s2);
    message = segments.Aggregate((s1,s2)=> s1 + '\r' + s2);

    //change the message
    MemoryStream ms = new MemoryStream(System.Text.Encoding.ASCII.GetBytes(message));
    pInMsg.BodyPart.Data = ms;
    pContext.ResourceTracker.AddResource(ms);

    return pInMsg;
}
```

After building the pipeline component, build and install the pipeline component to GAC (Global Assembly Cache) and the `C:\Program Files (x86)\Microsoft BizTalk Server 2013\Pipeline Components` folder.

Building a Custom Receive Pipeline

This step creates a custom receive pipeline that uses the BizTalk HL7 Disassembler pipeline component in the disassembler stage, along with your custom pipeline component created in the previous step at the decode stage, as shown in Figure 7-7. Please add this pipeline to the project HL7.Samples.Pipelines in same solution.

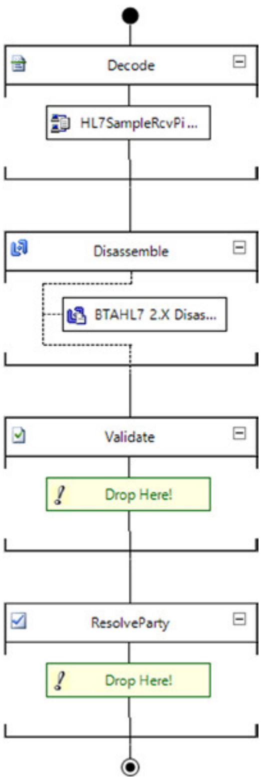


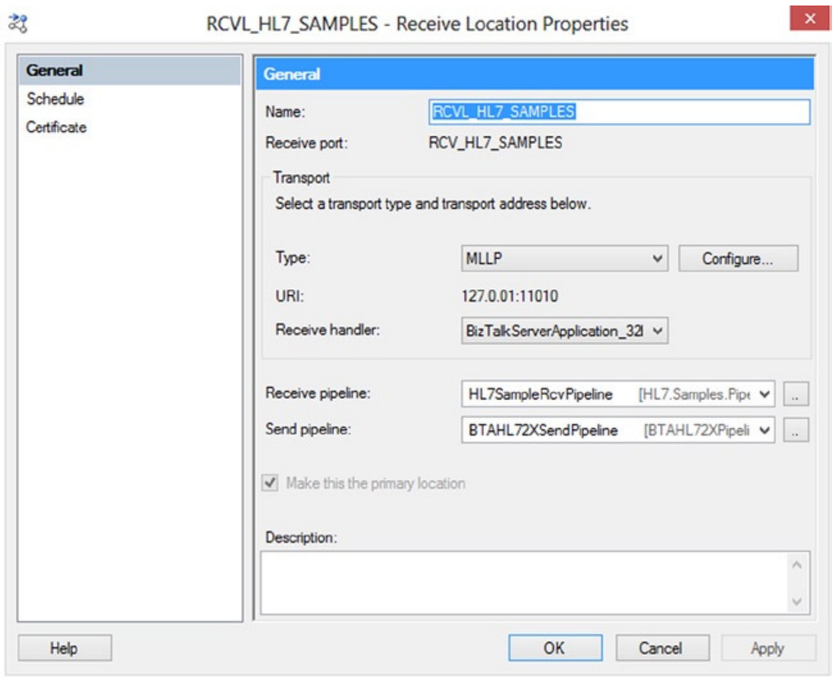
Figure 7-7. *Receive Pipeline*

Build the solution and deploy the projects to the `HL7.Samples` BizTalk application.

Configuration

After deploying the application, follow these steps to configure the application.

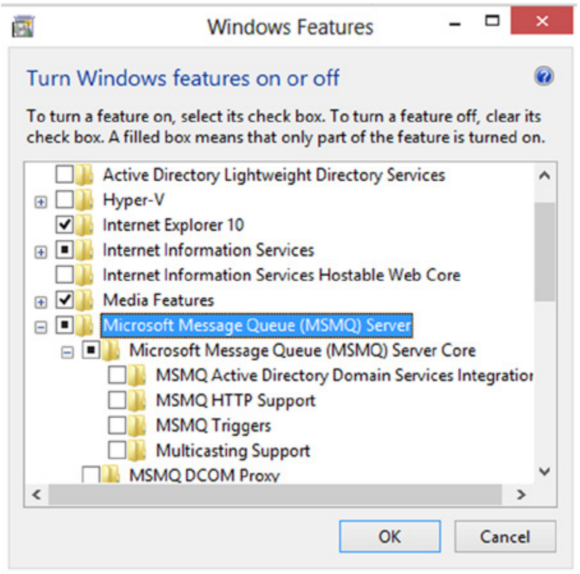
1. Create a HL7 Party with the name HL7_Samples and change the acknowledgment mode to original mode, uncheck the body validation, check the allow trailing delimiters checkbox, and change the schema namespace to `http://hl7.samples/schemas/hl7/2x`.
2. Create a two-way receive port, as shown in Figure 7-8.


RCVL_HL7_SAMPLES - Receive Location Properties

General
Name: RCVL_HL7_SAMPLES
Receive port: RCV_HL7_SAMPLES
Transport
Select a transport type and transport address below.
Type: MLLP
URI: 127.0.01:11010
Receive handler: BizTalkServerApplication_32
Receive pipeline: HL7SampleRcvPipeline [HL7.Samples.Pipe
Send pipeline: BTAHL72XSendPipeline [BTAHL72XPipeli
Make this the primary location
Description:

Figure 7-8. *Two-way receive port*

3. Configure MSMQ on the development machine if it's not already available. *You can check the Windows Add Features to see if MSMQ is configured or not, as shown in Figure 7-9.*


Windows Features

Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- ☐ Active Directory Lightweight Directory Services
- ☐ Hyper-V
- ✓ Internet Explorer 10
- ■ Internet Information Services
- ☐ Internet Information Services Hostable Web Core
- ✓ Media Features
- ■ Microsoft Message Queue (MSMQ) Server
  - ■ Microsoft Message Queue (MSMQ) Server Core
    - ☐ MSMQ Active Directory Domain Services Integratior
    - ☐ MSMQ HTTP Support
    - ☐ MSMQ Triggers
    - ☐ Multicasting Support
  - ☐ MSMQ DCOM Proxy

Figure 7-9. *MSMQ*

4. In Computer Management or Server Manager, go to Services and Applications ➤ Message Queuing. Create a private queue called hl7samples_adt with transaction set to true, as shown in Figure 7-10.

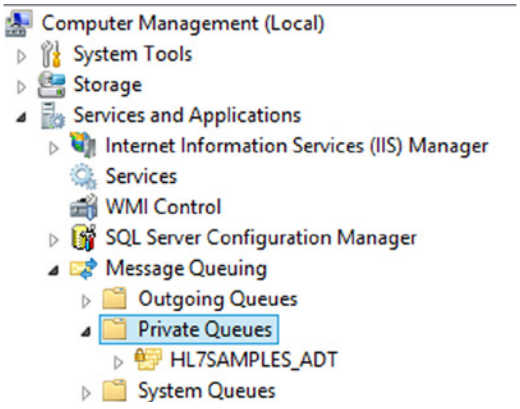
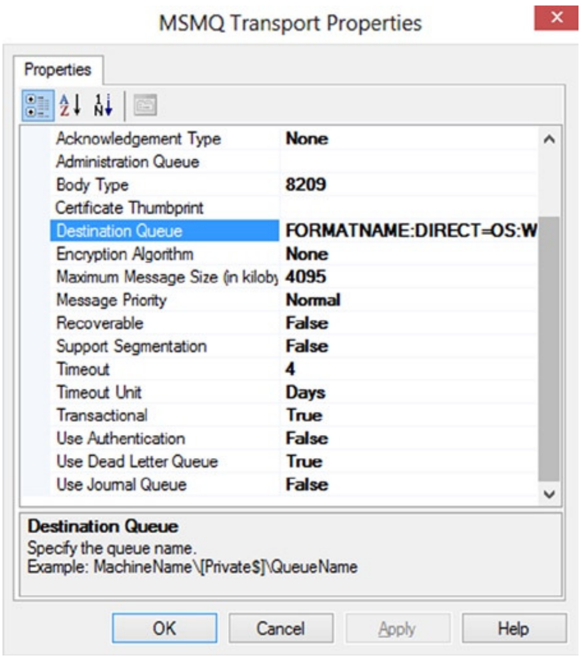

Computer Management (Local)
System Tools
Storage
Services and Applications
Internet Information Services (IIS) Manager
Services
WMI Control
SQL Server Configuration Manager
Message Queuing
Outgoing Queues
Private Queues
HL7SAMPLES_ADT
System Queues

Figure 7-10. *MSMQ Queue HL7Samples_ADT*

5. Create a one-way send port to save the incoming message MSMQ with the configuration shown in Figure 7-11.



Acknowledgement Type	None
Administration Queue	
Body Type	8209
Certificate Thumbprint	
Destination Queue	FORMATNAME:DIRECT=OS:W
Encryption Algorithm	None
Maximum Message Size (in kiloby	4095
Message Priority	Normal
Recoverable	False
Support Segmentation	False
Timeout	4
Timeout Unit	Days
Transactional	True
Use Authentication	False
Use Dead Letter Queue	True
Use Journal Queue	False

Destination Queue
Specify the queue name.
Example: MachineName\[Private$]\QueueName

Figure 7-11. *One-way send port*

Testing the Application

Once application is deployed and configured, you can test the scenario using the following sample message:

```
MSH|^~\&|ADT|MCM||MCM|198808181126|SECURITY|ADT^A01|MSG00001|P|2.3|||AL|NE
EVN|A01|198808181123
PID|||PATID1234^5^M11||J^WILL^A^III||19610611|M||2106-3|1 ELM STREET^^GREEN^NA^11111-
1020|GL|(111)111-1212|(111)111-3434~(111)111-3114||S||PATID12345001^2^M10|123456789|9-87654^NC
NK1|1|J^BA^K|SPO|||||20011105
PV1|1|I|2000^2012^01||||004777^LEBAU^SID^J.|||SUR||-||1|A0
AL1|2||^CAT DANDER
DG1|001|I9|1550|MAL NEO LIVER, PRIMARY|19880501103005|F||
ROL|45^RECORDER^ROLE MASTER LIST|AD|CP|K^SM^E|199505011201
GT1|1122|1519|BILL^GATES^A
IN1|001|A357|1234|BCMD|||||132987
IN2|ID1551001|SSN12345678
ROL|45^RECORDER^ROLE MASTER LIST|AD|CP|K^E|199505011201
```

As you can see, this message is of Version 2.3. When you test this message by sending it via MLLPSend or 7Edit to your receive location, you will receive the accept acknowledgment and the message is moved to your MSMQ queue with changed message version.

■ **Note** In some cases, you need to revert the values back to the original values when sending the data out. For such cases, you can revert them either in another pipeline component on the send side or in a transformation as per requirement. The receive pipeline component can save the changed value to the message context.

Ordered Delivery

In a HL7 messaging solution, ordered delivery of some of the messages is important because the order of those messages relates to real-life events that occur; for example, in patient administration, the patient admit (A01) event occurs before the patient discharge (A03) event. In a system, ADT^A01 should flow before ADT^A03 for a single patient, and this order should be maintained in all systems involved. The key point to note here is that for multiple patients the data can flow through in parallel, but for single patient it should be in order. In order to maintain the order delivery in BizTalk, use the following best practices.

Using Order Delivery on Receive Location

The input message should be received using adapters that support ordered delivery, such as `MSMQ`, `MQSeries`, and `MLLP` in BizTalk. This ensures that messages are published in order to the message box.

Using the Send Port with Ordered Delivery

Set the ordered delivery along with “stop sending subsequent messages on current message failure” of `MLLP` send ports from the transport advanced options. This will ensure that messages are delivered in order from the message box.

Using the Ticket Dispenser and Gate-Keeper Orchestration Pattern

In case of a messaging scenario, the first two options allow you to process the messages in order. However, in cases where you have multiple orchestrations to process the incoming messages before sending the message out, the message order is not guaranteed for messages published from orchestration because each orchestration processing may take a different amount of time to complete. In order to maintain the message order delivery in such scenarios, you need to use a ticket dispenser and re-sequencing pattern in addition to above two steps. The pattern is defined below on a high level.

Creating a Custom Pipeline

Create a custom pipeline component to assign each message a unique sequence number. This sequence is updated to the message context before the message is published to the message box. This sequence number will be used by the gatekeeper orchestration to re-sequence the outgoing messages.

Updating the Sequence Number in Orchestration

In business orchestrations, you can preserve the sequence number in outgoing messages by writing the sequence number in the message context. These orchestrations should also promote a property to make these messages go to gatekeeper orchestration, such as a destination property.

Implementing Gatekeeper Orchestration Using a Sequential Convoy

Implement a new orchestration that serves as gatekeeper for all outgoing messages. It receives all the messages and correlates them based upon a destination property before messages go out. It checks whether the current message is the next message to go out by checking the sequence number of the current message with last sent message sequence number. If the current message is not the one to go out, orchestration keeps it in an in-memory queue. For more details on the pattern, please go to:

`http://msdn.microsoft.com/en-us/library/bb851740(v=bts.10).aspx`

Optimization for Performance

HL7 message delivery usually takes place using the MLLP send adapter, which does not provide as good performance as other adapters such as WCF-SQL, FILE, etc. This can become even slower if the destination system is not accepting the messages at the rate you expect. This can create a bottleneck in BizTalk, and the BizTalk send host may start to throttle, which will further slow down the message delivery.

If the message incoming rate does not slow down, very quickly you will see lots of messages build up in the “Ready to Run” and “Active” states on the send host. Depending on the daily volume of the messages, this can be a huge problem. In order to avoid this problem, you need to follow some best practices, outlined in the following sections.

Using MSMQ for Message Delivery

You should use `MSMQ` or another queuing system like `MQSeries` to store messages before the messages are sent out to a specific destination. Using `MSMQ` is very simple and straightforward in BizTalk and does not require any additional development. `MSMQ` allows you to easily control the message flow to the destination by enabling/disabling the `MSMQ` receive locations whenever the destination system is not available.

Using BizTalk Receive Host Throttling

BizTalk `host throttling` is great way to control the messages coming into BizTalk Message box in case the destination message outgoing rate is slow. This works especially well in messaging-only scenarios. Use the `host throttling performance counters at` `http://msdn.microsoft.com/en-us/library/aa578302.aspx` to determine the right configuration settings for your environment.

Isolating MLLP Hosts

This is a standard BizTalk practice to isolate hosts at receive, send, and orchestration level; however, with HL7-based solutions you may need to further isolate hosts depending on the volume, especially for `MLLP send hosts`. Since it's difficult to scale the `MLLP` send hosts on multiple server, you may need to create multiple `MLLP` send hosts. One criteria to creating these hosts is based on message type, such as one `MLLP` send for `ADT`, one for `ORM` etc.

Using Messaging Scenario

Wherever possible you should use a messaging-based solution only without implementing orchestrations. For HL7 solutions, especially when you need ordered delivery, the complexity is increased with orchestration. Due to the HL7 message being a multipart message, the only way to create and invoke transformation is through orchestration. However, if transformation is simple and only requires few field changes, it is recommended to do it in pipeline component.

Monitoring

Having the ability to monitor your HL7 implementations is extremely important. Without proper monitoring you won't know what is going on with your applications.

There are several monitoring tools on the market, but there is only one that stands out. This product provides you with complete control of the entire HL7 environment. That product is BizTalk360 (`www.biztalk360.com`). BizTalk360 was developed by Saravana Kumar, who is also an Integrations MVP. There are too many features to list here, so it is recommended that you visit the BizTalk360 web site and see them for yourself.

■ **Note** You will be seeing new functionality being added to BizTalk360 to support HL7. One of these features will filter response messages from an HL7 transmission and capture error information. It is recommended that you subscribe to the BizTalk360 mailing list in order to be notified of new functionality.

Summary

In this chapter, you learned various best practices for HL7 solution implementation in BizTalk such as managing schemas, the message exchange pattern, ordered delivery, and more. Implementing these best practices is important, even essential, for any HL7 solution in order for it to perform better and to improve manageability. Lastly, you learned about the need for monitoring.

APPENDIX 1

HL7 Definitions

The following definitions are from the HL7 organization.

Message: A message is the entire unit of data transferred between systems in a single transmission. It is a series of segments in a defined sequence, with a message type and a trigger event.

Segment: A segment is a logical grouping of data fields. Segments within a defined message may be required or optional, may occur only once, or may be allowed to repeat. Each segment is named and is identified by a segment ID, a unique three-character code.

Field: A field is a string of characters. Each field is identified by the segment it is in and its position within the segment; for example, PID-5 is the fifth field of the PID segment. Optional data fields may be omitted. Whether a field is required, optional, or conditional in a segment is specified in the segment attribute tables. The designations are R=Required, O=Optional, C=Conditional on the trigger event or on some other field(s). The field definition should define any conditionality for the field: X=Not used with this trigger event, B=Left in for backward compatibility with previous versions of HL7. A maximum length of the field is stated as normative information. Exceeding the listed length should not be considered an error.

Component: A component is one of a logical grouping of items that comprise the contents of a coded or composite field. Within a field having several components, not all components are required to be valued.

Item number: Each field is assigned a unique item number. Fields that are used in more than one segment will retain their unique item number across segments.

Null and empty fields: The null value is transmitted as two double quote marks (“”). A null-valued field differs from an empty field. An empty field should not overwrite previously entered data in the field, while the null value means that any previous value in this field should be overwritten.

Data type: A data type restricts the contents and format of the data field. Data types are given a two- or three-letter code. Some data types are coded or composite types with several components. The applicable data type is listed and defined in each field definition. Appendix 2 provides a complete listing of data types used in this document and their definitions.

Delimiters: The delimiter values are given in MSH-2 and used throughout the message. Applications must use agreed-upon delimiters to parse the message. The recommended delimiters for immunization messages are <CR> = Segment Terminator; | = Field Separator; ^ = Component Separator; & = Sub-Component Separator; ~ = Repetition Separator; and \ = Escape Character.

Message syntax: Each message is defined in special notation that lists the segment three-letter identifiers in the order they will appear in the message. Braces, {}, indicate that one or more of the enclosed group of segments may repeat, and brackets, [ ], indicate that the enclosed group of segments is optional.

Z segments: All message types, trigger event codes, and segment ID codes beginning with Z are reserved for locally defined messages. No such codes will be defined within the HL7 standard. The users of this guide have agreed to eliminate Z segments from their implementations in order to produce a standard method that will be used nationally to transmit immunization data.

APPENDIX 2

HL7 Basic Message Construction Rules

The following information is provided directly from "Processing HL7 Messages" on Microsoft's MSDN site. You can access this and more information at http://msdn.microsoft.com/en-us/library/ee404922.aspx.

Encoding Rules for Sending Messages

- Encode each segment in the order specified in the abstract message format.
- Place the Segment ID first in the segment.
- Precede each data field with the field separator.
- Encode the data fields in the order and data type specified in the segment definition table.
- End each segment with the segment terminator.
- Components, subcomponents, or repetitions that are not valued at the end of a field need not be represented by component separators. The following example shows that the data fields are equivalent:

```
^XXX&YYY&&^ is equal to ^XXX&YYY^
|ABC^DEF^^| is equal to |ABC^DEF|
```

Encoding Rules for Receiving

- If a data segment that is expected is not included, treat it as if all data fields within were not present.
- If a data segment is included that is not expected, ignore it; this is not an error.
- If data fields are found at the end of a data segment that are not expected, ignore them; this is not an error.

APPENDIX 3

HL7 Version 2.x Data Types

The following are the data types supported by the HL7 Accelerator in BizTalk 2013. Microsoft added support for HL7 `Version 2.6` in BizTalk 2013 R2. You can view the changes for the Version 2.6 data types in Appendix 4.

Note *All the information contained within these sections is from the HL7 Version 2.5.1 Standard Implementation Guide. You can view the brief for this guide at* `www.hl7.org/implement/standards/product_brief.cfm?product_id=144`.

Data Structure—Definition

Table A3-1 lists the complex data structures used in Version 2.5.1.

Table A3-1. *Complex Data Structure Definitions*

Data Structure	Description
AD	Address
AUI	Authorization Information
CCD	Charge Code and Date
CCP	Channel Calibration Parameters
CD	Channel Definition
CE	Coded Element
CF	Coded Element with Formatted Values
CNE	Coded with No Exceptions
CNN	Composite ID Number and Name Simplified
CP	Composite Price
CQ	Composite Quantity with Units
CSU	Channel Sensitivity
CWE	Coded with Exceptions

(*continued*)

Table A3-1. (*continued*)

Data Structure	Description
CX	Extended Composite ID with Check Digit
DDI	Daily Deductible Information
DIN	Date and Institution Name
DLD	Discharge Location and Date
DLN	Driver's License Number
DLT	Delta
DR	Date/Time Range
DT	Date
DTM	Date/Time
DTN	Day Type and Number
ED	Encapsulated Data
EI	Entity Identifier
EIP	Entity Identifier Pair
ELD	Error Location and Description
ERL	Error Location
FC	Financial Class
FN	Family Name
FT	Formatted Text Data
GTS	General Timing Specification
HD	Hierarchic Designator
ICD	Insurance Certification Definition
ID	String Data
IS	String Data
JCC	Job Code/Class
LA1	Location with Address Variation 1
LA2	Location with Address Variation 2
MA	Multiplexed Array
MO	Money
MOC	Money and Code
MOP	Money or Percentage
MSG	Message Type
NA	Numeric Array
NDL	Name with Date and Location

(*continued*)

Table A3-1. (*continued*)

Data Structure	Description
NM	Numeric
NR	Numeric Range
NUL	
OCD	Occurrence Code and Date
OSD	Order Sequence Definition
OSP	Occurrence Span Code and Date
PIP	Practitioner Institutional Privileges
PL	Person Location
PLN	Practitioner License or Other ID Number
PPN	Performing Person Time Stamp
PRL	Parent Result Link
PT	Processing Type
PTA	Policy Type and Amount
QIP	Query Input Parameter List
QSC	Query Selection Criteria
RCD	Row Column Definition
RFR	Reference Range
RI	Repeat Interval
RMC	Room Coverage
RP	Reference Pointer
RPT	Repeat Pattern
SAD	Street Address
SCV	Scheduling Class Value Pair
SI	Sequence ID
SN	Structured Numeric
SPD	Specialty Description
SPS	Specimen Source
SRT	Sort Order
ST	String Data
TM	Time
TQ	Timing Quantity
TS	Time Stamp
TX	Text Data

(*continued*)

Table A3-1. (*continued*)

Data Structure	Description
UVC	UB Value Code and Amount
varies	Varies
VH	Visiting Hours
VID	Version Identifier
VR	Value Range
WVI	Channel Identifier
WVS	Waveform Source
XAD	Extended Address
XCN	Extended Composite ID Number and Name for Persons
XON	Extended Composite Name and Identification Number for Organizations
XPN	Extended Person Name
XTN	Extended Telecommunication Number

Each of the above complex data structures contains multiple components. Let's look at one of these.

Data Structure AD

This data structure defines what components are in the address data structure. Table A3-2 shows these.

Table A3-2. *Address Components*

Seq.	Index	Sec.	Table	Opt.	Length	Data Type
1	Street Address			O	120	ST
2	Other Designation			O	120	ST
3	City			O	50	ST
4	State or Province			O	50	ST
5	ZIP or Postal Code			O	12	ST
6	Country		0399	O	3	ID
7	Address Type		0190	O	3	ID
8	Other Geographic Designation			O	50	ST

The last column in Table A3-2 lists the base data type for a component. An example of this is the street address. The base data type for this is ***ST***. This refers to string data.

Table A3-3 shows all the HL7 data types for the components used in the data structures.

Table A3-3. *HL7 Data Types*

Data Type Category	Data Type	Data Type Name
Alphanumeric		
	ST	String
	TX	Text data
	FT	Formatted text
Numerical		
	NM	Numeric
	SI	Sequence ID
	SN	Structured numeric
Identifier		
	ID	Coded values for HL7 tables
	EI	Entity identifier
	PT	Processing type
Date/Time		
	DT	Date
	TS	Time stamp
Code Values		
	CE	Coded element
	CF	Coded element with formatted values
	CK	Composite ID with check digit
	CN	Composite ID number and name
	CX	Extended composite ID with check digit
	XCN	Extended composite ID number and name
Demographics		
	AD	Address
	PN	Person name
	TN	Telephone number
	XON	Extended composite name and ID number for organizations
	XTN	Extended telecommunications number

Rules Applied to Components

HL7 Version 2.5 and higher have restrictions on the maximum lengths of data types. The following are the rules that apply.

1. The upper bound maximum length of a data type is the sum of the following:
    a. The combined length of all components (including those not required) that are not mutually exclusive.
    b. The combined length of the largest set of mutually exclusive components.
    c. The combined length of message delimiters that are required to construct the field.
2. The lower bound maximum length of a data type is the sum of the following:
    a. The combined length of all required components that are not mutually exclusive.
    b. The combined length of the largest set of mutually exclusive required components.
    c. The combined length of message delimiters that are required to construct the field.

■ **Tip** The information contained in this appendix is just a sampling of the information about the HL7 Version 2.5 and above data types. You can find more information in *The HL7 Implementation Guide* for the HL7 version 2.x you are using, by viewing the HL7 Product Brief for the Version 2 Product suite at `www.hl7.org/implement/standards/product_brief.cfm?product_id=185`.

APPENDIX 4

HL7 Version 2.6

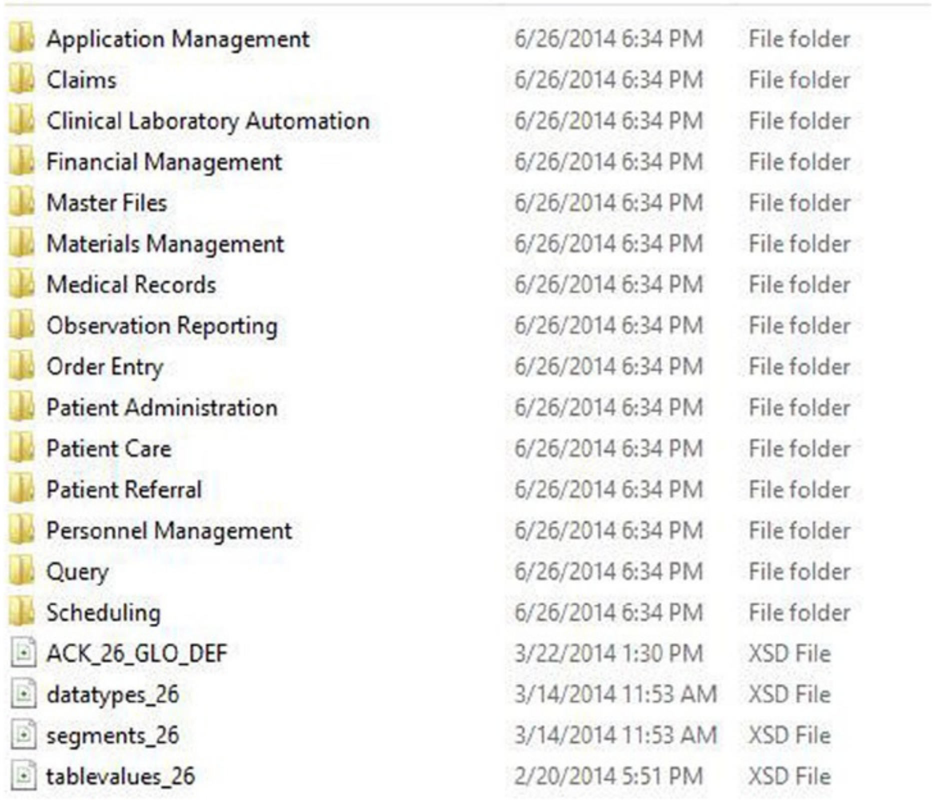
The release of Microsoft BizTalk 2013 R2 includes the HL7 Version 2.6 standard.

HL7 Version 2.6

If you compare Version 2.6 to Version 2.5.1, you will find that quite a few new events, segments, and messages have been added. In addition to these, two new chapters have been added to the HL7 Version 2.6 standard. These are Chapter 16 - Claims and Reimbursements and Chapter 17 - Materials Management.

Note Much of the information contained within this appendix is directly from *The HL7 Version 2.6 Standard Guide*.

If you open the folder `C:\Program Files (x86)\Microsoft BizTalk 2013 R2 Accelerator for HL7\Templates\Schemas\V2.X\2.6`, you will see what is included in Version 2.6 (also shown in Figure A4-1).



Application Management	6/26/2014 6:34 PM	File folder
Claims	6/26/2014 6:34 PM	File folder
Clinical Laboratory Automation	6/26/2014 6:34 PM	File folder
Financial Management	6/26/2014 6:34 PM	File folder
Master Files	6/26/2014 6:34 PM	File folder
Materials Management	6/26/2014 6:34 PM	File folder
Medical Records	6/26/2014 6:34 PM	File folder
Observation Reporting	6/26/2014 6:34 PM	File folder
Order Entry	6/26/2014 6:34 PM	File folder
Patient Administration	6/26/2014 6:34 PM	File folder
Patient Care	6/26/2014 6:34 PM	File folder
Patient Referral	6/26/2014 6:34 PM	File folder
Personnel Management	6/26/2014 6:34 PM	File folder
Query	6/26/2014 6:34 PM	File folder
Scheduling	6/26/2014 6:34 PM	File folder
ACK_26_GLO_DEF	3/22/2014 1:30 PM	XSD File
datatypes_26	3/14/2014 11:53 AM	XSD File
segments_26	3/14/2014 11:53 AM	XSD File
tablevalues_26	2/20/2014 5:51 PM	XSD File

Figure A4-1. *Folder heirarchy for Version 2.6*

Differences Between Version 2.6 and 2.5.1

Let's take a brief look at some of the differences between Version 2.6 and Version 2.5.1.

Data Type Changes

Three of the Version 2.5.1 data types are affected.

- The `CE` data type has been removed. It has been replaced with the `CNE - Coded with no exceptions` and `CWE - Coded with Exceptions` data types. Figures A4-2 and A4-3 show the components for these data types.

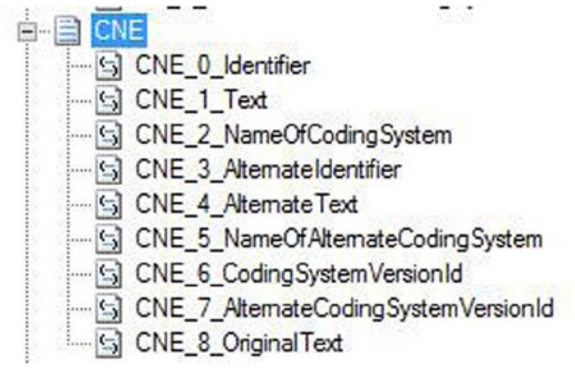

CNE
CNE_0_Identifier
CNE_1_Text
CNE_2_NameOfCodingSystem
CNE_3_AlternateIdentifier
CNE_4_AlternateText
CNE_5_NameOfAlternateCodingSystem
CNE_6_CodingSystemVersionId
CNE_7_AlternateCodingSystemVersionId
CNE_8_OriginalText

Figure A4-2. *CNE components*

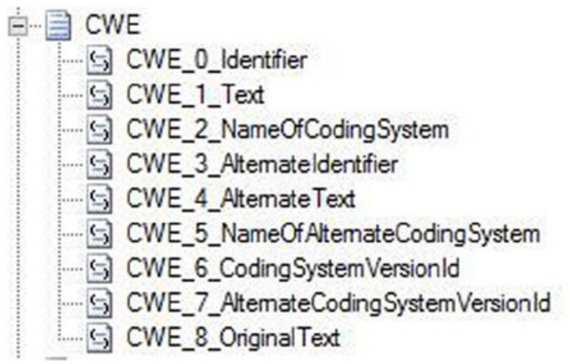

CWE
CWE_0_Identifier
CWE_1_Text
CWE_2_NameOfCodingSystem
CWE_3_AlternateIdentifier
CWE_4_AlternateText
CWE_5_NameOfAlternateCodingSystem
CWE_6_CodingSystemVersionId
CWE_7_AlternateCodingSystemVersionId
CWE_8_OriginalText

Figure A4-3. *CWE Components*

- The TS - Time Stamp data type is no longer used. It was replaced with the DTM data type, which is equal to the TS data type, but without the second component.
- The ED - Encapsulated Data data type uses a new table (table 0834). All the other tables referenced by this data type have been refined. The values contained within this table are a subset of *W3C MIME* types; audio, image, model, multipart, text, and video.

Segments

Two new segments have been added: UAC - User Authentication and ARV - Access Restrictions.

UAC - User Authentication

The UAC is an optional segment available in all the Version 2.6 messages. This segment provides user authentication credentials for the receiving system. Your choices for credentials are the following:

- Kerberos Service Ticket (http://technet.microsoft.com/en-us/library/bb742516.aspx)
- SAML Assertion (http://en.wikipedia.org/wiki/SAML)

The UAC segment includes two new fields:

- UAC-1 - Credential Type Code, which is a CWE data type
- UAC-2 - User Authentication Credential, which is an ED data type

ARV - Access Restrictions

According to Chapter 3 of *The HL7 Version 2.6 Standard Guide*, the ARV, which is a new segment, has been added to all the ADT message types. Its main use is to specify access restrictions. Let's say a person or patient objects to having specific data exposed to family members or friends.

The ARV Segment has six fields. The key fields are ARV-2, which contains the Action Code (Add, Delete, and Update), and the ARV-3 which contains the Access Restriction Value. The ARV-3 is used to identify what information is restricted.

Mood Code

The Mood Code field has been added. The Mood Code is used to *specify how the data contained within a particular segment be processed* by the receiving system. You will find that the Mood Code field has been added to the following segments; OBX, RXO, PRB, GOL, PTH and PRD.

■ **Tip** You can consider the Mood Code as the processing instruction for the segment. It is only allowed in new messages.

A good example of its use would be if there is an OBX segment in the outbound message. This field can specify if the OBX contains a result, or that the sender expects a result back.

In this sample, the Mood Code field is a CNE data type. These values are contained within the HL7 Version 2.6 - 0725 table. The components of the Mood Code are shown in Listing A4-1.

Listing A4-1. Mood Code Components

```
<Identifier (ST)> ^ <Text (ST)> ^ <Name of Coding System (ID)> ^ <Alternate Identifier (ST)> ^
<Alternate Text (ST)> ^ <Name of Alternate Coding System (ID)> ^ <Coding System Version ID (ST)> ^
<Alternate Coding System Version ID (ST)> ^ <Original Text (ST)>
```

Table A4-1 shows some common Mood Codes.

Table A4-1. *Mood Codes (Source: HL7 Version 2.6 Standard Guide)*

Value	Description
`APT`	Appointment
`ARQ`	Appointment Request
`EVN`	Event
`EVN,CRT`	Event Criterion
`EXP`	Expectation
`INT`	Intent
`PRMS`	Promise
`PRP`	Proposal
`RQO`	Request-Order

Besides the values listed in Table A4-1, the 0725 table contains two additional values: "Criterion applying to Eve" and "Eg Use in Care Plans."

New Message Types

If you are working with veterinary medicine, then the addition of two new message types will be of interest to you. These new message types have been added to the laboratory domain.

- `037: OPL` - Population/Location-Based Laboratory Order Message
- `038: OPR` - Population/Location-Based Laboratory Order Acknowledgment Message

You can read more about these message types in Chapter 4 of the HL7 Version 2.6 Standard Guide.

Two New Chapters

As previously mentioned, two new chapters have been added. Each chapter provides a new transaction set.

■ **Note** There is quite a bit of information contained within these new chapters. We won't go into too much detail here. If you read the chapters in the HL7 Version 2.6 guide, you will find all you need to know about these new transaction sets.

Claims and Reimbursements Transaction Set

This transaction set is in Chapter 16 of the HL7 Version 2.6 Guide. It supports the communication of claims information from a provider to payer or reimbursement authority.

■ **Note** The Claims and Reimbursements Transaction Set is not used in the United States; HIPAA is used instead.

Materials Management Transaction Set

This new transaction set supports supply chain management within a healthcare facility. Two new and distinct topics are covered within this transaction set.

Inventory Item Master File Updates

This topic covers the exchange of common materials-related reference files. This files are also known in HL7 as master files. Eight new segments are related to this topic. Table A4-2 describes these segments.

Table A4-2. *Master File Segments*

Identifier	Description
IIM	INVENTORY ITEM MASTER SEGMENT
ITM	MATERIAL ITEM SEGMENT
STZ	STERILIZATION PARAMETER SEGMENT
VND	PURCHASING VENDOR SEGMENT
PKG	PACKAGING SEGMENT
PCE	PATIENT CHARGE COST CENTER EXCEPTION
IVT	MATERIAL LOCATION SEGMENT
ILT	MATERIAL LOT SEGMENT

In addition to the segments, two new types have been added. These are `Placer Request` and `Filler Request`. You can read about these types in Chapter 17 of *The HL7 Version 2.7 Standard Guide*.

Sterilization and Decontamination

Sterilization and decontamination cover order-related information for an instrument tracking system and a sterilizer. Four new segments have been added.

- `SCP` - STERILIZER CONFIGURATION SEGMENT
- `SLT` - STERILIZATION LOT SEGMENT
- `SDD` - STERILIZATION DEVICE DATA SEGMENT
- `SCD` - ANTI-MICROBIAL CYCLE DATA SEGMENT

Queries

There are a few more modifications. In Chapter 5 of the HL7 Version 2.6 standard, the query profile has replaced query conformance statements. The reason for this is that the term “query conformance statements” was very misleading.

Query Profile

The guide describes a query profile as

> *A declaration which sets forth the name of the query supported by the Server, the logical structure of the information queried, and the logical structure of what can be returned.*
>
> Chapter 5, page 7 of the of the HL7 Version 2.6 Standard

Removed Message Types

Several query message types that exist in Version 2.5.1 have been removed. These are `EQQ` - embedded query language query, `RQQ` - event replay query, `SPQ` - stored procedure request, and `VQQ` - virtual table query.

■ **Tip** One thing you need to be aware of is that Version 2.6 has increased in size over the previous versions. Many new artifacts have been added. There are well over 300 trigger events and more than 2,000 fields.

Summary

As you have seen, HL7 Version 2.6 is very different then Version 2.5.1. If you are planning on using it, then you should review the complete standard. You can download it from `www.hl7.org/implement/standards/product_brief.cfm?product_id=145`.

Index

HL7 for BizTalk

Howard Edidin
Vikas Bhardwaj

Apress®

HL7 for BizTalk

ISBN-13 (pbk): 978-1-4302-6764-5

ISBN-13 (electronic): 978-1-4302-6763-8

Publisher: Heinz Weinheimer
Lead Editor: Ewan Buckingham
Developmental Editor: Dominic Shakeshaft
Technical Reviewer: Mike Diiori
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Louise Corrigan, Jim DeWolf, Jonathan Gennick, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Steve Weiss
Coordinating Editor: Jill Balzano
Copy Editor: Mary Behr
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

This book is dedicated to my two sons, Scott and Mark

—Howard S. Edidin

Contents

Foreword

In its 14 short years, BizTalk Server has grown to serve as the premier integration solution for more than 12,000 companies around the world including 81 percent of the Fortune Global 100 companies in various industries including many in healthcare.

Whether they are providers, payers, in clinical services or life sciences, organizations in the healthcare industry face challenges caused by decades of niche systems, changing regulations, shrinking budgets, acquisitions as well as disaggregation, and demanding consumers. Core to enabling this integration is the BizTalk Accelerator for HL7. It is a highly versatile and performant complete health care solution for the exchange of data between health care computer applications based on the Health Level Seven (HL7) standard. We think integrated hospitals ensure better and faster care. BizTalk enables many scenarios in health care like integrating Electronic Health Records (EHR), digitizing clinical dashboards, optimizing clinical supplies and automating patient scheduling.

In this recent release of BizTalk Server, Microsoft has made deep investments in performance improvements, cloud and hybrid cloud scenarios.

An example to illustrate the scale we see specifically in the healthcare industry is with HCA, the leading provider of healthcare services in the United States. HCA is a company composed of locally managed facilities that includes about 165 hospitals and 115 freestanding surgery centers in 20 states and England. Today, the system handles an average of 23,000,000 messages per day, average, with a high of 49.7 million message (at a rate of 266 messages per second).

We see continued investment with BizTalk in enabling complex on premise integration scenarios at scale and enabling the hybrid cloud solutions in healthcare.

I am certainly delighted to be asked to write the forward for this book. Howard and I had the great opportunity to collaborate at the quarterly HL7 Working Group meetings. Connectathons for the emerging standard FHIR are held the weekends leading up to the working group meetings. Howard has been an early adopter in proving integration scenarios using BizTalk, and Microsoft Online Services like O365 and Microsoft Azure and FHIR. You will be able to gain practical insights from this book.

—Leslie Sistla
Director, Technology Strategy | Worldwide Health Industry

Leslie Sistla is a Director of Technology Strategist on the Worldwide Health Industry Team. Leslie joined Microsoft Corp in 2000 bringing with her experiences working for Oracle Corp and McKesson (HBOC).

At Microsoft, Leslie helps customers and partners around the world understand how they can meet their privacy, security and regulatory compliance obligations with solutions built on the Microsoft platform, both mobile and cloud. She represents Microsoft at healthcare key Standards Development Organizations working group meetings, like HL7, and IHE. Finally, she works with customers and partners, helping design solutions framed by using a healthcare reference architecture, the Connected Health Framework (CHF).

About the Authors

Howard S. Edidin is an Integrations Architect specializing in enterprise application integration. Howard runs his own consulting firm, Edidin Group, Inc, which is a Gold Member of the HL7 International Organization. Howard's firm specializes in delivering HL7 and HIPAA healthcare solutions and providing guidance in the use of HL7 with BizTalk. Howard is active in several HL7 Working Groups and is involved with the development of a new HL7 standard. In addition to BizTalk, Howard works with Azure, SQL Server, and SharePoint. Howard and his wife, Sharon, live in a northern suburb of Chicago. Howard maintains the `biztalkin-howard.blogspot.com` and `fhir-biztalk.com` blogs and can be contacted directly at `hedidin@edidingroup.net`.

Vikas Bhardwaj is a Technical Architect at Syntel Inc. Vikas has 14 years of IT experience with Microsoft technologies like BizTalk Server, .NET, C#, and SQL Server. Vikas has implemented various integration solutions using BizTalk Server, including one of the largest implementations of BizTalk and HL7. Vikas presently lives in Nashville, Tennessee with his wife, Poonam, and two kids, Shivam and Ayaan. You can check out Vikas' blog at `http://vikas15bhardwaj.wordpress.com` and Vikas can be contacted directly at `vikas15.bhardwaj@gmail.com`.

About the Technical Reviewer

Mike Diiorio is an architect and developer with MedVirginia. He has 14 years of experience as a consultant specializing in web technologies, system integration, and enterprise application development. Mike is a Microsoft Certified Application Developer and holds Microsoft Certified Technical Specialist credentials in .NET Web Development and BizTalk Server. His primary focus is in the area of application integration based on the Microsoft .NET Framework including BizTalk Server, Microsoft Azure, Windows Communication Foundation, and Windows Workflow. He has also served as an Application Platform/BizTalk Specialist in the Microsoft Virtual Technology Solution Program (V-TSP). Mike holds bachelor degrees in Music Industry and Computer Information Systems from James Madison University in Virginia. Mike maintains his blog at `http://mikediiorio.net`.

Acknowledgments

I am fortunate to have a wife who gave me her support and put up with me over the many months I spent writing this book. My son Mark, who is following in my footsteps, provided quality feedback, for which I am grateful.

I especially would like to thank my co-author, Vikas Bhardwaj, for his support and dedication, and my good friend and colleague Tom Banaski for his help in providing us excellent real-world content. My editor, Dominick Shakeshaft of Apress, provided a number of suggestions that greatly improved the quality of the text, and Jill Balzano of Apress kept us on track.

And finally, my thanks to Mike Diiorio, our technical reviewer, for his expertise and guidance. Without Mike, this book would not be possible.

—Howard S. Edidin

First and foremost, I would like to thank my co-author Howard S. Edidin who continuously inspired me and provided me quality feedback.

I would also like to thank Apress team especially the Editor Dominick Shakeshaft who provided feedback on book readability and Jill Balzano who kept us on track and made sure book is delivered on time.

Last but not least I would like to thank my wife Poonam, who gave me all the support during her pregnancy months while writing this book. Without her support this book would not have been possible. I dedicate this book to my wife Poonam and my two son Shivam & Ayaan.

—Vikas Bhardwaj