Ensemble Best Practices

Version 2018.1
2020-11-13

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Ensemble Best Practices
Ensemble Version 2018.1 2020-11-13

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

About This Book

This book describes best practices for organizing and developing Ensemble productions. The Ensemble based software that is installed and running live today is evenly divided between two general types: Routing productions, and productions that provide application connectivity. This book describes best practices for both types of production.

This book contains the following chapters:

- Best Practices for Ensemble Development
- Design Model for a Routing Production
- Converting Interfaces to Ensemble

For a detailed outline, see the table of contents.

For information on debugging Ensemble, see the following chapters elsewhere in the Ensemble books:

- "Debugging Routing Rules" in *Developing Business Rules*
- "Testing and Debugging" in *Developing Ensemble Productions*

For more general information, see *Using InterSystems Documentation*.

1

Best Practices for Ensemble Development

This chapter is a general overview that prepares team members to work on Ensemble projects. It outlines development tasks and identifies sources of information about Ensemble and about enterprise integration. The information in this chapter pertains to all types of Ensemble project.

1.1 Project Goals

The goal of any Ensemble development project is to deliver an Ensemble production. A *production* is a specialized package of software and documentation that solves a specific integration problem for an enterprise customer. For an overview, see *Introducing Ensemble*.

This section describes Ensemble in terms of the software elements that application developers must create and configure in order to deliver a solution. The next section, "Project Delivery", summarizes the sequence and outcome of an Ensemble development project.

1.2 Project Delivery

The Ensemble product architecture supports various styles of delivery to the enterprise:

- An Ensemble production might comprise the entire integration solution for an enterprise, or the production can fit into existing solutions — or partial solutions — that are already in place at the enterprise.
- An Ensemble production can replace, upgrade, or add new features to a legacy system as needed, without requiring any part of the legacy system to be removed or changed.
- Ensemble supports incremental development projects, so a development team can choose to advance the boundary between old and new systems as rapidly or as slowly as the enterprise requires.

Many Ensemble development projects follow a phase sequence similar to this one:

Order	Phase	Goal	Focus
1	Specification	Specify the requirements for the production.	What must the production be able to do?
2	Design	Design the production software.	How must the elements interact?
3	Coding	Build the production software.	Are additional elements needed?
4	Test	Test the production software.	Does the production satisfy the requirements that you specified?
5	Deployment	Install the software in its target location.	Are you prepared to test, design, create, and rework as needed?
6	Release	Deliver software and project artifacts.	What will be useful to the system administration team?

1.3 Documentation

It is a fundamental best practice to read the Ensemble documentation. Some chapters you should read from start to finish. Others you can reference using the Search and Index features of the InterSystems online documentation system. For helpful tips, see *Using InterSystems Documentation*.

Your best starting points are the following books and chapters. Each provides further cross-references:

- *Introducing Ensemble* introduces the terminology and basic concepts.
- *Developing Ensemble Productions* describes the development tasks.
- *Configuring Ensemble Productions* describes how to perform the configuration tasks related to creating an Ensemble production.
- Later chapters in this book apply to deploying interface routing solutions. See "Design Model for a Routing Production" and "Converting Interfaces to Ensemble."

Caché and Ensemble share the same underlying technologies. For this reason, Caché documentation is an important resource for Ensemble developers. A useful starting point is the *Caché Programming Orientation Guide*.

Helpful background materials include the following:

- *Using Caché Basic*
- *Using Caché Objects*
- *Using Caché ObjectScript*
- *Using Studio*
- *Using the Caché SQL Gateway*
- *Creating Web Services and Web Clients in Caché*
- *Projecting Objects to XML*
- *Using Caché XML Tools*

Other useful development guides include:

- *Using Caché SQL*

- *Using Caché Multidimensional Storage (Globals)*
- *Caché High Availability Guide*
- *Using Java with Caché*
- *Using ActiveX with Caché*
- *Using the Caché ActiveX Gateway*
- *Using Zen*

Language reference materials include:

- *Caché Basic Reference*
- *Caché MultiValue Basic Reference*
- *Caché ObjectScript Reference*

For information about system utilities and security, consult the following guides:

- *Caché System Administration Guide*
- *Caché Security Administration Guide*

2

Design Model for a Routing Production

This chapter describes a design model that Ensemble customers have used successfully to build interface routing solutions. As such, it can be thought of as a collection of best practices for developing an Ensemble routing production.

This chapter describes only one approach. Other implementation strategies can be successful as well. Each organization has its own standards for writing code and conducting programming projects. This chapter simply describes those aspects of a routing production that are unique to Ensemble and that benefit from a carefully structured approach.

In general, the key to a good design is clarity and simplicity. Simplicity does not mean a small number of parts. Simplicity means that each part within the model has a clear function and is intuitively easy to find.

2.1 Configuration Items

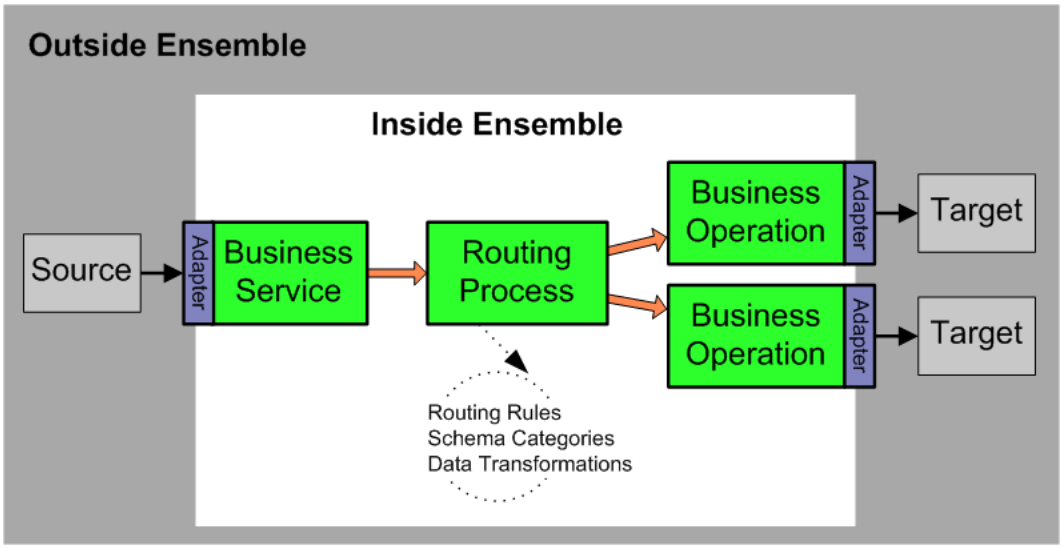
An Ensemble production consists of *configuration items*. There are three types of configuration item:

- *Business services* accept incoming messages.
- *Business operations* send outgoing messages.
- *Business processes* direct all the work in between. There are some specialized types of business process:

  A *routing process* routes and transforms messages by invoking these key components of a routing production:

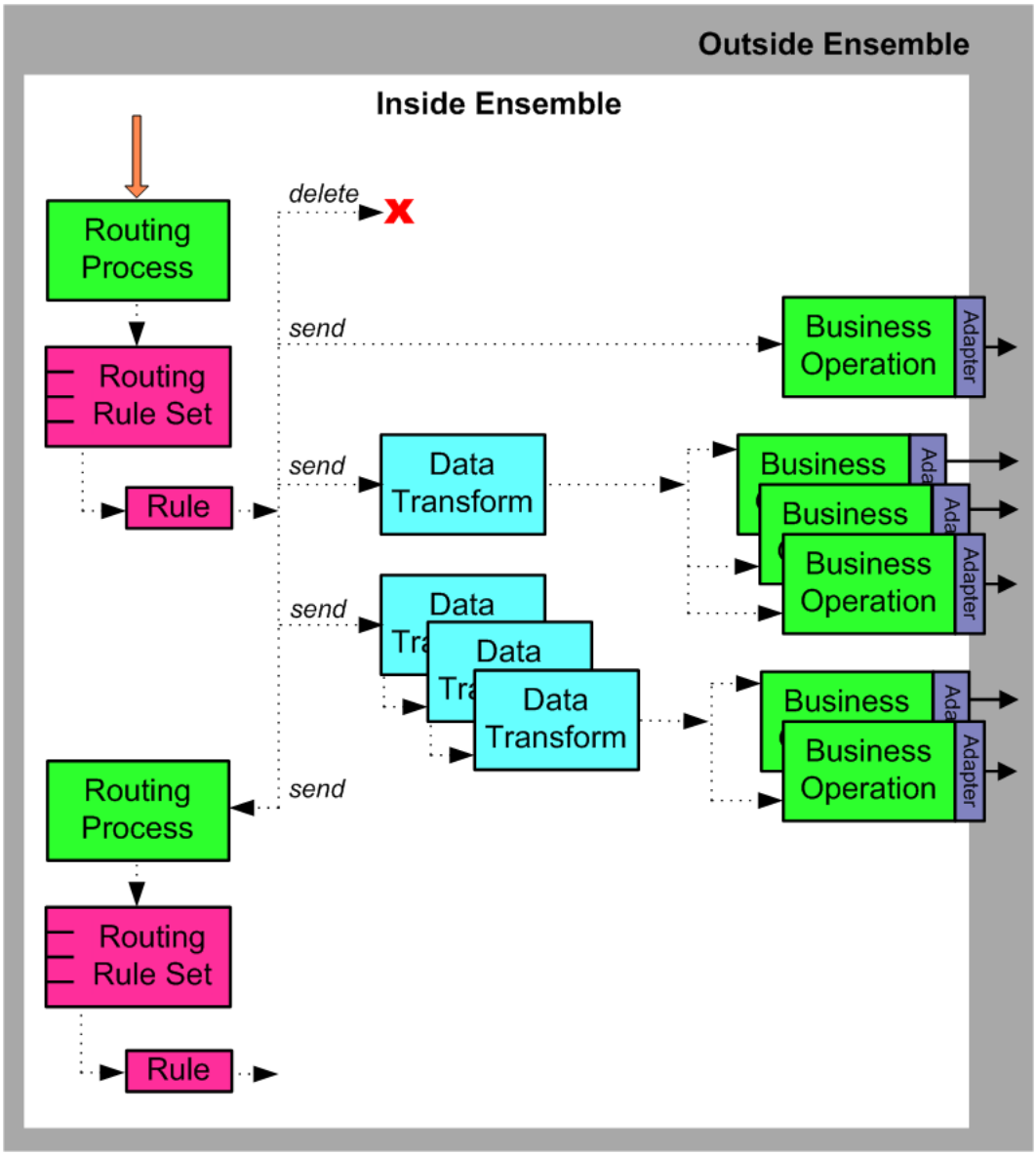
  - *Routing rules* direct messages to their destinations based on message contents.
  - *Schema categories* provide a means to validate and access message contents.
  - *Data transformations* apply changes to prepare messages for their destinations.

  A *sequence manager* ensures that related messages arrive at their targets with the proper sequence and timing



The following figure expands the diagram of a routing process to show how it uses routing rules to direct the flow of information through the interface. The following figure includes details about the routing rules and data transformations that are only implied by the dotted circle in the diagram above.

A routing process has a routing rule set associated with it. Depending on how the rule set is defined, and depending on the type of message that arrives from the source application, the rule set identifies which of its rules should execute. More than one rule can execute, in sequence, but for simplicity the following figure shows the case where the rule set chooses only one rule. From this point, as shown, a rule can either delete the message or it can send the message to a destination within Ensemble. If this destination is a business operation, the message then leaves Ensemble on its way to the target application.



2.2 Application Specification

Ideally, a clinical application provides an *HL7 application specification document*, or *implementation guide*, that explains which types of HL7 event the application can send (or receive), and which message segments and pieces of each segment the application sends (or expects) for these events.

A formal document of this kind is extremely detailed, and extremely useful. If a specification document is available, the administrator for the clinical application can show it to you, and can explain to you which of the many possible events are actually in use in the enterprise. This is usually a small subset of the full list of events.

Even if there is no application specification document, there is usually some informal documentation. Ask the application administrator to see any notes that are available.

As an alternative to documentation, or to validate that the existing documentation is correct, you can examine the messages themselves to determine which types of HL7 event the application sends or expects to receive. Ask the application administrator to provide you with a sampling of the message data, saved in files.

Once you have the HL7 message files, you can use the Document Viewer to view and parse them using various schema definitions. In this way you can quickly determine whether or not the application is using a standard HL7 schema. If not, you can use the Message Viewer to refine and test a custom HL7 schema for the application.

Background information that supports these tasks can be found in subsequent chapters of this book, and in other books:

- "Creating Custom Schema Categories" in *Ensemble Virtual Documents*
- "Controlling Message Validation" in *Ensemble Virtual Documents*
- "Available Tools" in the *Ensemble HL7 Version 2 Development Guide*
- "Choosing Schema Categories" in the chapter "Converting Interfaces to Ensemble"

2.3 Production Spreadsheet

It is helpful to maintain a spreadsheet that organizes your information system, application by application. You can create the spreadsheet on paper, or using any software tool that you prefer.

Feed	Application	Name	Type	Connection	Sends	Receives	ACK
Chicago Street Interface Engine	St. Vincent's HIS Contact: JW Smith Ph: 203-555-1234	StVint	TCP	IP: 10.10.10.128 Port: 4501	ADT_A01 ADT_A03 ADT_A08 ORM_O01		Yes
				IP: 10.10.10.128 Port: 4601		ORU_R01 ORM_O01	Yes
Chicago Street Interface Engine	Mercy HIS Contact: Mary Jones Ph: 314-555-0987	Mercy	TCP	IP: Port:	ADT_A01 ADT_A03 ADT_A08 ORM_O01		Yes
				IP: Port:		ORU_R01 ORM_O01	Yes
Memorial HIS Server	Memorial Hospital HIS Contact: Bill Sharp Ph: 609-567-8765	Memo	FTP	IP: 172.12.189.32 User: ftpclient PW: **********	ADT_A01 ADT_A03 ADT_A08		No
						ORU_R01 ORM_O01	No

As a general guideline, you should provide a row for each application that provides an incoming or outgoing data feed. In each row, the following columns are useful:

- **Feed** — An interface engine or server often feeds messages from several applications into the routing production. When this is the case, note the engine or server name here.
- **Application** — Briefly identify a specific application, its role in your system, and the person to contact about any issues or problems with this application. Ideally, this description makes sense to members of your organization who do not use Ensemble, but who are generally familiar with how the system works.
- **Name** — A unique 3– to 6–character name for this application.
- **Type** — The protocol that the application uses for external communications.
- **Connection** — Connection details, such as an IP address and port number.

- **Sends** — The HL7 Version 2 message structures that this application contributes to the information system.

  Consider the incoming message structure after it enters the information system. Does it need to be routed or transformed differently depending on the destination system or other factors? If so, list it multiple times, with a note regarding the differences. This way, your spreadsheet can serve as a checklist while you create the necessary routing rules, data transformations, and custom schemas for each case.

- **Receives** — The HL7 Version 2 message structures that this application consumes from the information system.

- **ACKs** — Acknowledgment details. Are ACK and NACK messages expected? Is there a separate interface for sending and receiving them? Should Ensemble generate the ACKs and NACKs, or will the receiving application do so?

When you begin the project, your initial spreadsheet does not need to describe every application in the information system. You can add to the spreadsheet as you deploy each new interface.

2.4 Interface Design

This topic outlines an effective way to keep interface designs modular so that your Ensemble production remains easy to understand, scale, and maintain at each stage of its development:

- Provide one business service for each application that sends HL7 messages into Ensemble. This is an application that has at least one entry in the **Sends** column of the production spreadsheet.

  One business service can receive all of the message structures for the same application. This is usually the appropriate design. When you set up a business service for this purpose you generally intend for it to stay connected to its application system all the time.

  There might be aspects of the configuration that require you to provide additional business services connected to the same application. For example, if the source application is already configured to send messages to two different TCP/IP ports, you could set up one business service to receive all the messages from one port, and another business service for the other port. This is generally consistent with the model of one business service per application, since there is one business service per communication source.

  Alternatively, when hundreds of users of the same clinical application are sending data into the enterprise, it can be useful to route all of these messages into Ensemble via one business service that may or may not stay permanently connected to any single instance of the application.

- Provide one routing process for each business service. The routing process can route messages based on the contents of the message itself, or information stored in Ensemble. If the routing depends on the contents of other messages, or requires invocation of external applications to determine the routing, the routing process should be a BPL business process that calls out to other classes. However, in most cases a routing process based on the built-in routing rule engine is sufficient.

  Consider using a sequence manager when it is vital that related messages arrive at their targets in the proper sequence. For details, see "Configuring the Production" in the *Ensemble HL7 Version 2 Development Guide*.

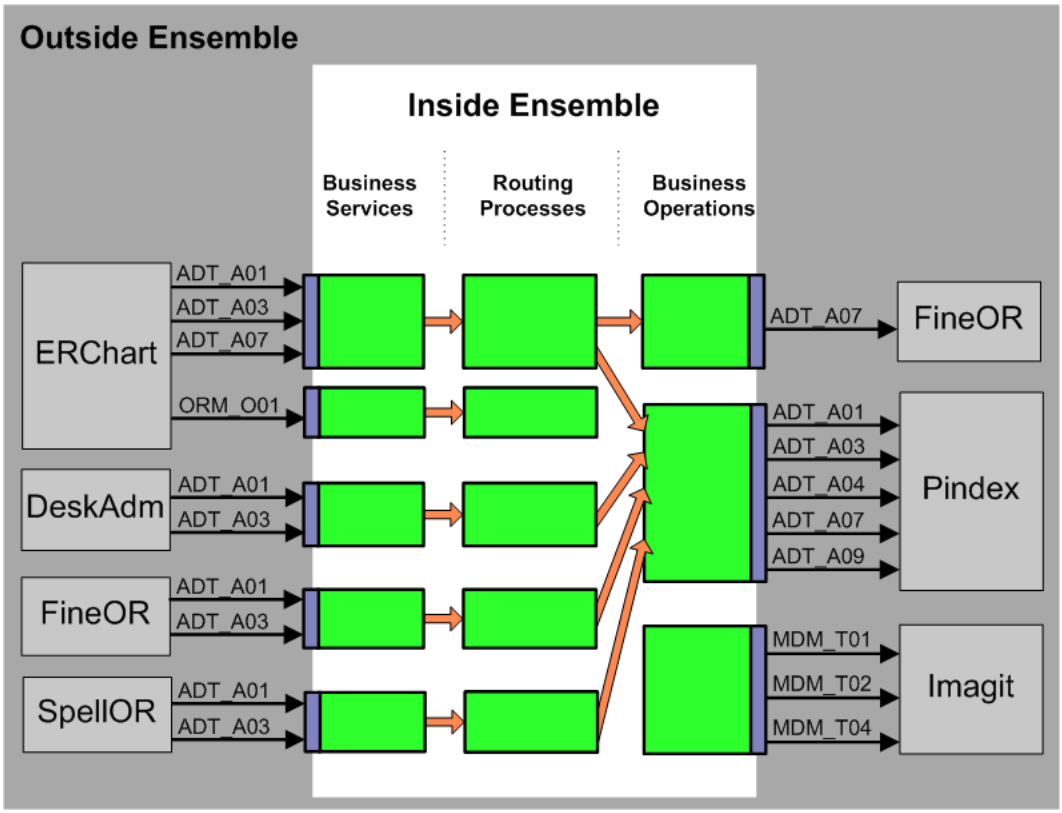
- Provide one business operation for each application that receives HL7 messages from Ensemble. This is an application that has at least one entry in the **Receives** column of the production spreadsheet.

  One business operation can send all the different message structures for the same application. This is usually the appropriate design. When you set up a business operation for this purpose you generally intend for it to stay connected to its application system all the time. However, as for business services, there might be variations on this model.

The guiding principle is to keep your design modular. That is, develop one interface at a time and use one routing process for each interface. This contrasts with a model in which a single large routing process serves as the routing engine for *all* interfaces. There are a number of reasons why many routing processes are better than one:

- Each routing rule set is simpler and easier to maintain because it only covers the cases required by one interface. It is easier to share and reuse work that is self-contained and solves a simple set of problems.
- Interfaces become easy to develop, replace, and maintain individually. If the enterprise must remove or upgrade a particular application, only those routing processes that deal with that application need to be touched. If an interface goes down, only that interface needs to be disabled, diagnosed, repaired, and retested before you can bring it back up.

  This is much cleaner than requiring you to retest and validate all of the rules in one large routing process every time you need to add, remove, or repair one interface. These considerations become especially important as some of your interfaces go live, while others are still in development. Each addition to a routing process that is already in production might require you to retest and validate hundreds of existing rules. The following figure shows a routing production with multiple interfaces:



Each configuration item in the production has a specific role to play. Keep each item to its role:

- Keep business services and business operations simple. In general, it is not necessary to write your own code for business services or business operations. Choose the built-in classes that Ensemble provides. This choice gives you the correct adapters automatically. Configure these classes using Management Portal settings.
- Place all complex activities under control of a routing process. If an interface is simple, its routing processes need not be complex, but if complexity exists, it belongs in a routing process, not in a business service or business operation. To accomplish tasks, a routing process can chain to other routing processes, or it can invoke business rules, routing rules, data transformations, business processes, or custom code.

2.5 Naming Conventions

This topic explains the importance of naming conventions and provides examples.

Typically you will develop your Ensemble production incrementally, one interface at a time. There is a temptation to name items "as you go," and to allow naming conventions to grow incrementally along with the project. InterSystems recommends you *do not* take this incremental approach to naming conventions.

Remember that a full HIS system can involve hundreds of interfaces. Anticipate the task of managing the routing production once all of these interfaces are included. The name of each component should clearly identify its purpose. This way, developers and system administrators can easily predict a component's name based on its function.

InterSystems recommends that you establish a full set of naming conventions at the start of the project, and then stick with them. This alleviates the need to backtrack into your production configuration just to correct names.

Any convention that clearly identifies components by function is fine. The following is a full set of naming conventions that several Ensemble customers have used successfully:

- For rules on configuration item names, see "Configuration Names," in *Configuring Ensemble Productions*.
- List the individual segments that you will assemble to create names. These might include:
  - Keywords:
    - `From` for incoming
    - `To` for outgoing
    - `Router` for routing
    - `Rules` for rule sets
    - A base schema category number — 2.1, 2.2, 2.3, 2.3.1, 2.4, 2.5, 2.5.1, 2.6, 2.7, or 2.7.1— for schemas
  - A short name for each application
  - Keywords that identify a general class of message structures (ADT, ORM, etc.)
  - Full message structure names (ADT_A01, ADT_A04, ORM_O01, etc.)
- Keep each segment of the name brief (3–6 characters)
- Remember that names are case-sensitive

For each element of the production, assemble a name from the segments that you have defined:

- Business services: `From`*SourceAppName*
- Business operations: `To`*TargetAppName*
- Routing processes: *SourceAppName*`Router`
- Routing rule sets: *SourceAppName*`Rules`
- Data transformations:

  SourceAppNameSourceMessageType`To`*TargetAppNameTargetMessageType*
- Custom schema categories: *SourceAppNameBaseSchemaNumber*

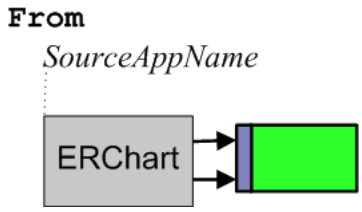
For more complex designs:

- Business services: `From`*SourceAppNameMessageTypes*
- Business operations: `To`*TargetAppNameMessageTypes*
- Routing processes: *SourceAppNameMessageTypes*`Router`
- Routing rule sets: *SourceAppNameMessageTypes*`Rules`

The following topics provide explanations and examples for these naming conventions.

2.5.1 Business Services

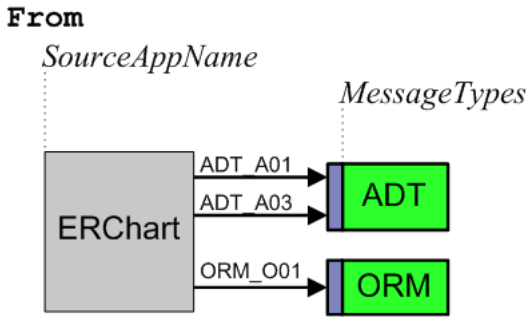
Convention



Examples

`FromERChart`, `FromFineOR`, `FromDeskAdm`

Variation



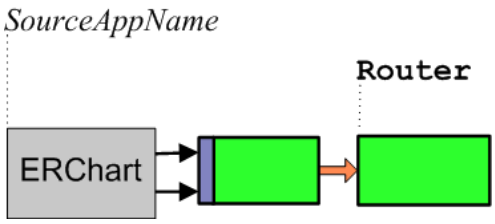
If you have decided to organize an interface so that one application sends messages into Ensemble via more than one business service, also include a keyword that classifies the *MessageTypes* that pass through each business service. A typical *MessageTypes* keyword uses the first three letters of the HL7 Version 2 message structure, such as ADT, ORM, or ORU. In this case the convention for the business service name is `From`*SourceAppNameMessageTypes*.

Examples

`FromERChartADT`, `FromERChartORM`, `FromERChartOther`

2.5.2 Routing Processes

Convention

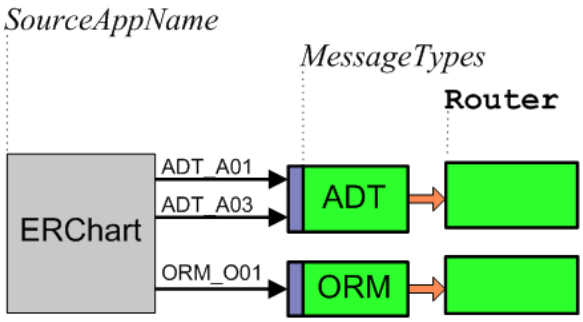


Examples

`ERChartRouter`, `FineORRouter`, `DeskAdmRouter`

If you have decided to organize an interface so that one application sends messages into Ensemble via more than one business service, also include a keyword that classifies the *MessageTypes* that pass through each business service to its routing process. In this case the format for the routing process name is as follows:

Variation



Examples

`ERChartADTRouter`, `ERChartORMRouter`, `ERChartOtherRouter`

2.5.3 Routing Rule Sets for HL7

Convention

Each routing process has an associated rule set whose rules determine message destinations. The rules direct the messages to data transformations or business operations based on message type, message contents, time of day, or other factors. Name each rule set to match its routing process. That is, for each *SourceAppName*`Router`, call the rule set *SourceAppName*`Rules`. For each *SourceAppNameMessageTypes*`Router`, call the rule set *SourceAppNameMessageTypes*`Rules`.

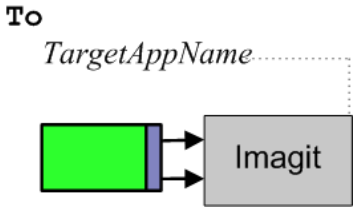
Examples

`DeskAdmRules`

`DeskAdmORMRules`

2.5.4 Business Operations

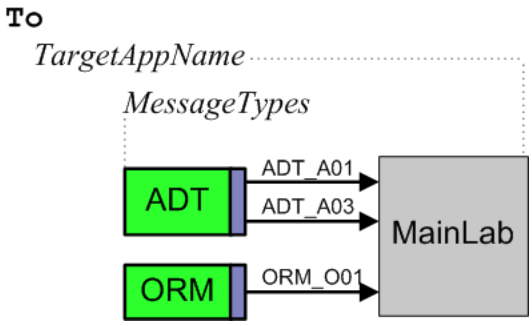
Convention



Examples

`ToImagit`, `ToFineOR`, `ToPindex`

Variation



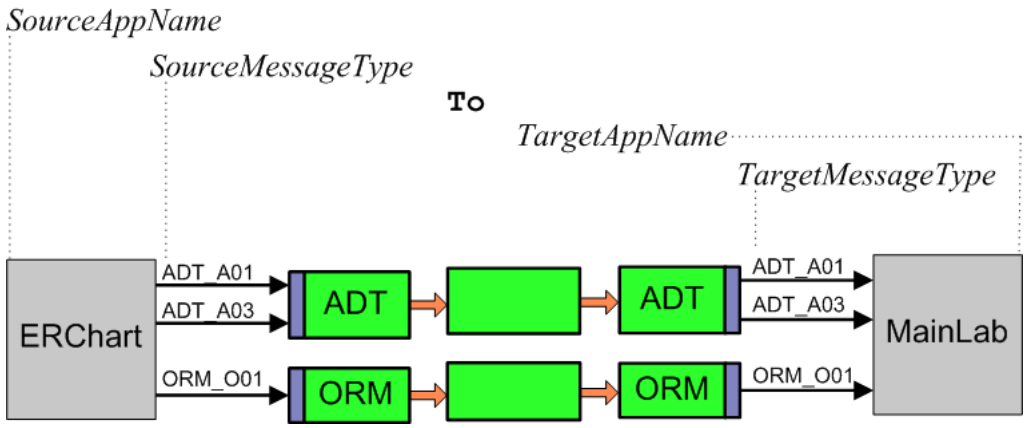
If you have decided to organize an interface so that one application receives messages from Ensemble via more than one business operation, also include a keyword that classifies the *MessageTypes* that passes through each business operation. A typical *MessageTypes* keyword uses the first three letters of the HL7 Version 2 message structure, such as ADT, ORM, or ORU. In this case the convention for the business operation name is as follows:

Examples

`ToMainLabADT`, `ToMainLabORM`, `ToMainLabOther`

2.5.5 Data Transformations

Convention



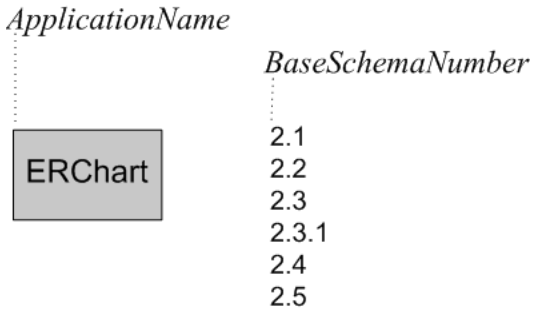
Remember that the *SourceMessageType* and *TargetMessageType* may have the same name, but represent the same message structure in different schema categories.

Examples

`ERChartADTA03ToMainLabADTA03`, `ERChartADTA02ToMainLabADTA01`

2.5.6 Custom Schema Categories

Convention



The *ApplicationName* can represent a sending application or a receiving application.

Examples

`ERChart2.2`, `Imagit2.3.1`, `OurFavorite2.5`

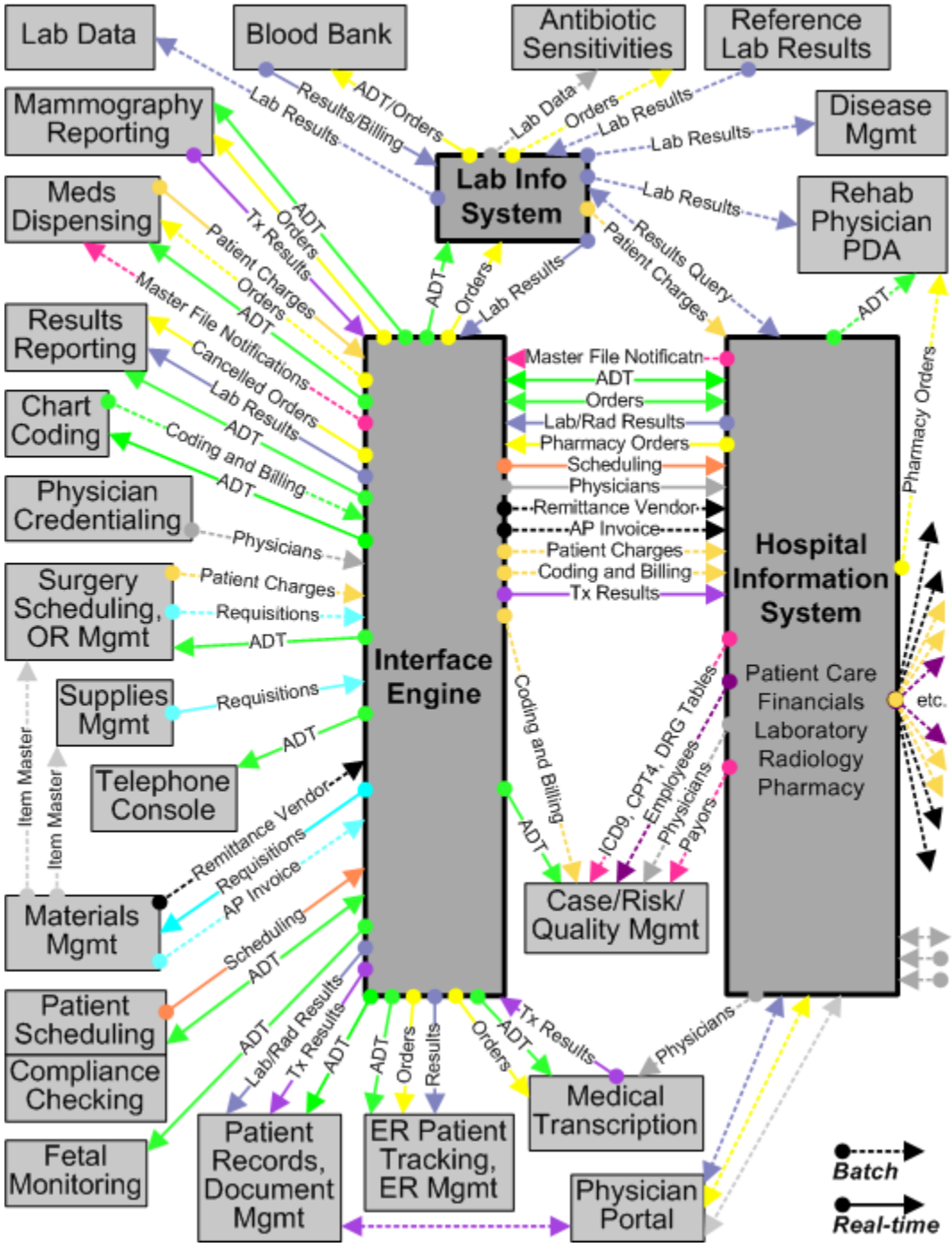
3

Converting Interfaces to Ensemble

The following diagram shows some of the information technology required to run a typical hospital. The diagram reflects the complex challenges faced by hospital IT staff. In the diagram, the various end-user applications exchange HL7 messages with the lab information system and hospital information system (HIS). An interface engine routes HL7 messages between the systems. Not all hospital applications are shown; the focus is on the HL7 messages that enter and leave the interface engine.

This chapter assumes that for each clinical application, you might need to define both inbound and outbound interfaces. The figure illustrates this by providing a circle at the source of each type of message and an arrow at each destination. Various combinations of inbound and outbound interfaces are shown. Within this chapter, an interface is inbound or outbound relative to the clinical application. An inbound interface receives messages from the HIS or lab system via the interface engine. An outbound interface sends messages to the HIS or lab system via the interface engine. Other interfaces within the hospital IT configuration are outside the scope of this chapter, even if they appear in the diagram.

In the diagram, solid arrows represent real-time information exchange; dotted arrows represent batch transmissions. Variations in arrow color highlight the different categories of data: green for admitting, yellow for pharmacy, gold for patients, orange for scheduling, pink for master file notifications, purple for transactions, aqua for requisitions, blue for laboratory, gray for physicians, and black for vendors.



Often when you bring Ensemble into such a configuration, it is for the purpose of replacing an existing interface engine with Ensemble. In order to do this you must:

1. Use an Ensemble production as a routing engine, as described in previous chapters.
2. Convert each of the existing interfaces to use the Ensemble routing engine instead of the previous interface engine.
3. Once all of the interfaces are converted, remove the previous interface engine from the configuration.

The conversion approach in this chapter leaves the existing interface engine in place and converts the interfaces one by one to use Ensemble. Unconverted interfaces continue to use the previous engine while you test and confirm that the new interface works properly using Ensemble. As soon as the converted interface works correctly, you begin work on the next

unconverted interface. This incremental change policy ensures the best chance of uninterrupted, error-free service during the conversion.

You could take an approach that converts groups of related interfaces at the same time. That is, you could decide to convert all of the interfaces that convey messages to and from the Blood Bank application, then move on to Fetal Monitoring. Still, you may find that the work boils down to converting one interface at a time.

This chapter explains how to convert one existing HL7 interface to use Ensemble. The general steps are as follows. Each section of this chapter describes one of these steps in detail:

1. Back up the production
2. Describe the interface
3. Choose schema categories
4. Create data transformations
5. Define routing rule sets
6. Add business operations
7. Create or edit a routing process
8. Add business services
9. Test the interface
10. Deploy the interface

The procedure in this chapter reverses the order in which you would usually describe an Ensemble production. (“An HL7 business service receives an incoming message...”) However, experience with interface conversion shows that if you develop the elements top-down, beginning with the business service and working down to the custom schema, you must often backtrack to previous steps to fill in or correct the names of the lower-level items, in the various forms where you have configured higher-level items.

The procedure in this chapter takes a bottom-up approach, reducing backtracking to a minimum by creating the lower-level items first, so that their names are known when it is time to configure the higher-level items. Bottom-up order works as shown in this chapter: Steps 1 and 2; Steps 3, 4, 5, 6, 7, 8; Steps 9 and 10.

You might have success performing this procedure in top-down order, especially if you have well-established naming conventions, which reduce simple errors. Top-down order works as follows: Steps 1 and 2; Steps 8, 7, 6, 5, 4, 3; Steps 9 and 10.

3.1 Backing Up the Production

The following steps outline a general-purpose backup procedure for Ensemble productions that use HL7. The exact steps will vary depending on your packaging and naming conventions for Ensemble classes, custom classes, routing rules, and custom HL7 category definitions.

Note: Regarding XML backup files: If you use a UNIX® system, never FTP a backup XML file in binary mode. Regular FTP will convert this file from DOS to UNIX® properly, but binary FTP might not.

3.1.1 Backup from Studio

To perform a full backup from Studio:

1. Start Studio and change to the namespace that you are backing up.
2. Export your own package(s) of classes from Studio. This should export all of your business service, business process, business operation, data transformation, and utility classes. To back up a package:
   - Select the package name in the Workspace window
   - Right-click and select **Export** from the context menu.
   - Select the **Export in XML format** check box, choose an appropriate file name, and click **OK**.

   If necessary, rather than choosing a package you can select individual classes for export from Studio:
   - Select **Tools** and **Export**, then click **Add**.
   - Choose **Files of type** `All Files`, enter **File name** `*.cls`, then click **Open**.
   - Select your classes from the list and click **Open** to add them to the **Export** list.
   - Select the **Export in XML format** check box, choose an appropriate file name, and click **OK**.
3. Export your own business rules (**File name** `*.rul`).
4. Export your own Custom Schemas (**File name** `*.hl7`).
5. The resulting XML file(s) can be imported into any Ensemble-enabled namespace.

3.1.2 Backup from ObjectScript

The steps in the above procedure can be cumbersome if you need to repeat them often from Studio. You should code them into an ObjectScript method as soon as is convenient. The exact code will vary, depending on your packaging and naming conventions for Ensemble classes, custom classes (including data transformations), business rules, and custom category definitions. The general idea is that an ObjectScript version of this procedure would create a single master list of all items to be exported, and then export everything on the list to a single XML file, appropriately dated. In detail:

1. Use the $system.OBJ.GetPackageList method with the ObjectScript $ORDER or $O function to assemble a list of all classes (`*.cls`) in each of the packages of interest. The code for *each* package will look something like the following example for a custom package named `Mutro`:

   ```
   do $system.OBJ.GetPackageList(.tList,"Mutro","ars")
   set class="" for  {
     set class=$o(tList(class)) quit:class=""
     set tFullList(class_".cls")=""
   }
   ```

   For more information about ObjectScript functions like $ORDER and $EXTRACT, see the "ObjectScript Functions" chapter in the *Caché ObjectScript Reference*.
2. Use the `^EnsHL7.Schema` global and the ObjectScript functions $ORDER and $EXTRACT to assemble a list of all custom schemas (`*.hl7`) within the Ensemble-enabled namespace. In the following example, the list is easily created because the customer used a naming convention for custom schemas; all names start with the prefix `Mutro_`:

   ```
   set cat="" for  {
   set cat=$o(^EnsHL7.Schema(cat)) quit:cat=""
   if $e(cat,1,6)="Mutro_" set tFullList(cat_".hl7")=""
   }
   ```
3. Set the output directory, create the XML filename using a time stamp, and export the files. Something like:

```
; Normalize the directory name and figure the file name
if pDir'="" set pDir=$tr(pDir,"\","/")
if $e(pDir,$l(pDir))'="/" set pDir=pDir_"/"
for count=1:1  {
  set filename=pDir_"backup_"_$zdate($h,8)_"_"_count_".xml"
  quit:##class(%File).Exists(filename)=0
  }
;
; Export all the items on the list
Set tSC=$system.OBJ.Export(.tFullList,pDir_"backup_"_$zdate($h,8)_".xml")
Do:('tSC) $system.Status.DisplayError(tSC)
;
Set tSC=$system.OBJ.Export(.tFullList,filename)
Do:('tSC) $system.Status.DisplayError(tSC)
;
write !,"Backup to file "_filename_" done.",!
```

The above example expects a parameter *pDir* to be passed to the method, but *pDir* can quite easily be set to a specific directory by the backup method. The example uses the ObjectScript functions $TRANSLATE, $EXTRACT, $ZDATE, and $HOROLOG as well as the %File.Exists() and $system.OBJ.Export() methods.

3.2 Describing the Interface

Each clinical application in use at the enterprise provides its own HL7 application specification document. The specification document explains which types of HL7 event the application can send (or receive), and which message segments and pieces of each segment the application sends (or expects) for these events. For example, for an MSH segment the HL7 application specification document shows exactly what the segment will look like, what will be hardcoded, what format will be used for strings, etc. The information is extremely detailed. The administrator of each clinical application can show you this document for his or her application.

The source and target applications each have an HL7 application specification document. However, any single interface between these applications, such as the interface that you are converting, uses a small subset of the possibilities listed in the specification document. Your best source for a description of an interface is actually the interface definition that you are replacing.

Open the existing interface definition (for example, in eGate, the collaboration rule). With this definition in view, open a text file and type into it a brief description of what the interface does. Do not try to reproduce the coding conventions of the existing definition (LOOP, CASE, etc.) in detail. Simply identify:

- The message segments that you expect to arrive from the source
- How the interface changes them before sending them to the target

In preparing a description of an interface that you are converting to Ensemble, it can be helpful to compare terminology between the technology you are replacing and Ensemble. As an example, the following table lists analogies between See-Beyond and Ensemble terminology.

SeeBeyond	Ensemble
Collaboration rule	Interface definition
COPY or DUPLICATE	<assign> statement in a DTL data transformation
DISPLAY	<trace> statement in a DTL data transformation
LOOP	<foreach> or the shorthand notation () in DTL
IF	<if> statement in a DTL data transformation
CASE	Routing rule
FUNCTION	Code in business services or business operations; also <code>
CHANGE-PATTERN	Text replacement using functions like $CHAR and $PIECE

3.2.1 The Inbound Interface

Suppose you have a source application from which a variety of HL7 messages arrive. However, only ancillary orders are of interest, and then only for certain types of patients, certain types of orders, and when the message is genuine and not a test of the system. In describing which messages are of interest from this source, you might type something like:

```
Send only:
  Message Type "ORM_O01 "
  PV1:18 Patient Type = "ER" or "ECM" or "ERQ"
  ORC:1 Order Control = "NW" or "CA" or "OC"
  PID:5 Last Name not "TEST"
```

3.2.2 The Outbound Interface

How the interface changes these message segments, before sending them, depends on what the target application needs to receive. Based on your knowledge of the interface and the sources of information listed above, you might type something like:

```
Copy source MSH to target.
Set target MSH:11 Processing ID = "P".
Copy source MSH:10 Control ID to target MSH:13 Sequence Number.

Copy source PID to target.
Set target PID:1 Set ID = "1".
Copy source PID:2 Patient External ID
  to the 2nd repetition of target PID:3 Patient Internal ID and
  set its Identifier Type = "PI".
Null out target PID:2.

Copy source ORC to target.
If source ORC:1 Order Control = "CA" or "OC",
  then set target ORC:5 Order Status to "CA".
If source ORC:2.1 Placer Order Number is empty, then
  copy source ORC:3.1 Filler Order Number
  to target ORC:2.1 Placer Order Number and
  set target ORC:2.2 Placer Application = "ULTRA".
Set target ORC:3.2 Filler Application = "ULTRA".

Copy source OBR to target.
If source OBR:2.1 Placer Order Number is empty, then
  copy source OBR:3.1 Filler Order Number
  to target OBR:2.1 Placer Order Number and
  set target OBR:2.2 Placer Application = "ULTRA".
Set target OBR:3.2 Filler Application = "ULTRA".
Set target OBR:4.3 Coding Scheme (SIM Department) = "LAB".
```

3.3 Choosing Schema Categories

In this step, you must determine which HL7 schema will be used for the inbound and outbound sides of the interface. This begins with an approximate choice, which you refine by testing messages against that schema in the Management Portal.

Each clinical application has an HL7 application specification document that identifies the HL7 schema version that it uses when sending messages, and that it expects when receiving messages. You can confirm this version against the message data sent by the application. Its outgoing message MSH segment states an HL7 version number that (in theory) announces what HL7 version the application is using.

You may discover that neither source of information is strictly accurate. Clinical applications can update their HL7 versions at different times so that one application uses a new HL7 version and another does not. Applications can upgrade HL7 versions or add or change Z segments or without changing the contents of the MSH segment or documenting the change.

3.3.1 Choosing an Incoming Schema Category

Choose a schema category for the inbound side of the interface (as the message enters Ensemble via an HL7 business service) as follows:

1. Capture some sample messages from the source application in text files. You can use archived file copies of these messages, if you have them.
2. In the Management Portal, use the HL7 Schema Structures page to find an HL7 standard schema that seems to match the structure of the message segments you are interested in.
3. Use the Message Viewer page to test your text file messages against your chosen schema. Try different choices of schema until you find the standard schema that *most closely* matches your message structure.

   Tip: Do not be concerned with the specific HL7 separator characters expected on each side of the interface. The HL7 business operation handles these appropriately.

4. The messages from the source application might contain extra segments that are not in the schema. If so, when you parse the messages using the **Message Viewer** option, the messages may either accept the message, or generate a bad message error, as follows:

   - If the message contains extra Z segments, and the schema defines those Z segments, the message is okay.
   - If the message contains extra Z segments, but the schema does not define *any* Z segments, the message is okay.
   - If the message contains extra Z segments, and the schema defines Z segments, but the Z segments found in the message do not match the Z segments in the schema, it is a bad message.
   - If the message contains extra non-Z segments, then regardless of Z segments, it is a bad message.

   For background, see "Controlling Message Validation" in *Ensemble Virtual Documents*. For HL7 details, see the reference for the Validation setting in the *Ensemble HL7 Version 2 Development Guide*.

5. If the messages trigger one of the bad message error conditions, try choosing a different standard schema.

   If that does not work, create a custom HL7 category definition to accommodate the nonstandard message structure. See "Creating Custom Schema Categories" in *Ensemble Virtual Documents*.

3.3.2 Choosing an Outgoing Schema Category

Repeat a similar procedure to choose a schema for the outbound interface.

3.4 Defining Routing Rule Sets

Create routing rule sets. For information, see "Defining Routing Rule Sets for HL7" in the *Ensemble HL7 Version 2 Development Guide*.

3.5 Creating Data Transformations

Create DTL data transformations. For information, see "Defining DTL Data Transformations for HL7" in the *Ensemble HL7 Version 2 Development Guide*.

3.6 Adding Business Operations

The business operation for an HL7 interface controls the outgoing message transmission to the target application. For information on adding HL7 business operations, see "Configuring the Production" in the *Ensemble HL7 Version 2 Development Guide*.

For testing purposes, it is useful to configure a production with two HL7 business operations that have the same configuration name:

- One is an FTP or TCP business operation that controls the FTP or TCP transmission of HL7 messages across the interface when the production is running normally.
- The other is a File business operation that sends HL7 messages to a file during testing or troubleshooting of the interface.

By Ensemble convention (the items have the same configured name) only one of these configuration items can be enabled at a time. Enable one or the other depending on whether you want a "test" environment (the File operation) or a "live" environment (the TCP or FTP operation).

The steps to create a "live" business operation and its "test" counterpart are as follows:

1. Examine the target application to see which separators it expects HL7 messages to contain.
2. Create the "live" (FTP or TCP) HL7 business operation.
3. Use similar steps to create a "test" (File) HL7 business operation.

   Give the "test" (File) operation the same configuration **Name** as the "live" one.
4. Use the **Enabled** field to enable and disable the "live" (FTP or TCP) or "test" (File) versions of the business operation. Only one business service of the same name can be active at one time.

For additional details, see "Working with Multiple Versions of a Business Host" in *Configuring Ensemble Productions*.

3.7 Creating or Editing a Routing Process

To enable your new interface to work with the Ensemble routing engine, you must add a routing process to the production that:

1. Tells how to interpret data from the source (identifies a routing rule)

2. Tells where to send the interpreted data (identifies a business operation)

You can create a new HL7 routing process. For details, see "Configuring the Production" in the *Ensemble HL7 Version 2 Development Guide*.

3.8 Adding Business Services

The business service for an HL7 interface receives the incoming messages from the source application. For information on adding HL7 business services, see "Configuring the Production" in the *Ensemble HL7 Version 2 Development Guide*.

For testing purposes, it is useful to configure a production with two HL7 business services that have the same configuration name:

- One is an FTP or TCP business service that receives HL7 messages from the source application via FTP or TCP when the production is running normally.
- The other is a File business service that receives HL7 messages from a file during testing or troubleshooting of the interface.

By Ensemble convention (the items have the same configured name) only one of these configuration items can be enabled at a time. Enable one or the other depending on whether you want a "test" environment (the File service) or a "live" environment (the TCP or FTP service).

The steps to create a "live" business service and its "test" counterpart are as follows:

1. Create the "live" (FTP or TCP) HL7 business service.
2. Use similar steps to create a "test" (File) HL7 business service.

   Give the "test" (File) service the same configuration **Name** as the "live" one.
3. Use the **Enabled** field to enable and disable the "live" (FTP or TCP) or "test" (File) versions of the business service. Only one business service of the same name can be active at one time.

For additional details, see "Working with Multiple Versions of a Business Host" in *Configuring Ensemble Productions*.

3.9 Testing the Interface

Generally you need to maintain a separate "test" production that is an exact copy of the production that runs "live" at your health facility. Develop the new Ensemble interface within the "test" production. When that is done, you can migrate a copy of the new interface to the "live" production.

To test a new interface:

1. Capture some sample messages from the source application in files.
2. In the "test" production, enable the File business service and the File business operation and send the messages as files.
3. Examine the resulting HL7 message data in the output files to see if it meets the requirements for the target application.
4. If necessary, adjust the interface elements and retest.
5. Selectively disable the "test" (File) versions and re-enable the "live" (FTP or TCP) versions of the business service and business operation, still within the "test" production.

3.10 Deploying the Interface

Once you have completed testing the Ensemble interface in the test production, it is time to add the new Ensemble interface elements to the production that you are running live. To do this:

1. Back up the complete live production as described in the first step of this procedure.
2. Export the new elements of the test production, created in the previous steps:
   - Export the new classes for the interface: business process, data transformation(s), business operations, business services, and any utility classes.
   - Export the business rule(s) `*.rul`
   - Export the custom schema(s) `*.hl7`
3. Copy the new XML <Item> elements that the new interface added to the `XDATA` section of the production class. For example:

Configuration Item	Sample Start of <Item> Element
Business service (Test)	`<Item name="App1toApp2_In"`
Business service (Live)	`<Item name="App1toApp2_In"`
Business process (if new)	`<Item name="App1toApp2"`
Business operation (Test)	`<Item name="App1toApp2_Out"`
Business operation (Live)	`<Item name="App1toApp2_Out"`

4. Transfer the new interface elements to the live production as follows:
   - Suspend or shut down the live production.
   - Import and compile the new classes, rules, and schema.
   - If your choice was to edit the ACTIONS string on the business process on the “live” system rather than importing the business process as a class, do so now. Compile the edited class.
   - Paste the new <Item> elements into the `XDATA` section of the production class. Compile the production class.
   - Resume or restart the live production.
5. Optionally, configure alerts for the new production elements:
   - HL7 business service and business operations have a configuration setting called **Alert On Error**. When **Alert On Error** is set to True, as soon as the item encounters any type of error condition it automatically triggers an alert. An alert writes a message to the Ensemble Event Log and can also send notification to a user via email or pager. Setting **Alert On Error** to False disables the option.
   - **Alert Grace Period** (for business services) and **Alert Retry Grace Period** (for business operations) provide useful constraints on the frequency of alerts when they are enabled.

   Note: This procedure describes how to add interfaces to the “live” production one by one. Alerts are easy to configure as described in this step, once they have been set up. To set them up for the “test” or “live” production, see “Pager and Email Alerts” in the *Ensemble HL7 Version 2 Development Guide*.
6. Ensure that the new interface is processing all new messages.

7. Disable or clean up the previous interface technology:
    - Ensure that all previously pending requests are satisfied and all queues are empty.
    - Disable the old interface. For eGate elements, disable the “start automatically” option.