Best Practices for Creating Productions

Version 2024.1
2024-07-02

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

Best Practices for Creating Productions
InterSystems IRIS Data Platform Version 2024.1 2024-07-02

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

List of Tables

1

Best Practices for Production Development

This topic is a general overview that prepares team members to work on production projects. It outlines development tasks and identifies sources of information about InterSystems IRIS® data platform and about enterprise integration. The information pertains to all types of production projects.

1.1 Project Goals

The goal of any production development project is to connect two or more systems. 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 Interoperability Productions.

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

1.2 Project Delivery

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

- A 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.
- A 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.
- InterSystems IRIS 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 InterSystems IRIS 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 InterSystems IRIS documentation. Your best starting points are the following items. Each provides further cross-references:

- Introducing Interoperability Productions introduces the terminology and basic concepts.
- Developing Productions describes the development tasks.
- Configuring Productions describes how to perform the configuration tasks related to creating a production.
- Design Model for a Routing Production and Converting Interfaces to Production Elements apply to deploying interface routing solutions.

It is also important to understand the platform features of InterSystems IRIS that support productions. A useful starting point is the Orientation Guide for Server-Side Programming.

Helpful background materials include the following:

- Defining and Using Classes
- Using ObjectScript
- Using the InterSystems SQL Gateway
- Creating Web Services and Web Clients

Other useful development guides include:

- Using InterSystems SQL
- Using InterSystems IRIS Multidimensional Storage (Globals)
- High Availability Guide

Language reference materials include:

- ObjectScript Reference

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

- System Administration Guide

- About InterSystems Security

2

Design Model for a Routing Production

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

This topic 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 topic simply describes those aspects of a routing production that are unique to InterSystems IRIS® 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

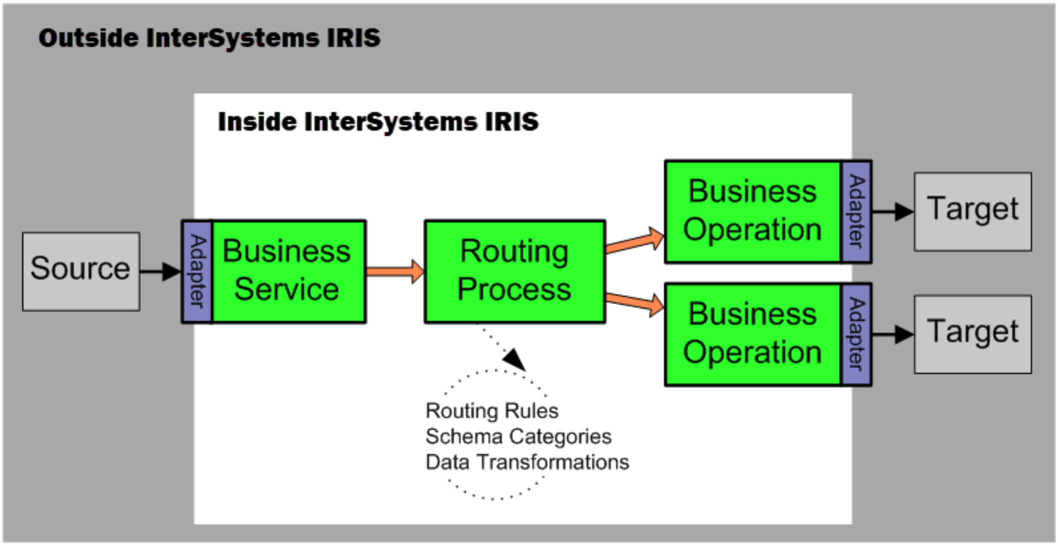
A production consists of *configuration items*. There are three types of configuration item:

- *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:

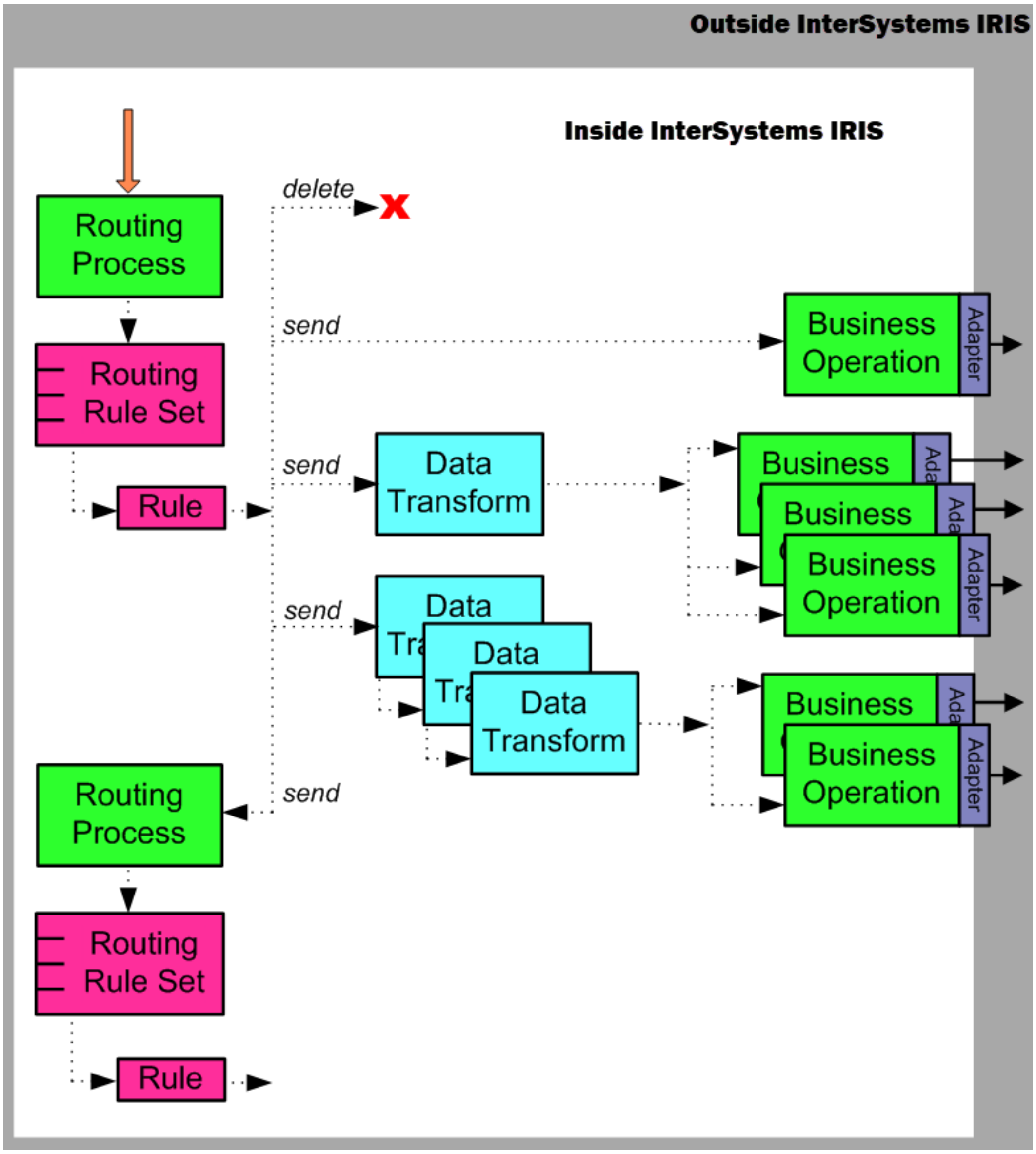
  - *Routing rules* direct messages to their destinations based on message contents.
  - *Schema categories* provide a means to validate and access message contents.
  - *Data transformations* apply changes to prepare messages for their destinations.

  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 InterSystems IRIS. If this destination is a business operation, the message then leaves the production on its way to the target application.



2.2 Application Specification

Ideally, an application provides an application specification document, or implementation guide, that explains which types of messages 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 application can show it to you, and can explain to you which of the many possible messages are actually in use in the enterprise.

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 the application sends or expects to receive. Ask the application administrator to provide you with a sampling of the message data, saved in files.

Background information that supports these tasks can be found in these topics:

- Creating Custom Schema Categories
- Controlling Message Validation
- Choosing Schema Categories

2.3 Production Spreadsheet

It is helpful to maintain a spreadsheet that organizes your information system, application by application. 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 InterSystems IRIS, 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 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 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 the production 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 production remains easy to understand, scale, and maintain at each stage of its development:

- Provide one business service for each application that sends messages into InterSystems IRIS. 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 InterSystems IRIS via one business service that may or may not stay permanently connected to any single instance of the application.

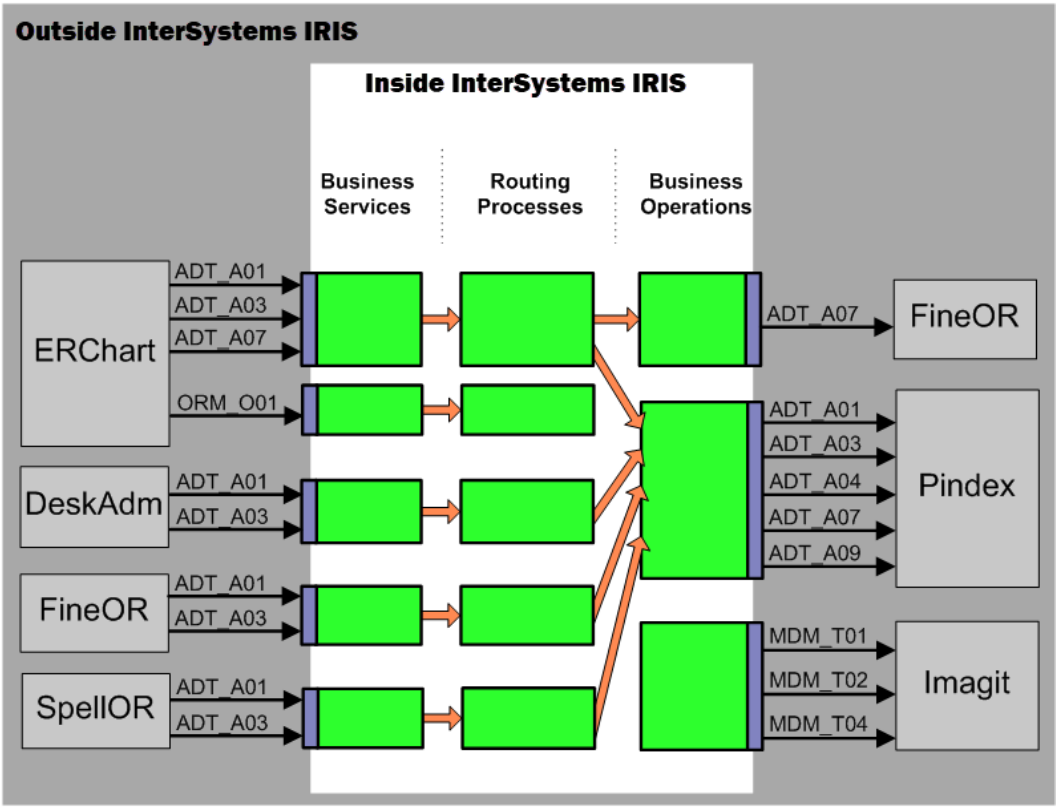
- Provide one routing process for each business service. The routing process can route messages based on the contents of the message itself, or information stored in InterSystems IRIS. If the routing depends on the contents of other messages, or requires invocation of external applications to determine 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.
- Provide one business operation for each application that receives messages from the production. 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 InterSystems IRIS 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 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 InterSystems customers have used successfully:

- For rules on configuration item names, see Configuration Names.
- 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*

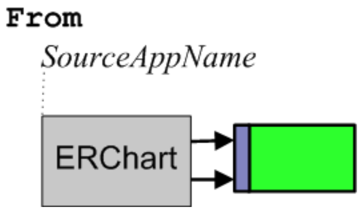
For more complex designs:

- 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

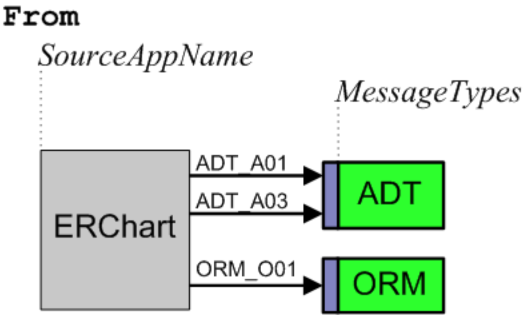
Convention



Examples

`FromERChart`, `FromFineOR`, `FromDeskAdm`

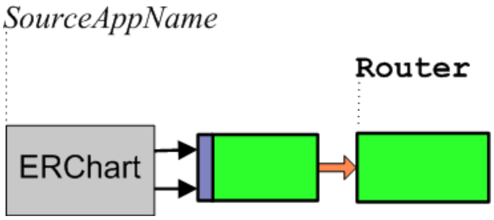
Variation



If you have decided to organize an interface so that one application sends messages into InterSystems IRIS 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 message structure. In this case the convention for the business service name is `From`*SourceAppNameMessageTypes*.

2.5.2 Routing Processes

Convention

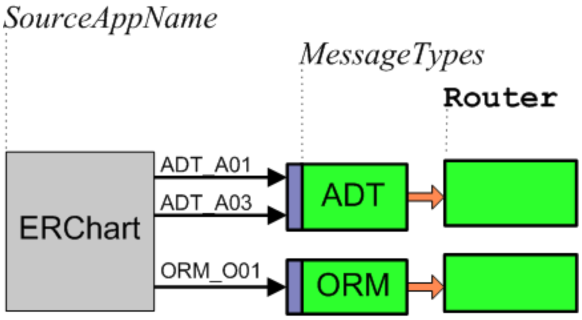


Examples

`ERChartRouter`, `FineORRouter`, `DeskAdmRouter`

If you have decided to organize an interface so that one application sends messages into InterSystems IRIS 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

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`.

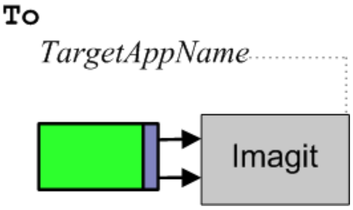
Examples

`DeskAdmRules`

`DeskAdmORMRules`

2.5.4 Business Operations

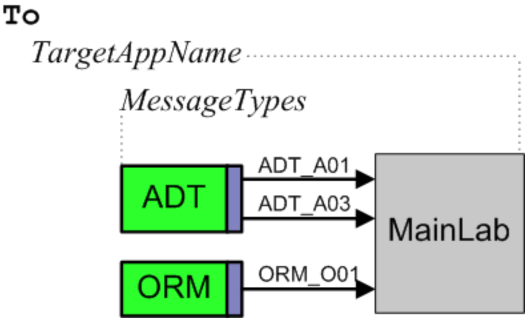
Convention



Examples

`ToImagit`, `ToFineOR`, `ToPindex`

Variation



If you have decided to organize an interface so that one application receives messages from InterSystems IRIS via more than one business operation, also include a keyword that classifies the *MessageTypes* that passes through each business operation.

2.5.5 Data Transformations

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

2.5.6 Custom Schema Categories

Convention

ApplicationName BaseSchemaNumber

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

3

Converting Interfaces to Production Elements

This topic assumes that for each application, you might need to define both inbound and outbound interfaces.

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

1. Use a production as a routing engine.
2. Convert each of the existing interfaces to use the InterSystems IRIS 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 leaves the existing interface engine in place and converts the interfaces one by one to use InterSystems IRIS. Unconverted interfaces continue to use the previous engine while you test and confirm that the new interface works properly using InterSystems IRIS. 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. Still, you may find that the work boils down to converting one interface at a time.

This topic explains how to convert one existing interface to use InterSystems IRIS. The general steps are as follows. Each section 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 reverses the order in which you would usually describe a production. (“A 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 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: 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

To perform a full backup, see Deploying a Production.

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.

3.2 Describing the Interface

Each application in use at the enterprise should provide its own application specification document. The specification document explains which types of messages the application can send (or receive), and which message segments and pieces of each segment the application sends (or expects) for these events. The information is extremely detailed. The administrator of each application can show you this document for his or her application.

The source and target applications should each have an 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. 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

3.3 Choosing Schema Categories

In this step, you must determine which 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 application should have an application specification document that identifies the schema that it uses when sending messages, and that it expects when receiving messages.

3.4 Defining Routing Rule Sets

Create routing rule sets. For information, see Creating and Editing Rule Sets.

3.5 Creating Data Transformations

Create DTL data transformations. For information, see Developing DTL Transformations.

3.6 Adding Business Operations

The business operation for an interface controls the outgoing message transmission to the target application. For information on adding business operations, see Adding Business Hosts.

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

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

By 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 messages to contain.
2. Create the “live” (FTP or TCP) business operation.
3. Use similar steps to create a “test” (File) 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.

3.7 Creating or Editing a Routing Process

To enable your new interface to work with the 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 routing process.

3.8 Adding Business Services

The business service for an interface receives the incoming messages from the source application.

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

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

By 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) business service.
2. Use similar steps to create a “test” (File) 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.

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”. Develop the new 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 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 interface in the test production, it is time to add the new interface elements to the production that you are running live. To do this:

1. Back up the complete live production using the process described in Exporting a Production.

   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.

2. Export the new elements of the test production using the process described in Exporting a Production.

   For example, you might export the following items:

   - The new classes for the interface: business process, data transformation(s), business operations, business services, and any utility classes
   - The business rule(s)
   - The custom schema(s)

   InterSystems IRIS creates a deployment package file in the location that you specify.

3. Deploy the deployment package file with the new elements to the live production.

   For instructions, see Deploying a Production on a Target System.

4. Optionally, configure alerts for the new production elements:

   - 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 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: To set up alerts for the production as a whole, see Monitoring Alerts.

5. Ensure that the new interface is processing all new messages.

6. 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. Disable the “start automatically” option.

4

Managing Databases for Productions

In order to develop a production, you will need to use or create an interoperability-enabled namespace. While you can rely on a single database for your production, it is often desirable to define a separate database to store the data associated with messages. This data has the potential to increase the database size rapidly during times with higher than usual traffic or if a purge process is not scheduled to run. This practice insulates the database associated with your production's code from the database with your message data. This page describes the process, including the recommended globals that should be mapped and guidelines for mapping on both new and existing systems.

4.1 Globals Containing Message Data

The following table contains the recommended globals which should be mapped to a separate messages database. Select the tab for the product you are using. Additionally, if you have any custom message classes that contain message data, you should consider mapping these to the same message database.

Table 4–1: InterSystems IRIS

^Ens.MessageHeaderD
^Ens.MessageHeaderI†
^Ens.MessageBodyD
^Ens.MessageBodyI†
^Ens.MessageBodyS
^EnsHL7.Segment
^EnsLib.H.MessageD
^Ens.Util.LogD
^Ens.Util.LogI†

Table 4–2: InterSystems IRIS for Health

^Ens.MessageHeaderD
^Ens.MessageHeaderI[†]
^Ens.MessageBodyD
^Ens.MessageBodyI[†]
^Ens.MessageBodyS
^EnsHL7.Segment
^EnsLib.H.MessageD
^Ens.Util.LogD
^Ens.Util.LogI[†]

Table 4–3: HealthConnect

^Ens.MessageHeaderD
^Ens.MessageHeaderI[†]
^Ens.MessageBodyD
^Ens.MessageBodyI[†]
^Ens.MessageBodyS
^EnsHL7.Segment
^EnsLib.H.MessageD
^Ens.Util.LogD
^Ens.Util.LogI[†]
^HS.Message.XMLMessageD
^HS.Message.XMLMessageS

Table 4–4: HealthShare

^Ens.MessageHeaderD
^Ens.MessageHeaderI[†]
^Ens.MessageBodyD
^Ens.MessageBodyI[†]
^Ens.MessageBodyS
^EnsHL7.Segment
^EnsLib.H.MessageD
^Ens.Util.LogD

^Ens.Util.LogI[†]
^HS.Message.XMLMessageD
^HS.Message.XMLMessageS
^HS.IHE.Comm8F7E.MIMEAttachmentD
^HS.IHE.Comm8F7E.MIMEAttachmentS
^HS.IHE.Comm8F7E.MIMEAttachmentI[†]

Note: [†]Globals that store indexes for classes cannot be mapped using subscript-level mapping on existing systems. For details, see Mapping Globals on an Existing System

4.2 Globals for Default Database

The following globals should *NOT* be mapped to the messages database and need to remain in the default database for the interoperability-enabled namespace. They are essential for interoperability to function properly:

- **`^Ens.Runtime`**
- **`^Ens.Queue`**
- **`^Ens.JobLock`**
- **`^Ens.JobStatus`**
- **`^Ens.ActiveMessage`**
- **`^Ens.JobRequest`**
- **`^Ens.Suspended`**
- **`^Ens.SuspendedAppData`**
- **`^Ens.Alarm`**
- **`^Ens.AppData`**
- **`^Ens.Rule.Notification`**

4.3 Mapping Globals on a New System

If you set up a separate database for the data associated with messages during the initial configuration of your production, the process of mapping the relevant globals is straightforward. Simply create a new messages database and map all relevant globals to your newly created database. For details on mapping globals to a database, see Add Global, Routine, and Package Mapping to a Namespace.

4.4 Mapping Globals on an Existing System

If you already have a production in use, the process of mapping the existing globals to your new database is more involved. Again, you should start by creating a new messages database. For globals that feature an incrementing ID as the top-level subscript, you should perform a subscript-level mapping. Choose a higher subscript than the current highest subscript of the global. This will allow the global to roll over to the new database when it increments to the designated subscript. For details on subscript-level mapping, see Global Mapping and Subscript-Level Mapping.

For globals that do not have an incrementing ID as the top-level subscript, including globals that store indexes for a class and globals subscripted by $JOB, you can perform a DataMove to map the subscripts to the newly created database.