Infor ION Development Guide

Release 12.0.x

Publication Information

Release: Infor ION 12.0.x
Publication Date: November 20, 2019
Document code: ion_12.0.x_ionopdg_en-us

Contents

About this guide

This guide explains how to adopt ION for new applications that you want to connect to ION.

The guide also explains the contents of the ION Registry, and how you can add metadata for your own documents or for extensions on standard documents.

Intended audience

The document is intended for this audience:

- System Administrators
- Business Process Administrators
- Business Analysts
- Database Administrators
- Application Administrators

Related documents

You can find the documents in the product documentation section of the Infor Support portal, as described in "Contacting Infor".

- *Infor ION Desk User Guide*
- *Infor Operating Service Administration Guide*
- *Infor Federation Services Standalone Installation Guide* - Version 12.0.x
- *Infor ION API Administration Guide*

Contacting Infor

If you have questions about Infor products, go to Infor Concierge at https://concierge.infor.com/ and create a support incident.

If we update this document after the product release, we will post the new version on the Infor Support Portal. To access documentation, select **Search > Browse Documentation**. We recommend that you check this portal periodically for updated documentation.

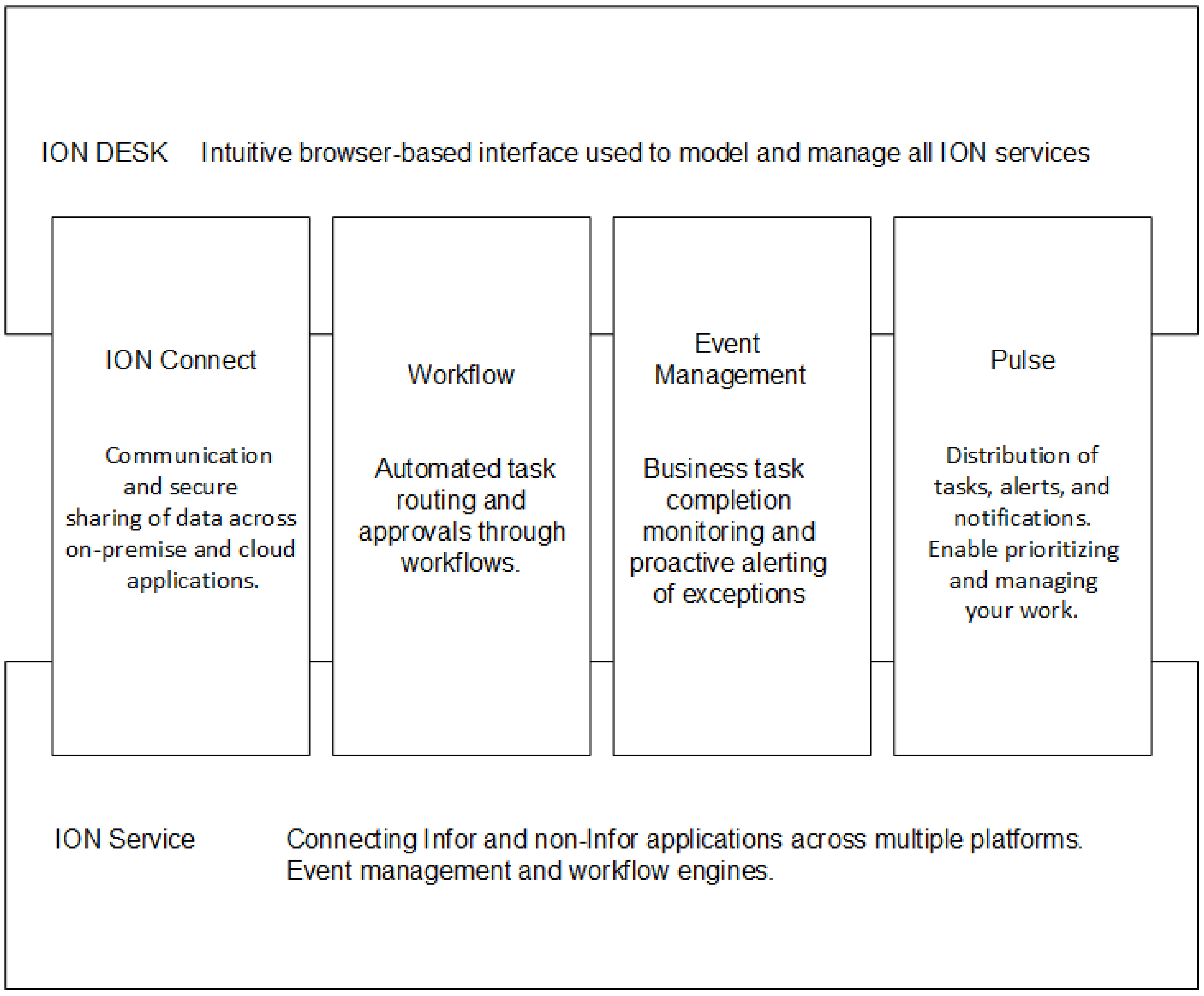
If you have comments about Infor documentation, contact documentation@infor.com.

Chapter 1: Introduction

ION is a new generation of business middleware that is lighter weight, less technically demanding to implement, and built on open standards.

In addition to connectivity with ION, you get workflow and business event monitoring in a single, consistent architecture. ION uses an event-driven architecture. It can pro-actively push data, work activities, and exception notifications to users. The ION Suite includes several powerful services to install and configure.

This diagram shows the ION Suite services:


ION DESK
Intuitive browser-based interface used to model and manage all ION services
ION Connect
Communication and secure sharing of data across on-premise and cloud applications.
Workflow
Automated task routing and approvals through workflows.
Event Management
Business task completion monitoring and proactive alerting of exceptions
Pulse
Distribution of tasks, alerts, and notifications. Enable prioritizing and managing your work.
ION Service
Connecting Infor and non-Infor applications across multiple platforms. Event management and workflow engines.

ION Connect

With ION Connect you can establish connections between applications, which can either be Infor applications or third party applications. A set of connectors is available to connect many types of resources such as Infor applications, databases, message queues, or files. This varies from cloud or on premises. In ION Desk you can model document flows between applications. Such flows can represent a business process. But also more technical flows can be defined. For example, to map data from a third party application to a standard business object document as used by an Infor application. You can also use filtering or content-based routing.

Workflow

In Workflow you can model business processes. A workflow can include tasks to be executed by a user, notifications to be sent to a user, decisions, parallel flows and loop backs. The modeling is done graphically. Workflows can be used to automate approval processes, and for other types of business processes. For example, a review flow consisting of several parallel tasks that are sent to multiple users to review the same document.

Event Management

Using Event Management you can monitor business events that are based on business rules. Users receive an alert when an exception occurs. For example, if a stock quantity is low, if a shipment is late, or if a contract must be renewed.

Pulse

With Pulse you can follow what happens in your organization. Either by following specific business documents or by following alerts, tasks or notifications.

Adopting ION

Adoption of ION is relevant in these situations:

- You want to integrate with other applications.
- You want to extend your application with Event Management, Workflow, or Pulse functionality.

To adopt ION these steps are required:

1 Preparation

Getting the requirements clear is a major factor for success. What is the business case? In particular to identify business case of integration, system(s) to integrate with, data to exchange between systems, and mapping from BODs to data in involved applications.

Event Management or Workflow capabilities, or both, can play an important role in adoption consideration.

An introduction to ION as a product, see the *Infor ION Desk User Guide*.

Additionally, it is important to understand some basic terms that are relevant for ION.

See BODs and messages on page 11.

2 Connect to ION.

Enable the application to connect to ION.

See "Connecting to ION".

3 If required, adopt Event Management, Workflow, or Pulse.

See Adopting Event Management, Workflow, or Pulse on page 48.

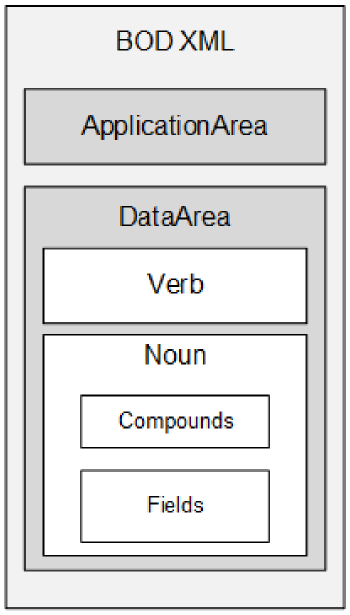
Chapter 2: BODs and messages

This section explains the terminology to adopt ION for an application.

BOD

A Business Object Document (BOD) is an XML document being a generic representation of a business object. A common language used for information integration. Infor have defined a set of standard BODs. At a high level, all BODs have some common characteristics. This standardization makes it easier to understand and use various BODs.

A BOD contains two parts: a noun and a verb.



Noun

A noun is a definition of a set of business data contained in a BOD. The noun represents the properties of one business object. Examples of nouns are SalesOrder, Item, and BusinessPartner. In ION Desk, a noun is called a document.

A BOD is data instance of one noun definition. A BOD message can contain multiple instances of the same noun definition.

Verb

The verb describes the action that is requested for the noun or indicates a response to an action.

A verb can:

- Announce that a business object is created, updated or deleted.
- Indicate a request to create, update or delete a business object.
- Provide a response to a request.
- Report an exception.

This table shows the request verbs supported by Infor:

Request verb	Description
Sync	A synchronization message containing changes that took place to a business object. A Sync message is sent by the owner of the data and can be delivered to any other application for which this information is relevant.
Process	A request to create a business object or to apply changes to an existing business object. A Process message is sent from any application to the application that owns the data. The owner will send an Acknowledge message in response to the Process request. The loaded document can be refused.
Get	A request to get the details for a business object. A Get message is sent to the owner of the data. The owner will send a Show message in response to the Get request.
Load	The Load verb is used when a document is created by an application that will not be the owner. A Load message is sent to the owner of the data. The loaded document cannot be refused.
Post	The Post verb is similar to the Process verb, but it does not trigger the creation of an Acknowledge message.
Update	The Update verb is used when data is changed by an application that does not own the data. The Update verb is similar to the Load verb, in the sense that it must be accepted by the owner. Namely, the Update message informs the owner of the data that an event took place and what data was changed by the event.

This table shows the response verbs supported by Infor:

Response verb	Description
Acknowledge	An Acknowledge response is sent in reply to a Process request. An Acknowledge response indicates whether the object to be processed was accepted, modified, or rejected.
Show	A Show response is sent in reply to a Get request.
Confirm	A Confirm verb is used when a failure happens. A Confirm verb is processed within ION and is not routed to any other application. The Confirm verb is used only for the BOD noun. The ConfirmBOD contains a copy of the original message, to enable an ION administrator to resubmit the same message after fixing the cause of the problem.

If you implement custom nouns, we recommend that you use these verb patterns as applicable:

- Sync
- Process and Acknowledge
- Get and Show

General concepts

The general concepts are explained here.

Tenant
A tenant is a hosting or software as a service (SaaS) concept where all the data of one tenant is always separated from all the data of other tenants. There is no cross sharing or viewing of data with other tenants. This concept requires all participants in the messaging to share the same identity for the same tenant. Therefore, a Tenant Id such as "infor" must have exactly the same meaning on every system in the messaging space. Tenants can also be used to separate data for a single on-premises customer, for example, separating data for a test environment from data from the production environment. Tenant Id is alphanumeric, maximum length is 22 characters.

Message Id
The Message Id is the unique identifier required for each message. The Message Id is used to detect duplication and to refer to the message in other messages, especially response documents and ConfirmBOD messages, and for logging. Message Id is alphanumeric, maximum length is 250 characters.

Logical Id
Logical Id provides a name for the instance of an application connected to ION. As the sender and or receiver of all messages, the application instance is identified by its Logical Id.

Applications must ensure that the Logical Id used in the message matches the Logical Id defined in ION.

Logical Ids are also used to drill back to an application in an Infor Ming.le environment.

The Logical Id has format: infor.[application or connection type].[Instance name in ION]. The ‘application or connection type’ can refer to Infor applications such as ‘syteline’, ‘eam’ or ’ln’, or refer to technology connectors, such as ‘file’, ‘ws’ or ‘listener’. The 'instance name' is derived from the connection point name or message listener name. Logical Ids are alphanumeric, maximum length is 250 characters.

The only characters permitted in a Logical Id are the lowercase letters a-z, the dot(.), the digits 0-9, an underscore (_) and a dash (-).

Documentation Identification

Here is explained how to identify documentation.

Document ID
The ID of the document, is also available in the BOD, for example: SyncSalesOrder/DataArea/SalesOrder/Header/DocumentID/ID. As the unique identifier of an object Document ID is made up of several elements: the tenant, accounting entity, location, ID, and RevisionID.

Document ID is the simple ID for this object in its context, for example respective accounting entity, tenant etc. When in most cases, it is included in the BOD, Document Id (possibly along with Revision Id) is referenced in identifying the document in order to track the document in ION OneView, trigger event monitors, activation policies etc.

Document ID is alphanumeric, maximum length is 100 characters.

Revision ID
Revision ID sets the value that is used to keep multiple versions of the same document unique. An example of Revision ID: SyncSalesOrder/DataArea/SalesOrder/Header/DocumentID/RevisionID. Revision ID is alphanumeric, maximum length is 22 characters.

Variation ID
The variation ID of the document, when a Sync BOD is sent out from a system, a variationID is required. This variationID can be used at the receiving end to discover messages that are received out of sequence. For example: SyncSalesOrder/DataArea/SalesOrder/Head- er/DocumentID/ID/@variationID. Variation ID is numeric, the maximum value is 9223372036854775807.

Accounting Entity
Accounting Entity usually represents a legal or business entity, which owns its general ledger. Every single transaction only belongs to an Accounting Entity. Accounting Entity can also be defined as the owner of certain master data among the enterprise. Accounting Entity is alphanumeric, maximum length is 22 characters.

Location
A location is a physical place that is associated to a transaction. A location is owned by a single accounting entity, and may be used by multiple accounting entities. Location is alphanumeric, maximum length is 22 characters.

Message headers

Some header fields are identified as optional and some are mandatory in message headers.

Mandatory fields

Applications must guarantee the required fields are filled with correct values and format. Otherwise the message cannot be delivered or processed by consuming applications.

These header fields are required:

TenantID
The Tenant Id identifies the message as belonging to a specific tenant.

See the explanation in General concepts on page 13

MessageID
The Message Id is the unique identifier required for each message. See the explanation in General concepts.

BODType
The BOD Type header is used in ION Service to determine the verb and noun contained in the message. The verb and noun are used in routing algorithms. They are also used by client applications to filter messages from the inbound message queue. This header is case-sensitive and must follow the case of the verb and noun as used in the OAGIS schema. In the BOD Type, verb and noun are connected using a dot, that is: [verb].[noun]. For example, `Process.PurchaseOrder` or `Sync.Shipment`. We recommend that you use a noun name with maximum length of 30 characters. The verb does not exceed 11 characters. Acknowledge is the longest supported verb name. The maximum length of a BOD Type is 100 characters.

FromLogicalID
The Logical Id of the sending application. This Id is used to identify an application. Every application must have a unique identifier so that ION can direct documents to that application.

An example of a logical Id is:

```
lid://infor.eam.myeaminstance
```

For maximum length and other details, see the explanation on logical ID in General concepts on page 13.

Note: In case of network (tenant-tenant) connection.

When messages are transferred to a different tenant, the original source `FromLogicalID` is replaced with the logicalID of Network connection point on the target side.

ToLogicalID
The Logical Id of the destination application.

- Explicit routing:

  ION Service routes the message according to the Logical Id provided. This Logical Id must be valid for the specified routing rules.

  You must use explicit routing for BODs that have an Acknowledge or Show verb.

  Note: In case of network (tenant-tenant) connection.

When messages are transferred to a different tenant, the original `ToLogicalID` is replaced with the 'default' value on the target side.

- Implicit routing:

  The sending application is not aware who is interested in its message. ION Service routes the message according to specified routing rules. Therefore, the `ToLogicalId` header must be set to `lid://default`

  You must use implicit routing for BODs that have a Sync, Process, Load, Post, Update, Get, or Confirm verb.

For maximum length and other details, see the explanation on logical ID in General concepts on page 13.

Optional fields

The sending application must leave the optional header fields out or ensure they are filled correctly, consistent with the BOD XML contents.

In ION the header fields are not filled if they are missing or containing blank fields; but the data is transported. The receiving application can use the values if they are available, otherwise it can fall back to the BOD XML contents. The optional header fields are not used in all connectors. For example, the values of the header fields are not passed on to the stored procedure of a database connection point.

The optional header fields values cannot be null. Using `0` or a blank string `""` is allowed. When using an Oracle database for in-box and outbox, blank strings are treated as null. Consequently:

- When your application tries to write an optional header field with a blank string value to the outbox in Oracle, the insert action fails.
- When delivering a message to an in-box in Oracle, optional header fields with a blank string value are skipped in ION. This is to avoid failure of the message delivery.

Note: The data for the header fields is usually also available inside the BOD message. The sender is responsible for the consistency between the data in the header and the data inside the BOD. Header fields as set by the sender are not corrected in ION.

This list shows the header fields to identify the document that is included in the BOD.

- `AccountingEntity`
- `Location`
- `DocumentId`
- `RevisionId`
- `VariationId`

For the definition of each of these fields, see Documentation Identification on page 14.

Various header fields are available to support batch processing for large documents. In case of batch processing, the data can be sent in multiple BODs. The batch information is included inside the BOD, in the BODID element. Including it in the header can help the receiving application to handle the messages belonging to the same batch without opening the BOD messages.

Database connection point AnySQL type is populating batch headers.

This list shows the header fields to support batch processing for large documents:

- `BatchId`
- `BatchSequence`
- `BatchSize`
- `BatchRevision`
- `BatchAbortIndicator`

For details on these fields, see Sending messages in batch on page 44.

To further describe the message content, you can use these additional header fields:

- Instances

  Instances header is used to store count of object instances in the message. For example:

  - For a streaming JSON message, it indicates the number of instances in the stream.
  - For a Show BOD that includes multiple instances, it indicates the number of instances in the DataArea.
- Source

  Source header can be used to preserve information about the message source. In case read file action of File Connector is used, the source header is automatically populated with the source file name including file extension.
- Custom

  Custom headers can be used to preserve any other type of information that is not covered by other headers. You can use up to three custom headers per document. Each header must have the `Custom_` prefix.

Deprecated field

'Encoding' is an optional header field which is deprecated.

Details of Encoding see Documents encoding on page 26.

Chapter 3: Verbs and Verb Patterns

Standards are applicable to send your documents through ION and to enable all ION functionality for your document.

Functionality such as content-based routing and event monitoring. These standards are relevant for all documents, including custom documents.

Verbs

An important concept related to the BODs is the system of record. This is the application instance that owns a business object. A system of record can own all instances of a noun or it can own part of the instances.

Being the system of record or not determines the verbs to be used:

- Sync is sent from the system of record.
- Process, Get, Load or Update are requests that are sent to the system of record. For Process, the system of record will send an Acknowledge in reply. For Get, the system of record will send Show in reply.
- Confirm is a special type of verb. It is used for noun 'BOD', and it is sent when an error occurs in processing inbound BOD.

This table shows which verb to use for a specific goal:

Goal	Verb(s) to use
Publish changes on data owned by my application	Sync
Request changes on data owned by another application	Process/Acknowledge
Retrieve data owned by another application	Get/Show
Initial load, recovery	Get/Show (*)
Report an exception	Confirm

(*) This is the preferred approach. In theory you can use Show without Get, but then the sender must know the address (logical ID) of the receiver. An alternative is to use Sync. Especially when adding a new system of record. But when using this verb, event monitors or activation policies can be triggered.

This is what you do not want if Sync messages were already published before for the same data set. Because the messaging is asynchronous, the application that sends the process must have a way of handling the pending state until it receives the Acknowledge. For example, when requesting creation of a new item, the requesting application cannot use the item as if it were there already. In the meantime the application can use a specific status for the item such as ‘Pending’. Do not use other verbs. In addition to the listed verbs, ION supports using the Load and Update verbs. These are meant for integrations where data is loaded into an application where the application must avoid refusing the data. For example in an EDI scenario.

Action codes

The verb only indicates the action at a high level. It does not indicate whether a document is created, updated, or deleted. For such details, an action code can be included in the BOD. The action code is an attribute that is part of the verb section.

This table shows action codes for request verbs (except Get):

Action Code	Description
Add	A new business object instance is created.
Replace	A business object is changed and the complete business object is available in the message.
Change	A business object is changed and the message only contains the document ID and the changed properties. It is not recommended to use this action code.
Delete	A business object is deleted.

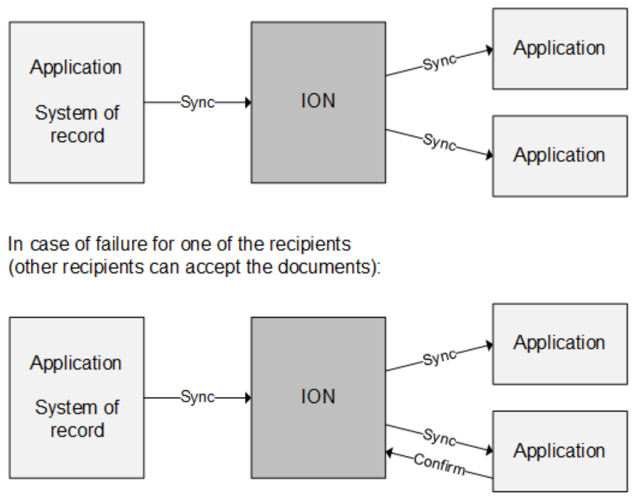
This table shows action codes for the Acknowledge response verb:

Action Code	Description
Accepted	The business object was created, changed or deleted in accordance with the Process request.
Modified	The business object was created or changed, but the resulting business object differs partly from the requested one.
Rejected	The creation, change or deletion was not done.

Note: For Sync the action code is the view of the system of record. That may not match the status of the receiving application. For example, SyncMyDocument with action code ‘Add’ and status ‘Draft’ is sent by application A. But it is not delivered to application B, because of a filter in the document flow. When application A sends SyncMyDocument with action code ‘Replace’ and status ‘Approved, then this will be the first document sent to application B. Application B has to add data, even though the action code is ‘Replace’.

Sync verb

Sync is sent from the system of record to anyone who is interested (‘broadcast’). It indicates that a business object was created, changed or deleted. The action code can be Add, Replace or Delete. Do not use action code Change. Use Delete only for tenant-level master data.



Publishing a Sync BOD

If you are the owner of a piece of business object data that can be represented in a BOD document. You can send Sync BODs to inform other parties about the current status. It is important to find the right balance when publishing Sync BODs. You do not want to publish too many BODs and not all changes to a document are relevant for the outside world. However, publishing less BODs can limit a customer when creating an integration or when using event management or workflow.

Take this into account:

1. When is my business object complete?

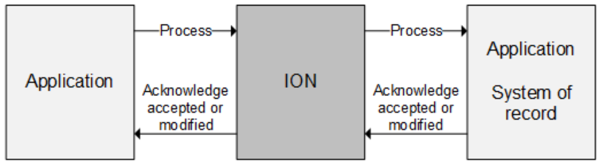
   In case of order entry, when the order header is saved but the lines are not added yet. It is not useful to publish a sync BOD for the order document. On the other hand, don’t be too late. For example, you can say “I will not publish this document until it has status ‘Approved’, because it is still being updated”. But this means that an event monitor cannot generate an alert until the document is Approved, which can be too late for corrective action.
2. Which status changes are relevant?

   It is hard to give general guidelines. What is important for one business document may not be important for another document. Changing a description or adding a note may not be relevant outside the context of your application. Changing the status or changing important data such as amounts or dates will usually be relevant.
3. When is my business object at the end of its lifecycle?

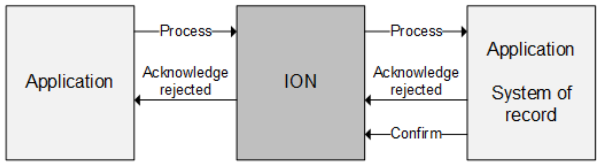
Usually data is not deleted but archived when a business object is no longer current. This is important to sync out. It will inform other applications to not use the business object anymore.

Process and Acknowledge verbs

Process is a request sent to the system of record to add, change or delete a business object. Action codes for Process: Add, Change, Delete. Acknowledge is the reply sent back to the requestor. Action codes for Acknowledge: Accepted, Modified, Rejected.


Application
Process
Acknowledge accepted or modified
ION
Process
Acknowledge accepted or modified
Application
System of record

In case of failure:



Because the messaging is asynchronous, the application that sends the process must have some way of handling the pending state until it receives the Acknowledge. For example, when requesting creation of a new item, the requesting application cannot use the item as if it were there already. In the meantime the application can use a specific status for the item such as ‘Pending’.

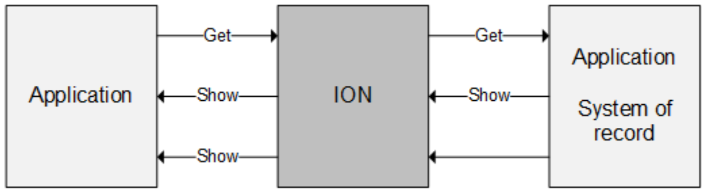
When sending a `Process` BOD with an `Add` action code, the system requests a new record to be created in the Systems of Record (SOR), at this point the unique ID of the object is yet to be created in the SOR. Hence no Document Id must be populated in the `Process` BOD, but is populated in `Acknowledge` BOD when the record is successfully created in the SOR.

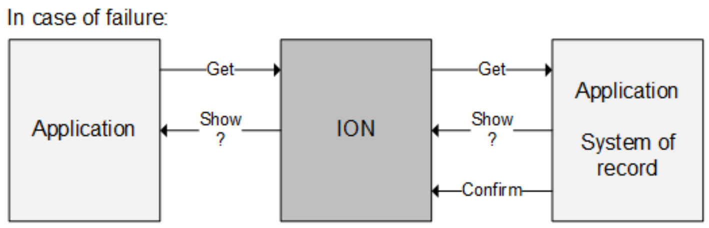
When the `Process` BOD gets action code `Change` the sending system requests a record to be either modified or deleted or to kick off a process related to the data object sent through the BOD. In this case Document Id, and Variation Id must still be populated to identify corresponding record in SOR.

Infor does not use action code `Delete` in Process BODs.

Get and Show verbs

A Get request is sent to the system of record to retrieve one or more business objects. A ‘query language’ kind is used to select the business object(s). The Show is the reply to a Get request. In Infor, the Show is the only BOD that can contain multiple nouns instances. A Show can also be sent without a Get request. This can be used to push data for initial load or recovery. We do not recommend this, because it requires that the sender of the Show must know the applications that are interested in this message. No action codes are used in Get and Show.





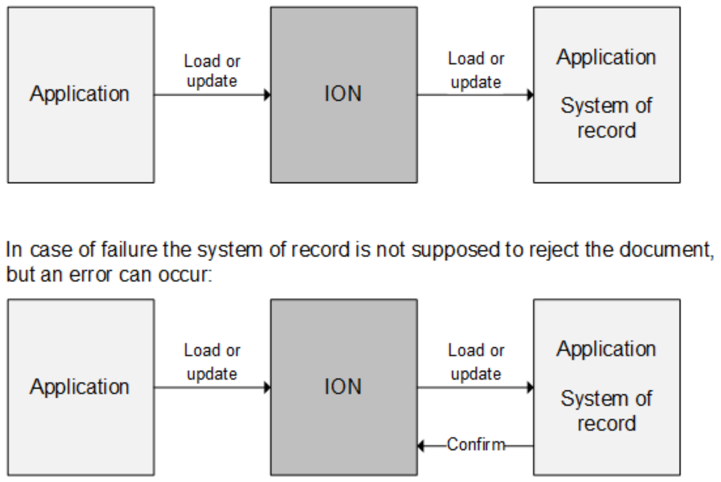
The Get verb is used to request data from a system of record (SOR). Either for:

- The purposes of an initial load or reload.
- To get specific instances of a document that the sending system typically does not receive.

Include the Document Id if the Get BOD requests a specific object from the receiving system. Otherwise a selection criteria through expression element is used.

Load and Update verbs

Load and Update are used in special cases. The loaded documents cannot be refused by the system of record. These verbs are typically used for EDI. The Load verb is used primarily when a document is created by a trading partner and then passed into the SOR for the noun. Therefore in similar Process, there is no Document Id to specify in the BOD. Load is sent to the system of record to add an object. The action code will always be Add. Update is sent to the system of record to notify of a change. The action code will always be Replace.


In case of failure the system of record is not supposed to reject the document, but an error can occur:

Confirm BOD

A ConfirmBOD is sent by an application when processing any inbound BOD and something has gone wrong. A ConfirmBOD is always handled inside the ION Service, apart from ION it can never be sent to another application.

When to send Confirm BOD

When an exception that results in an inbound BOD not being processed occurs. The application must abort the transaction and send a ConfirmBOD.

Sending Confirm BOD from Process BOD

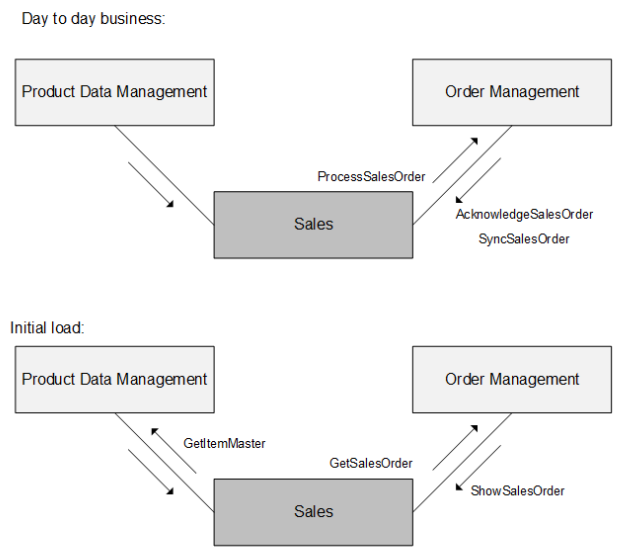
When sending a ConfirmBOD for a Process BOD, also send an Acknowledge with action code ‘Rejected’. Include a ReasonCode to explain the problem. The sender must be informed through the Acknowledge. In an Acknowledge BOD you must include the original ApplicationArea from the corresponding Process BOD. If you cannot get the original ApplicationArea, you can send a ConfirmBOD without an Acknowledge ‘Rejected’ BOD.

Example of Use verbs

Let us assume that three applications exist:

- Product Data Management: maintain product data (system of record for the ItemMaster noun).
- Order Management: handle sales orders (system of record for the SalesOrder noun).
- Sales: view the products and place orders

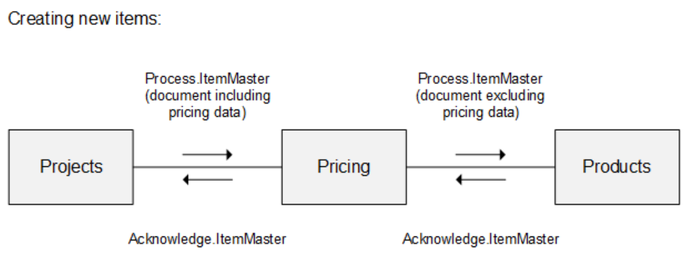
In this case, these BODs can be used to integrate these applications:

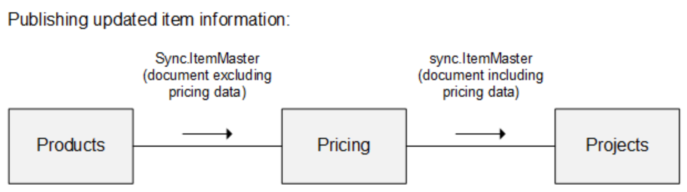


Fragmented data

Data for a single business object can be stored in multiple applications. But it is strongly advised that at any point in time one application is the owner of a specific piece of information.

If parts of the business object are owned by multiple applications, an enrichment or pipeline pattern can be used. For example, an ItemMaster business object is partly owned by a Product application and partly by a Pricing application. It is assumed that a third application exists called Projects. This application requests items to be created (using Process BODs) and must receive messages if an item is changed (using Sync BODs). See this diagram for the flows you can use:


Creating new items:
Process.ItemMaster (document including pricing data)
Process.ItemMaster (document excluding pricing data)
Projects
Pricing
Products
Acknowledge.ItemMaster
Acknowledge.ItemMaster


Publishing updated item information:
Sync.ItemMaster (document excluding pricing data)
sync.ItemMaster (document including pricing data)
Products
Pricing
Projects

For more information about verb definition and context to use, see the Infor Messaging Standards in Message contents on page 26.

Network connection

Reply verb patterns, such as Process and Acknowledge, Get and Show, Load and Update, have some limitations when the source and target applications are in different tenants.

In this scenario, the target application does not receive the original source `FromLogicalID` in the message header and is therefore not able to set `ToLogicalID` directly to the source application. The reply message is delivered to the source tenant with `ToLogicalID` 'default'. Then the automatic fallback mechanism ensures that the message is delivered to the source application. If there are more applications producing the same request message in the source tenant, each reply message is delivered to all of them.

Chapter 4: Message contents

Miscellaneous Infor messaging standards are supported by ION which applications must follow.

Noun references

Many nouns are referenced inside other nouns, both master data and transactional nouns. Referenced information can be found inside a BOD in these ways:

- Specific references used for transactional data, end with the word `Reference` and have the noun name at the beginning.
- The generic reference of transactional data through the `DocumentReference` element that uses the `type` attribute which contains the noun name of the referenced object.
- Used with master data which uses the name of the object as the reference and includes additional information with the reference information.

Documents encoding

An Infor standard is that all XML documents use the UTF-8 encoding. Therefore, there is no need to set this key. The default is UTF-8.

All readers and writers to the database tables that are used by the Infor Application Connector must use the exact same encoding mechanisms. When writing a message to the `COR_OUTBOX_ENTRY` table, serialize your XML string to the C_XML column as a UTF-8 byte stream. Similarly, when reading the `COR_INBOX_ENTRY` table. You extract your XML string from the C_XML column by converting the bytes as a UTF-8 byte stream.

If you do not use the UTF-8 decoding when reading the XML from the table, and the XML contains characters that require two bytes, the XML can be invalid. Also the formatting is affected. A worst case scenario is that you cannot parse the XML document.

Date and time

When working on ION integrated date and time format fields, follow these guidelines:

- Dates and times in BODs are in UTC time.
- They must be represented in a common format, using the capital letter Z at the end of the value. For example: 2009-08-13T15:30Z.

Chapter 5: Connecting to ION

The preferred approach to connect to ION is using the Infor Application Connector.

The application will be truly event-driven. BODs are used as the standard interface. Using BODs in combination with the inbox/outbox avoids unwanted dependencies and makes the integration robust. In ION, hardly any configuration is required to connect your application.

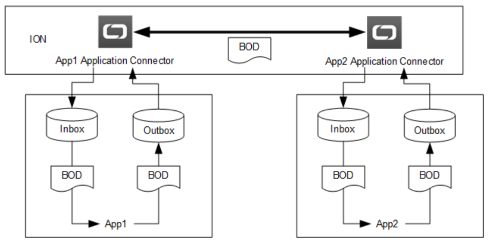
On the other hand, you must change or extend the application to enable the application to publish and receive BODs. Consequently this is less suitable for customer-specific integration of legacy applications.

Alternatives in that case are third-party connectors such as Database Connector, Web Service Connector, or Message Queue (JMS) Connector.

Infor Application Connector

The Infor Application Connector makes use of inbox/outbox tables in your applications. ION will post messages into your inbox and will pick up messages from your outbox.

This diagram shows an example of two applications exchanging messages using inbox and outbox tables:



To use the Infor Application Connector:

1. Create the inbox/outbox tables. Inbox and outbox tables must be created inside your application. The tables will be accessed by ION through JDBC. SQL scripts are available from ION Desk.
2. Implement the sending and receiving of the BODs. When a BOD must be published, create the BOD XML and determine the correct values for the header fields. Then insert the data into the Outbox tables. When a BOD is received in the Inbox tables, pick up the BOD XML and based on the content insert or update you application data.

For details, see Using the Infor Application Connector on page 36.

Using third-party connectors

Alternative connectors

Some alternatives for the Infor Application Connector exist, because ION provides a set of third-party connectors.

The most important third-party connectors are:

- Message Queue (JMS) Connector. Messages are sent and received through message queues using the JMS standard.
- Web Service Connector. ION can invoke existing web services to send or retrieve data.
- The Database Connector. Data is read from or written to the database directly through stored procedures.
- File Connector. Files are used to send or receive messages.

For details on these connectors, see the *Infor ION Desk User Guide* .

Advantages and disadvantages of each connector

Using the Infor Application Connector is preferred, because it is decoupled and event-driven, and the application can validate the correctness and consistency of incoming data. From a modeling and management perspective the Infor Application Connector also is the best choice, because the modeling is very simple and the management in ION Desk is the richest for this connector.

If you cannot use the Infor Application Connector, you can use the following sequence as a rule of thumb to select one of the other connectors:

1. Message Queue Connector
2. Web Service Connector
3. File Connector
4. Database Connector

This table shows advantages and disadvantages of the individual third-party connectors:

Connector	Advantages	Disadvantages
Message Queue (JMS)	Event-driven solution, so it closely matches the ION architecture.	The message queue only provides a technical communication channel. Behind the message queue the messages must be created or processed. If the message is not a BOD, it must be transformed.

Connector	Advantages	Disadvantages
Web Service	No change required in the application, existing web services can be used.	In the model you must transform the messages from BOD to web service input and from web service output to BOD. Existing web services often do not offer operations to retrieve only the changed objects. Transactionally less robust than database or message queue connector, so higher risk or delivering the same message twice in case of interruptions or timeouts.
File	Suitable for legacy applications that only support a file-based integration. Event-driven.	The application at the other side must be in a position to create files in a format that ION can process or read files as provided by ION. The data must be transformed to a BOD (XML) message. Multilevel data (header and lines) can be formatted in multiple ways in one or more files, while the File Connector does not support all options. Transactionally less robust than database or message queue connector, so higher risk or delivering the same message twice in case of interruptions or timeouts.

Connector	Advantages	Disadvantages
Database	No change required in the application (except the addition of stored procedures in the database schema).	Risk for data integrity, because you bypass the application that owns the database, while updating the database directly. You must write stored procedures to query the database and transform the result to XML or to retrieve the data from the incoming XML and update the database . These procedures can become complex when handling multi-level objects (such as headers and lines). Additionally these procedures will be database-dependent. Not event driven. The database must offer data to retrieve the new or changed data, such as log tables. Timestamps can be used for new data, but will not expose all intermediate changes that occurred between two scheduled read actions.

Chapter 6: Infor Application Connector (IMS)

This section explains the adoption procedures that applications must fulfill to integrate with ION through the ION Messaging Service, (IMS), connection point.

IMS is a connector that allows applications to integrate with ION through REST/JSON APIs. Unlike the IO Box connector, IMS does not require direct access to an application's database. Instead, IMS communicates through the secured https protocol, through OAuth 1.0 for authentication. Therefore, IMS is a loosely coupled connector that makes integrations easier.

IMS specifications include well defined API methods. These methods are implemented by ION and must be implemented by the concerned application. After this, they can push messages to each other through the APIs.

IMS can send and receive multiple message requests in parallel. Therefore, sequence of message transport is not guaranteed when using the IMS connector. If sequencing is important, you must use, for example, a `VariationID`.

The current IMS protocol supports two versions: v1 and v2. The v2 protocol offers the same functionality as the v1 protocol and more. Therefore, new adopters should use the v2 protocol. The v1 protocol is not described in this document.

Message or multipartMessage

In the v2 protocol two variants of message sending are supported:

- The `message` method

  This method is still supported to be compatible with v1. The `message` method has a disadvantage: you must include the payload, that is, the actual message content, as a property value in a json message. This often requires base64 encoding to ensure you remain json compliant.
- The `multipartMessage` method

  This method has less overhead in processing the messages, especially when larger messages are involved.

Note: We strongly recommend that you use the `multipartMessage` method.

Encoding

For transport performance reasons, especially where it involves larger messages, you can compress the message before sending. In that case you can use the DEFLATE or GZIP encoding. ION ensures the message is decompressed before it is delivered to the receiving application. If you use the `message` method, you must encode the compressed message in base64 format. For the `multipartMessage` method, this is not required.

IMS interaction

Application sends a message to ION

Configuration

Ensure you know in what format you expect to exchange information with ION.

- messageMethod = "message" or "multipartMessage".
- supportedEncoding:
  - message: NONE, GZIP64, DEFLATE64, or BASE64
  - multipartMessage: NONE, GZIP, or DEFLATE
- supportedCharacterSet : UTF-8
- Know your logical-id.
- IMS version is set to v2.

Prepare for sending messages

These methods are used by your application to get information and verify whether message processing is enabled:

- `GET <Application>/service/ping`

  Checks whether ION can be reached and you have the correct OAuth 1.0a keypair.

  Note: Although the ION ping still returns a body, ignore that body. That body response is deprecated and isl no longer sent when API v1 is no longer supported.
- `GET <ION>/service/versions`

  Checks whether ION supports this version (v2).

Send a message

This is the proposed sequence when your application sends a message to ION:

1 `GET <ION>/service/ping`

If this is successful, continue. Otherwise retry and raise errors.

For the retry, we recommend that you increase the delay between retries if the problem exists longer. To prevent log flooding, consider aggregating the ERROR message.

2 `GET <ION>/service/versions`

Checks whether ION supports this version (v2). Halt if this fails; you must upgrade ION to a version that supports v2.

3 `POST <ION>/service/v2/message or POST <ION>/service/v2/multipartMessage Http header X-TenantId`

If sending the message fails, perform one of these actions, based on the error code:

- Decide to retry, that is, return to step 1.
- Raise an ERROR for the specific message and continue with the next message.

If the message was sent successfully, continue with the next message.

ION sends a message to an application

Configuration

Ensure an Infor Application (IMS) connection point is configured in ION. This connection point must be used in a document flow. Ensure this document flow is activated.

Prepare for sending messages

Ensure your application exposes these methods, which are used by ION to get information and verify whether message processing is enabled:

- `GET <Application>/service/ping`

  Checks whether ION can connect to your application and checks whether ION is authorized for your application.
- `GET <Application>/service/protocol`

  Called by ION to retrieve, from the application, the expected IMS API version and protocol parameters.
- `POST <Application>/service/v2/discovery Http header X-TenantId`

  Called during modeling of the connection point in ION Desk. Used to retrieve, from the application, the documents that are supported to be exchanged.

Send a message

When a message is being sent, ION executes these methods in the given sequence:

1 `GET <Application>/service/ping`

If this is successful continue, otherwise retry and raise errors.

2 `GET <Application>/service/protocol`

With this call ION retrieves the IMS protocol parameters from the application.

These protocol parameters dictate how ION sends the message in the next step.

3 `POST <Application>/service/v2/multipartMessage or POST <Application>/service/v2/multipartMessage Http header X-TenantId`

If sending the message fails, ION performs one of these actions:

- Generates a Confirm BOD and continues with the next message.
- Retries from step one onward for the same message.

If the message was sent successfully, ION continues with the next message.

API specifications

For details on these methods, see the swagger documentation for IMS.

This documentation is exposed by your ION installation through this URL:

```
https://<ION host>:<ION router port>/api/ion/messaging/service/swagger/v2/
ui
```

For a typical installation the port is 9543.

This is the https port of the ION Grid XI Platform router.

Chapter 7: Using the Infor Application Connector

This section explains the adoption procedures that applications must fulfill to integrate to ION using the Infor Application Connector.

You must have some specific knowledge about these topics:

- Application connection points
- In-box and outbox tables
- Removing messages from the in-box and outbox tables

Application connection points

To connect an application to ION, the application owner must define which BODs that application can send and receive. To define the BODs for an application, you must create an application connection point. You can export connection points to an XML file. When you export the connection point without properties, you can use it as a template for the application.

See the *Infor ION Desk User Guide* on how to create and export connection points.

Inbox and outbox tables

All applications must add some of these new tables to their existing database so that the application can read and write the tables in the same transaction as their business logic:

- COR_OUTBOX_ENTRY
- COR_OUTBOX_HEADERS
- COR_INBOX_ENTRY
- COR_INBOX_HEADERS
- ESB_INBOUND_DUPLICATE

Caution: When an application adds a new message to the COR_OUTBOX_ENTRY and COR_OUTBOX_HEADER tables, the inserts must be performed within the same transaction. Inserts that are not performed within the same transaction cause data corruption.

To download the scripts to create those tables:

1. Start ION Desk.
2. Select **Configure > ION Service**.

   The **Configure ION Service** page is displayed.
3. Click the **Configuration Files** tab and then click **Download Scripts to create I/O Box**.
4. Specify a file name and click **Save**. A zip file is downloaded.
5. Extract the zip file and use the db vendor-specific sql files that are applicable to you. For Unicode-compatible BOD header fields, use the scripts available with the _unicode suffix.

Note: The script might also generate a COR_PROPERTY table. This table is included for future use, so you can safely ignore it.

COR_OUTBOX_ENTRY

This table shows the COR_OUTBOX_ENTRY table API:

COR_OUTBOX_ENTRY	Description
C_ID	The row's primary key - all of the provided database schemas have this set as auto-increment.
C_XML	The message that you are sending. The message must be encoded as described below.
C_TENANT_ID	The Tenant Id identifies the message as belonging to a specific tenant. A tenant is a hosting or software as a service (SaaS) concept where all the data for one tenant is always separated from all the data of other tenants. There is no cross-sharing or viewing of data with other tenants. This concept requires all the participants in the messaging to share the same identity for the same tenant. Therefore, a Tenant ID of " infor " must have exactly the same meaning on every system in the messaging space.
C_LOGICAL_ID	This field is added since ION 11.1.2. It is added by running the scripts present in the '3.0' folder in your inbox/outbox. If present, then this field must contain the value of the 'from logical id' of the application that publishes the BOD and it must be populated with the ‘lid://’ prefix.
C_MESSAGE_PRIORITY	Messages with a higher priority are sent before messages with a lower priority. You can set the priority from 0 to 9, 9 being the highest priority. High priority messages should be limited - most messages should be set at 4.

COR_OUTBOX_ENTRY	Description
C_CREATED_DATE_TIME	The date and time the message was inserted into the outbox table. You must specify the time in the variable in UTC format.
C_WAS_PROCESSED	Users should never provide a value for this column. The column is used by ION Service to determine whether a message has been sent. Unprocessed messages are marked as 0; processed messages are marked as 1.

ION Service removes all the processed messages older than the number of hours specified in the cleanup advanced properties.

COR_OUTBOX_HEADERS

This table shows the COR_OUTBOX_HEADERS table API:

COR_OUTBOX_HEADERS	Description
C_ID	The row's primary key - all of the provided database schemas have this set as auto-increment.
C_OUTBOX_ID	Used to join the headers to the message's COR_OUTBOX_ENTRY row. As such, this value should be the same as the message's COR_OUTBOX_ENTRY C_ID column.
C_HEADER_KEY	The key used to describe the type of header. For valid keys, see "Message headers".
C_HEADER_VALUE	The header's value.

If any of the required headers are not provided, ION Service creates a Confirm BOD for that message.

For the headers to be used, see Message headers on page 15.

COR_INBOX_ENTRY

This table shows the COR_INBOX_ENTRY table API:

COR_INBOX_ENTRY	Description
C_ID	The row's primary key - all of the provided database schemas have this set as auto-increment.

COR_INBOX_ENTRY	Description
C_XML	The message you are receiving. The message must be encoded as described.
C_TENANT_ID	The Tenant ID identifies the message as belonging to a specific tenant. A tenant is a hosting or software as a service (SaaS) concept where all the data of one tenant is always separated from all the data of other tenants. There is no cross-sharing or viewing of data with other tenants. This concept requires all participants in the messaging to share the same identity for the same tenant. Therefore, a Tenant ID of " infor " must have exactly the same meaning on every system in the messaging space.
C_LOGICAL_ID	This field is added since ION 11.1.2. It is added by running the scripts present in the '3.0' folder in your inbox/outbox. If present, then this field contains the 'To logical id' value of the application to which the BOD is delivered.
C_MESSAGE_PRIORITY	Message priority as provided by the application that sent the message.
C_CREATED_DATE_TIME	The date and time the message was inserted into the inbox table. Date and time are in UTC format.
C_WAS_PROCESSED	ION Service always sets this to 0. It is the application's responsibility to remove processed messages.

COR_INBOX_HEADERS

This table shows the COR_ INBOX _HEADERS table API:

COR_ INBOX _HEADERS	Description
C_ID	The row's primary key - all of the provided database schemas have this set as auto-increment.
C_ INBOX _ID	Used to join the headers to the message's COR_ INBOX _ENTRY row. As such, this value should be the same as the message's COR_ INBOX _ENTRY C_ID column.
C_HEADER_KEY	The key used to describe the type of header. For valid keys, see "Message headers".

COR_ INBOX _HEADERS	Description
C_HEADER_VALUE	The header's value.

Incoming messages are placed in the COR_INBOX_ENTRY and COR_INBOX_HEADERS tables. All rows are inserted in the same transaction.

ESB_INBOUND_DUPLICATE

This table is used by ION to maintain the unique Message IDs. ION uses this table to reject duplicate messages in the ION Service. Applications should not use this table.

Removing messages from the inbox and outbox tables

ION Service removes messages from the COR_OUTBOX_ENTRY and COR_OUTBOX_HEADERS tables. Removing these messages is achieved in these ways:

- The message is deleted after it is successfully sent by ION Service.
- The message is deleted if it has been successfully sent and is older than XX hours. ION Service checks for expired messages when it is started, and then checks every hour.

By default, the second option is used. By not deleting the messages immediately, the Manage tab in ION Desk monitors the COR_OUTBOX_ENTRY table and reports the number of processed and unprocessed messages. To change this behavior, set the application polling property within ION Desk:

```
Delete Processed Messages=true
```

Therefore, ION Service deletes the message after it is sent.

Caution: ION Service does not remove messages from the COR_INBOX_ENTRY/COR_INBOX_HEADERS tables. The application must provide the code to clean up these tables. If the tables are not cleaned up, the file system of the database server can get full.

Polling Message Preference

When you implement ION integration you can decide to use one of these options:

- single inbox/outbox shared by multiple sites (represented by multiple Logical Ids)
- single inbox/outbox shared by multiple tenants in the Cloud.

A ‘Message processing preference in I/O box’ setting is available in ION Desk Connection Point Advanced Settings, to cater for two requirements.

Single I/O Box for Multi-tenant

In a multi-tenant environment a single set of Inbox and Outbox tables of the application can be shared by multiple tenant instances. In such situations an application connection point must process messages belonging to its own tenant.

To setup Single I/O Box for Multi-tenant:

1. Go to **Connection Point definitionConnectionAdvancedMessage processing preference in I/O box**.
2. Select the **by Tenant** option to be `true`.
3. When publishing a BOD from your application, specify this information:
   - The `C_TENANT_ID` value of the `COR_OUTBOX_ENTRY` table.
   - The Tenant Id key in the `COR_OUTBOX_HEADERS` table with the correct tenant value. It must match with the Tenant value specified in the connection point.

Ensure you do not have more than one connection point from the same tenant sharing the same Inbox and outbox tables.

The Tenant value is matched in ION in a case-sensitive manner. If the tenant value is blank in the connection point, the default tenant value of 'INFOR' is assigned. To avoid inconsistencies in tenant processing, define the Tenant Id according to the Infor standards.

Duplicate detection of messages is not enforced by ION. Under uncommon circumstances such as an incomplete message processing, this results in delivering the same message twice to an inbox. The application must be prepared to receive duplicate messages.

Single I/O Box for Multi-Logical Ids

This addresses requirements from applications which have multiple sites where an individual connection point is defined per site in ION. These applications use the same Inbox and Outbox table between them. In such situations, messages based on Tenant and based on Logical Id must be processed.

To setup Single I/O Box for Multi-Logical Ids:

1. Upgrade the I/O Box to version 3.0. Go to the 3.0 folder in your I/O box and run the I/O box script `<db>_upgrade.sql`.

   The downloaded zip file contains these folders: 1.0 and 3.0. The folder 1.0 contains the scripts to create a base I/O box for the standard databases. This includes: MS SQL server, Oracle, DB2, DB2400 and MySQL.

   The 3.0 folder contains the scripts required to prepare your I/O box to support multiple logical Ids sharing the same I/O box. Run the scripts in the 3.0 folder to create I/O Box tables in your application. For Unicode compatible BOD header fields, you can use the scripts available in the Unicode folder.

2 Specify the message processing preference to be **by logical Id** in each connection point defined for your application.

Go to **Connection Point definition > Connection > Advanced > Message processing preference in I/O box**.

Select **by Logical ID** to be true and select **by Tenant** to be true.

Before you proceed, check if the column called `C_LOGICAL_ID` exists in both `COR_OUTBOX_ENTRY` and `COR_INBOX_ENTRY`.

3 When you publish BODs from your application, ensure that the correct values are specified in these columns:

- `COR_OUTBOX_ENTRY` table, specify `C_LOGICAL_ID` with your actual Logical Id value.
- `COR_OUTBOX_HEADERS` table, specify the `FromLogicalId` key with the correct Logical Id value.
- Both Logical Ids must match the Logical Id value specified in the connection point.

Note: The Tenant and the Logical Id value are matched in a case-sensitive manner. To avoid inconsistency in tenant processing, specify the Tenant Id and the Logical Id according to the Infor standards.

When these properties are not selected, you must ensure that each Infor application connection point uses its own Inbox and Outbox tables. This will be guaranteed by using different URLs. If you use the same URL in multiple connection points, then use different users. Ensure the users are linked to different database schemas in the database management system.

Chapter 8: ION Connecting Considerations

There are some considerations for applications to take into account when implementing integration through ION.

Handling transactions

Ensure the publishing and processing of the BODs is done in such a way that no data is lost. Publish the BODs inside the application transaction that inserts or changes the corresponding business data. Or, if publishing is done offline, have another mechanism in place that ensures no BODs are lost.

Because ION is event driven you must consider how to publish historical data before you enable ION integration with your system. Data consistency must be honored. When handling an incoming BOD, set the status in the Inbox to ‘1’ (processed) in the same transaction as the database updates that are done while processing the BOD.

Message sequence

Delivering messages in sequence is not guaranteed. Only delivery is guaranteed (at least once). In some cases the sequence of receiving is the same as the sequence of sending, but you cannot rely on this. Many factors can impact the sequence, such as:

- Parallel processing (multi-threading).
- Documents traffic.
- Intermediate steps in the process, such as content-based routing, filtering or mapping.

Take these situations into account:

1 Message Delivered with Delay or Out of Sequence sometimes due to network or BOD traffic BODs are not always delivered on time or in sequence. Especially in case they reference to each other. For example; a `Sync.ItemMaster` BOD reaches the application later than a `Sync.Purchase Order`. The application may well send out a Confirm BOD for the `Sync.PurchaseOder` BOD. Or is highly recommended to set a retry mechanism to have a few tempts based on intervals to sort out the message delay or out of sequence issue.

2 Message Delivered Out of Sequence for Different Verbs of the Same Object. It happens sometimes when an application awaits the Acknowledge BOD from the SOR for the object via Process BOD. The Sync BOD of the same object from the SOR arrives prior to any Acknowledge BOD. This is similar to point 1, caused by BOD traffic or network, and can be solved by a few times retry in turn. Therefore applications are recommended to include this into IONAdoption consideration.

3 Multiple updates from SOR on same document out of sequence Multiple changes on the same document may not arrive in the correct sequence. Use the Variation ID to check whether a Sync message is out of date. If the Variation ID of an incoming message is lower than a Variation ID you already processed for the same document type and document ID, then you must not overwrite the newer information you already have.

4 Data with interdependency break down into a set of documents. This is the same as what is explained in "Sending messages in batch".

Duplicated messages

When publishing a message in an Infor Application Outbox, the message ID from the header will be checked. If a message with the same message ID was processed successfully before, the new message is ignored.

At the end of a flow, in exceptional cases the same message can be delivered twice (guaranteed delivery ‘at least once’). The receiving party must ignore a message if a message with the same message ID was processed before.

For that reason, ensure to use a unique message ID when sending a message to your outbox. Otherwise the message will be ignored. The message ID must be globally unique, so a sequence number generated by your application is not sufficient.

Sending documents in batch

Avoid sending large documents. Documents up to 5 MB are handled throughout ION.

Do not include large files inside the documents, for example, images or PDF files. Instead, make the files available in a document management system or another location. Include a reference (URL) in the document.

Sometimes you must divide a large file into multiple documents. For example, the first document can contain a header and the first 100 lines. The next document can contain the next 100 lines. Each document must be valid.

Batch message headers are used to indicate that the document is part of a batch. Batch fields should not be sent for single documents (batch size = 1).

For BODs, Batch fields are used to indicate that the BOD is part of a batch. The batch fields must be included inside the BOD XML, in the BODID, and can be included in the message header. The message header fields are optional. We recommend that you include them. It allows the receiver to handle the batch without having to open each BOD to determine the correct sequence. Note that the total length for the BODID cannot exceed 255 characters.

The data types and maximum lengths of the header fields are specified in the table.

This table shows the fields to use:

Header Field Name	Name of element in BODID	Description
BatchId	`batchID`	The unique ID of the batch. The documents with this number must be processed sequentially on the receiving side. The **BatchId** is alphanumeric, maximum length is 250 characters.
BatchSequence	`batchSequence`	The sequence number of this document in the batch. This is required because the documents can arrive out of sequence. The BatchSequence is numeric, the maximum value is 9223372036854775807.
BatchSize	`batchSize`	The total number of documents in the batch. This can be unknown until the last document and omitted for the other documents. The **BatchSize** is numeric, the maximum value is 9223372036854775807.
BatchRevision	`batchRevision`	The revision number of the batch. This is used when one set of documents fails. The complete set can be resent using a new revision number. The revision number must be increasing, but does not have to be sequential. The **BatchRevision** is numeric, the maximum value is 9223372036854775807.
BatchAbortIndicator	`abortIndicator`	This indicator type attribute is set to true when a system that is sending a batch determines that it will not finish. The receiving system is notified that any documents received as a part of this batch must be discarded. The **BatchAbortIndicator** value is either ‘true’ or ‘false’.

For example, publishing daily balance updates through SourceSystemGLMovement BOD. Often you must split the BOD. It is assumed that the tenant is ‘acme’, the accounting entity is 10 and the location is 1. The subsequence BODIDs is one of these options:

- `Infor-nid:acme:10:1_A:0?SourceSystemGLMovement&verb=Sync&batchSequence=1&batchID=lid://infor.sunsystems.5:1`
- `Infor-nid:acme:10:1_A:0?SourceSystemGLMovement&verb=Sync&batchSequence=2&batchID=lid://infor.sunsystems.5:1`

- `Infor-nid:acme:10:1_A:0?SourceSystemGLMovement&verb=Sync&batchSequence=3&batchID=lid://infor.sunsystems.5:1&batchSize=3`

Publish historical data

Due to the synchronous nature of data in an event-oriented architecture, you can only keep data in sync after your application is integrated with ION.

You may need to consider how to share historical data with downstream systems that are interested to receive. This concerns the data which may not be amended in your system anymore. We recommend that you provide a facility to publish historical data when the application is ION-enabled. For example, driven by end users at time when required. This can happen more than once due to online offline deployment.

To keep the consistence of message ID and variation ID your system must publish BODs after historical data is published.

Message reprocessing

When BODs are delivered to your application inbox/outbox tables, it is the application's responsibility to consume or reject them due to circumstances. There are options to set intervals in ION Desk to clean up inbox/outbox tables. You can develop your own clean up scheme considering BODs volume and storage.

In case of processing inbound BODs fails for business data integrity, we recommend that you develop a user-driven retry mechanism. For example, workflow integration rather than mandating business users to submit the document for approval again. For technical reasons, it makes better sense if the system admin is able to resubmit the same BOD to get through. Same with consuming transaction BODs, ledger entry or purchase order quite often a retry is mandatory to sort out interdependency of BODs delivery.

When developing system automatic retry, it is important to retry at reasonable intervals and end the process if the problem is not solved. This is to avoid deadlock of your integration engine focusing on a few BODs and leaving the Inbound BODs queue too long to process.

Performance

Good performance is a key to success especially in an event driven integration architecture. Sensible plan and proof of concept testing with real business scenarios will eliminate substantial issues with customer implementation after releasing your ION integration.

Depending on the integration requirements we highly recommend that you take performance into account during the start and design phase.

Chapter 9: Adopting Event Management, Workflow, or Pulse

When the connection to ION is completed, you can adopt Event Management, Workflow or Pulse.

With Event Management you can generate alerts for business data that you publish. The alerts are created by an event monitor that checks the Sync BODs that are published against a user-defined rule.

Workflow enables you to execute workflows, either based on published business data or by explicitly starting a workflow. In Workflow you can implement decisions and bring tasks and notifications to users.

You can also create alerts, notifications, or tasks directly from your application. In that case you do not use Event Management or Workflow, but you directly request the Pulse engine to create an alert, notification, or task.

Alerts, notifications and tasks

A task is sent if the user is expected to execute a defined task. The user must complete the task in Infor Ming.le using the Tasks widget in the Homepages or in the Infor Ming.le Mobile application. To complete a task, the user specifies data or selects a specific action, such as 'Approve' or 'Reject', when closing the task.

An alert means a user must be notified of an exception. It indicates that something happened that is extraordinary or that is not in line with how the business should run. The user can decide whether to take action and then close the alert.

A notification is a message to one or more users 'for your information'.

Note: Some differences exist between the features that ION offers for tasks, alerts and notifications:

- The names (labels) of the data elements for an alert are not translatable. For tasks and notifications created from a workflow the labels are translatable.
- In alerts, drill-back links are automatically generated based on document references. For tasks and notifications that are created from a workflow, drill-back links can be configured. To configure these drill-back links, use drill-back views that are retrieved from the Infor Ming.le configuration.

When to use Pulse, Event Management and Workflow

Use Pulse if you want to create an alert, notification, or task directly. Your application is fully in control. The application logic decides whether and when to create an alert, notification, or task and also defines all aspects such as the data to be included and the user(s) who will receive the item. For tasks, your application also follows up if required when the task is completed.

Use Event Management if you need alerting, but you want to delegate the monitoring to ION. You do not have to change your application; the only requirement is that you publish Sync BODs for the business data owned in your application. You can define rules in ION event monitors. Customers using your application can adapt the rules to their needs or define new rules. ION monitors the BODs published from your application and creates alerts in Pulse when required.

Use Workflow if you want to model a business process (or allow your customers to model a business process) outside your application. Tasks and notifications are created automatically based on the modeled process. If required, the user interacts with your application based on the tasks that the user receives.

The following chapters describe:

- How to start workflows from an application.
- How to create alerts, tasks, or notifications from an application.

Chapter 10: Starting a workflow from an application

This section describes the details of the Workflow BOD and its Process/Acknowledge messages.

You can trigger workflow definitions in various ways. To trigger a workflow instance from an application, you can use these methods:

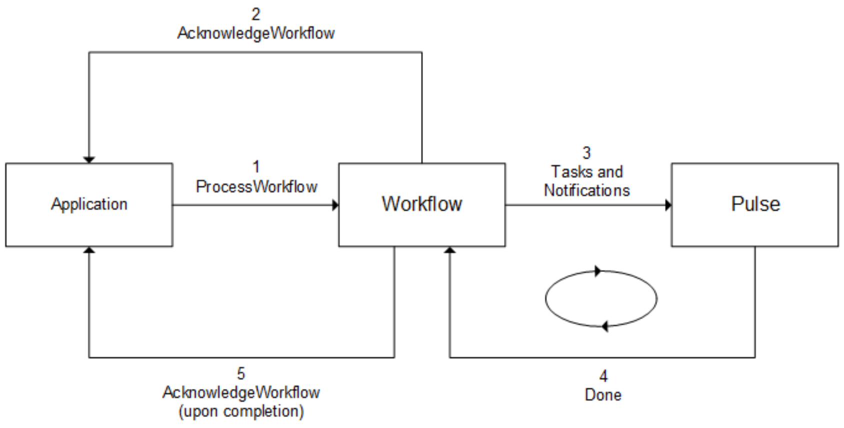
- Indirectly, by creating an activation policy or monitor that evaluates Sync BODs that are sent by this application. When a workflow instance that is started by this method completes and has output parameters. The activation policy creates a Process BOD of the same noun as the monitored Sync BOD. This Process BOD is sent to the originating application with the values resulting from the workflow execution.
- Directly, by sending a `ProcessWorkflow` BOD. When a workflow instance is started by this method, an `AcknowledgeWorkflow` is sent initially to inform that the initiation of the workflow was successful. When the workflow is completed, another `AcknowledgeWorkflow` BOD, which contains the values of the workflow output parameters, is sent back to the application.

For details about how to model and start a workflow and start workflow from activation policy and document flow, see the *Infor ION Desk User Guide*.

Starting a workflow through ProcessWorkflow

Create a workflow definition using the ION Desk workflow modeling. The actual process as modeled inside the workflow can be changed later, but your application depends on the interface of the workflow definition. The interface consists of the workflow name, the input parameters and the output parameters. To start a workflow, publish a `ProcessWorkflow` BOD, including the name of the workflow definition to be started and the values for the input parameters. This starts the workflow.

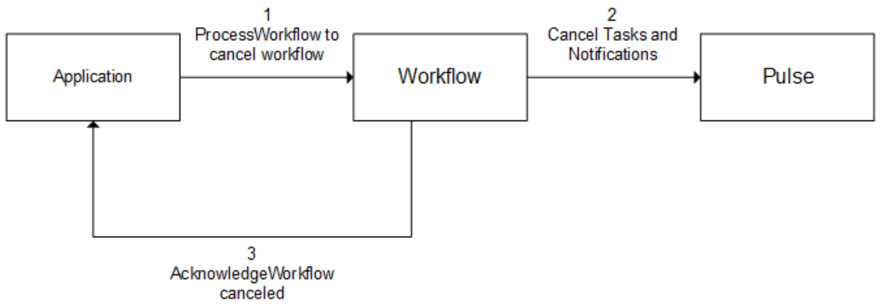
See this diagram:



You receive an `AcknowledgeWorkflow` BOD when the workflow is created. You receive also an `AcknowledgeWorkflow` BOD when the status of the workflow changes. For example, the workflow is completed or cancelled. If the workflow is completed, the BOD contains the values for the output parameters of the workflow. You can use the result in your application.

Note the difference between Workflow and Pulse BOD if you use Process verb. For Pulse BODs you receive only one `Acknowledge` BOD. For Workflow BOD you receive multiple `Acknowledge` BODs to be updated with different statues of the workflow task(s). If you started a workflow but the workflow is not relevant anymore, you can cancel it. To cancel a workflow, publish a `ProcessWorkflow` BOD.

See this diagram:



In this case you receive an `AcknowledgeWorkflow` BOD when the workflow is canceled. Specifications of how to create and cancel Workflow BOD to ION are discussed later.

To start a workflow, add this action code: `ProcessWorkflow/DataArea/Process/ActionCriteria/ActionExpression/@actionCode`.

This table shows the elements you can use in the noun instance, `ProcessWorkflow/DataArea/Workflow`:

Element	Note
WorkflowDefinitionCode	Required. This is the name of the workflow definition as modeled in ION.
Property/NameValue	Properties are required if the workflow model has input parameters. You must specify values for the input parameters that are required.

Do not use other elements, such as DocumentID and Status, when initiating a new workflow instance. These elements are determined by the Pulse engine.

The resulting AcknowledgeWorkflow BODs contain the actionCode with these possible values:

- "Accepted", when processing the request was successful.
- "Modified", to inform about an update in the workflow definition execution.
- "Rejected", if the request could not be processed.

This table shows the elements that are included in the `AcknowledgeWorkflow/Workflow` section if the actionCode is "Accepted":

Element	Note
DocumentID/ID	Unique identification of the workflow instance in this ION installation.
Status/Code	Value is "Initial" to indicate the workflow was started.
WorkflowDefinitionCode	The name of the workflow started.

This table shows the elements that are included in the `AcknowledgeWorkflow/Workflow` section if the actionCode is "Modified":

Element	Note
DocumentID/ID	Unique identification of the workflow instance in this ION installation.
Status/Code	Can be "Cancelled", "Failed", or "Completed".
Status/Reason	Available for Status/Code "Cancelled" or "Failed".
WorkflowDefinitionCode	The name of the workflow that was canceled, failed, or completed.
Property/NameValue	Only available if the Status/Code is "Completed" and the workflow has output parameters. The Properties contain the resulting values of the workflow output parameters.

This table shows the elements that are included in the `AcknowledgeWorkflow/Workflow` section if the actionCode is "Rejected":

Element	Note
Status/Code	Value is "Failed".
Status/Reason	The reason for failure.

Canceling a workflow through ProcessWorkflow

To cancel a workflow, the action code, `ProcessWorkflow/DataArea/Process/ActionCriteria/ActionExpression/@actionCode`, must be "Change".

This table shows the elements you can use in the noun instance, `ProcessWorkflow/DataArea/Workflow`:

Element	Note
DocumentID/ID	Required. Unique identification of the workflow instance that must be canceled.
WorkflowDefinitionCode	Required. This is the name of the workflow definition as modeled in ION.
Status/Code	Must be "Cancelled".

The resulting AcknowledgeWorkflow BOD contains the actionCode="Accepted" if cancelation was performed, or actionCode="Rejected" if the cancelation was not possible. The other elements included are similar to those described for AcknowledgeWorkflow BODs.

See Starting a workflow through ProcessWorkflow on page 50.

Workflow BOD details

The definitions of ProcessWorkflow and AcknowledgeWorkflow are available on this site:

http://schema.infor.com

This table shows the elements that exist in these documents:

Element	Note
DocumentID/ID	Unique identification of the workflow instance in this ION installation.
Status/Code	Can have these values: • "Initial": the request to start a new workflow instance was performed successfully. • "Completed": the workflow instance completed successfully. • "Cancelled": the workflow instance was canceled. • "Failed": the execution of the workflow instance failed.
Status/Reason	Available for Status/Code "Cancelled" or "Failed".
WorkflowDefinitionCode	This is the name of the workflow definition as modeled in ION.

Element	Note
Property/NameValue	Values of the workflow input and output parameters. Must match the data type of the parameter. When included in a ProcessWorkflow with actionCode="Add", these are input parameters. When included in an AcknowledgeWorkflow with actionCode="Completed", these are output parameters.
Property/NameValue/@name	The name of the parameter as defined in the workflow model.
Property/NameValue/@type	One of the pre-defined types that you can map to the workflow parameter types. These types are supported: • IndicatorType • NumericType • IntegerNumericType • StringType • DateType • DateTimeType See the table below for an overview of mapping to workflow parameter types.
TreeProperty/TreeNode	TreeProperty contains data for a workflow structure. In the first TreeNode, these attributes are used: • ID = 1 • NodeName = the name of the structure as used in workflow • One or more NodeProperty elements with fields from the root of the structure The first node does not have a ParentID.
TreeProperty/TreeNode/ID	The unique identifier for this tree node within the document. It must only be unique within the document.
TreeProperty/TreeNode/ParentID	The ID of the node that is the parent of the current node within the tree.
TreeProperty/TreeNode/NodeName	Must be identical to the level name from the workflow structure.
TreeProperty/TreeNode/NodeProperty/NameValue	Values for the fields from the workflow structure situated on the level corresponding to the current node.
TreeProperty/TreeNode/NodeProperty/NameValue/@name	The name of the field as defined in the workflow structure.

Element	Note
TreeProperty/TreeNode/NodeProperty/NameValue/@type	The type of the field as defined in the workflow structure. These types are supported: • IndicatorType • DateType • NumericType • IntegerNumericType • StringType • DateTimeType

This table shows all available workflow parameter types and how these types are mapped to the Property types in the Workflow BOD:

Workflow Parameter Type	Property Type	Notes
Boolean	IndicatorType	Represents a true or false value. Possible values: true, false, 0, 1.
Code	StringType	The value is expected to be part of the Codes modeled in the Workflow Modeler, but no validation is enforced.
Date	DateType	The date part of a date/time stamp.
DateTime	DateTimeType	The date part and time part of a date/time stamp, separated by "T" and ending with Z (is always UTC).
Decimal	NumericType	A numeric type with a floating precision. The Decimal data type in Workflow corresponds to the double data type and is a double-precision 64-bit IEEE 754 floating point.
Hyperlink	StringType	Is displayed as a clickable link in Infor Ming.le.
Integer	IntegerNumericType	A numeric type that represents a whole number.
String	StringType	A string value of up to 4000 characters in length.

Sample workflow BODs

Sample ProcessWorkflow to start a workflow

```
<ProcessWorkflow xmlns="http://schema.infor.com/InforOAGIS/2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLoca
tion="http://schema.infor.com/InforOAGIS/2 http://schema.infor.com/Trunk/In
forOAGIS/BODs/Developer/ProcessWorkflow.xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" releaseID="11.1" version
ID="2.10.0">
   <ApplicationArea>
      <Sender>
         <LogicalID>lid://infor.test.app1</LogicalID>
         <ComponentID>ComponentID0</ComponentID>
         <TaskID>TaskID0</TaskID>
         <AuthorizationID>AuthorizationID0</AuthorizationID>
      </Sender>
      <CreationDateTime>2013-10-20T13:20:00Z</CreationDateTime>
      <BODID>infor-nid:infor:::Sample_NestedTree:1?Workflow&amp;verb=Pro
cess</BODID>
   </ApplicationArea>
   <DataArea>
      <Process>
         <TenantID>infor</TenantID>
         <ActionCriteria>
            <ActionExpression actionCode="Add"/>
         </ActionCriteria>
      </Process>
      <Workflow>
      <Status>
      <Code>Initial</Code>
      </Status>
     <WorkflowDefinitionCode>SimpleDataEntryTest</WorkflowDefinitionCode>

      <Property>
      <!-- true, false, 1 or 0 -->
      <NameValue name="aBoolean" type="IndicatorType">true</NameValue>
      </Property>
      <Property>
      <NameValue name="aCode" type="StringType">Approved</NameValue>
      </Property>
      <Property>
      <NameValue name="aDate" type="DateType">2012-12-10</NameValue>
      </Property>
      <Property>
      <NameValue name="aDateTime" type="DateTimeType">2012-12-
10T10:00:00Z</NameValue>
      </Property>
      <Property>
      <NameValue name="aDecimal" type="NumericType">453.99</NameValue>
      </Property>
      <Property>
     <NameValue name="anInteger" type="IntegerNumericType">250</NameValue>

      </Property>
      <Property>
      <NameValue name="aLink" type="StringType">http://www.inforx
treme.com</NameValue>
      </Property>
      <Property>
      <NameValue name="aString" type="StringType">This is a test
string</NameValue>
      </Property>
      <TreeProperty>
      <TreeNode>
      <ID>1</ID>
      <NodeName>Building</NodeName>
      <NodeProperty>
     <NameValue name="BuildingName" type="StringType">Office</NameValue>

      </NodeProperty>
      <NodeProperty>
     <NameValue name="NumberOfFloors" type="IntegerNumericType">1</NameVal
ue>
      </NodeProperty>
      </TreeNode>
      <TreeNode>
      <ID>2</ID>
      <ParentID>1</ParentID>
      <NodeName>Floor</NodeName>
      <NodeProperty>
      <NameValue name="FloorName" type="StringType">Ground
Floor</NameValue>
      </NodeProperty>
      </TreeNode>
      </TreeProperty>
      </Workflow>
   </DataArea>
</ProcessWorkflow>
```

Sample AcknowledgeWorkflow when the request was accepted

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<AcknowledgeWorkflow releaseID="2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLoca
tion="http://schema.infor.com/InforOAGIS/2 http://schema.infor.com/2.5.0/In
forOAGIS/BODs/Developer/AcknowledgeWorkflow.xsd" xmlns="http://schema.in
for.com/InforOAGIS/2">
   <ApplicationArea>
      <Sender>
```

```
         <LogicalID>infor.engine.workflow</LogicalID>
         <ComponentID>ION_Workflow_Engine</ComponentID>
      </Sender>
      <CreationDateTime>2013-10-24T08:18:18.380Z</CreationDateTime>
   </ApplicationArea>
   <DataArea>
      <Acknowledge>
         <TenantID>Infor</TenantID>
         <OriginalApplicationArea>
            <Sender>
               <LogicalID>lid://infor.test.app1</LogicalID>
               <ComponentID>ComponentID0</ComponentID>
               <TaskID>TaskID0</TaskID>
               <AuthorizationID>AuthorizationID0</AuthorizationID>
            </Sender>
            <CreationDateTime>2013-10-20T13:20:00Z</CreationDateTime>
        <BODID>infor-nid:infor:::Sample_NestedTree:1?Workflow&amp;verb=Pro
cess</BODID>
         </OriginalApplicationArea>
         <ResponseCriteria>
            <ResponseExpression actionCode="Accepted"/>
         </ResponseCriteria>
      </Acknowledge>
      <Workflow>
         <DocumentID>
            <ID>20</ID>
         </DocumentID>
         <Status>
            <Code>Initial</Code>
         </Status>
         <WorkflowDefinitionCode>SimpleDataEntryTest</WorkflowDefinition
Code>
      </Workflow>
   </DataArea>
</AcknowledgeWorkflow>
```

Chapter 11: Creating alerts, tasks, or notifications from an application

The Pulse engine is the component that handles alerts, tasks, and notifications as generated by Event Management and Workflow.

But any application that is connected to ION can create alerts, tasks, and notifications in Pulse directly. This is done by sending and receiving Pulse BODs such as. `PulseAlert`, `PulseTask`, `PulseNotification`.

This section explains how to create and manage alerts, tasks, and notifications through Pulse BODs.

From an application you can perform these actions:

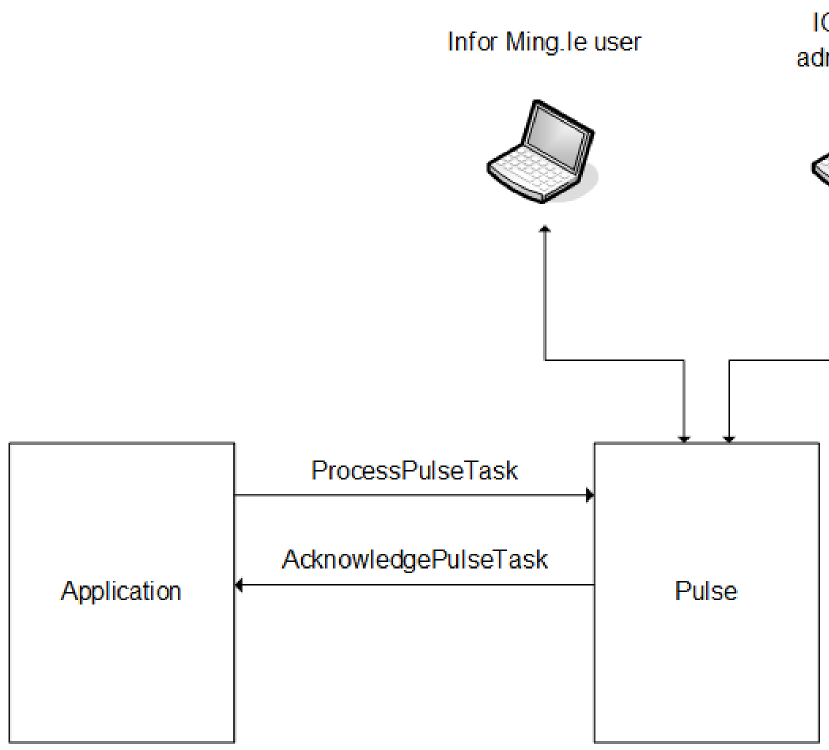
- Create alerts, tasks, or notifications.
- Receive status updates for an alert, task, or notification.
- Cancel a previously created alert, task, or notification.

Creating alerts, tasks, and notifications is not limited to applications that can send or receive BODs. For example, you can use the corresponding type of connection point to create alerts through a JMS message queue or by reading from a database.

Creating alerts, tasks, or notifications

Use `PulseTask` if the user is expected to perform a defined task. Use `PulseAlert` to notify a user of an exception. The user can decide whether to take action and then close the alert. Use `PulseNotification` to send a message 'for your information' to one or more users.

The diagram shows how tasks, alerts and notifications are created by sending Pulse BODs. In the diagram a task (`PulseTask`) is shown, but the process is the same for notifications (`PulseNotification`) and alerts (`PulseAlert`).



By publishing a `ProcessPulseTask` BOD you can create a new task in Pulse. The task is created in Pulse and an `AcknowledgePulseTask` is sent in reply.

Creating tasks from an application

To create tasks from an application:

1. Update your application so it can send `ProcessPulseTask` and to handle incoming `AcknowledgePulseTask` BODs. The `ProcessPulseTask` BOD must have action code ‘Add’ to create a new task.
2. Ensure your application is connected to ION. In the connection point, select `ProcessPulseTask` as a document to send.
3. Your application connection point must be used in at least one active document flow. If this is not the case, you can create a specific document flow containing an activity for your application and activate that flow.
4. In IFS, configure users, distribution groups, and contacts for people that must receive alerts. For users, ensure the 'Person' is filled with the value that is sent by the application as the person ID of a system user. For contacts, ensure the ‘Contact ID’ is filled with the same value that is sent by the application as the person ID of a non-system user. For distribution groups, the application must send the distribution group name as defined in IFS.

   For `PulseAlert` and `PulseNotification` BODs, the procedure is the same.

Important notes

Regarding the document flow, be aware that Pulse is an 'engine' inside ION. It is not a normal application for which you can create a connection point. Therefore, the configuration differs from a normal configuration. Normally, a document flow is created from A to B where you select the documents to be sent. This selection can be a subset of the documents sent by connection point A and received by connection point B.

If an application connection point that can publish `ProcessPulseAlert`, `ProcessPulseTask`, or `ProcessPulseNotification`, is used in an active document flow, these BODs are delivered directly to Pulse. Similarly, if the connection point is configured to receive `SyncPulseAlert`, `SyncPulseTask`, or `SyncPulseNotification`, it receives those BODs without selecting them in a document flow.

Therefore, you cannot use mapping, content-based routing, or filtering in a document flow for `PulseAlert`, `PulseTask`, and `PulseNotification`.

For details on how to define connection points and document flows, see the *Infor ION Desk User Guide*.

Creating an alert

When creating an alert, the action code (`ProcessPulseAlert/DataArea/Process/ActionCriteria/ ActionExpression/@actionCode`) must be "Add".

This table shows the elements you can use in the noun instance, `ProcessPulseAlert/DataArea/PulseAlert`:

Element	Note
Description	Required
Note	Optional
`AlertDetail` and child elements	Optional
`DistributionPerson` or `DistributionGroup`	Define at least one `DistributionPerson` or `DistributionGroup`. For a `DistributionPerson`, do not specify the `Distribution/ID`, because it is generated by Pulse.

Do not use other elements, such as DocumentID, CreationDateTime, and Status, when creating an alert. These elements are determined by the Pulse engine.

This table shows the elements that are included in the resulting AcknowledgePulseAlert BOD:

Element	Note
`DocumentID/ID`	Available if `actionCode` = "Accepted". If the alert cannot be created successfully, the `actionCode` is "Rejected".

Note: We recommend that you include a unique value for the BODID. You can use that to process the `AcknowledgePulseAlert` because the BODID is available again in the original application area.

If the alert could not be created, you receive an `AcknowledgePulseAlert` BOD with action code 'Rejected'.

If the `AcknowledgePulseAlert` BOD can not be delivered, a Confirm BOD is generated.

Creating a task

When creating a task, the action code (`ProcessPulseTask/DataArea/Process/ActionCriteria/ActionExpression/@actionCode)` must be "Add".

This table shows the elements you can use in the noun instance (`ProcessPulse Task/DataArea/PulseTask`):

Element	Note
Priority	Optional, default is MEDIUM.
Description	Required
Note	Optional
Parameter and child elements	Optional
`DistributionPerson` or `DistributionGroup`	Define at least one `DistributionPerson` or `DistributionGroup`. For a `DistributionPerson`, do not specify the `Distribution/ID`, because it is generated by Pulse.

Do not use other elements, such as DocumentID, CreationDateTime, and Status, when creating a task. These elements are determined by the Pulse engine.

This table shows the elements that are included in the resulting AcknowledgePulseTask BOD:

Element	Note
`DocumentID/ID`	Available if `actionCode` = "Accepted". If the task cannot be created successfully, the `a ctionCode` is "Rejected".

Note: We recommend that you include a unique value for the BODID. You can use that to process the `AcknowledgePulseTask` because the BODID is available again in the original application area.

If the task could not be created, you receive an `AcknowledgePulseTask` BOD with action code 'Rejected'.

If the `AcknowledgePulseTask` BOD can not be delivered, a Confirm BOD is generated.

Creating a notification

When creating a notification, the action code (`ProcessPulseNotification/DataArea/ Process/ActionCriteria/ActionExpression/@actionCode`) must be "Add".

This table shows the elements you can use in the noun instance (`ProcessPulseNotification/DataArea/PulseNotification`):

Element	Note
Description	Required
Parameter and child elements	Optional
`DistributionPerson` or `DistributionGroup`	Define at least one `DistributionPerson` or `DistributionGroup`. For a `DistributionPerson`, do not specify the `Distribution/ID`, because it is generated by Pulse.

Do not use other elements, such as `DocumentID`, `CreationDateTime`, and Status, when creating a notification. These elements are determined by the Pulse engine.

This table shows the elements that are included in the resulting `AcknowledgePulseNotification` BOD:

Element	Note
`DocumentID/ID`	Available if `actionCode` = "Accepted". If the notification cannot be created successfully, the `actionCode` is "Rejected".

Note: We recommend that you include a unique value for the BODID. You can use that to process the `AcknowledgePulseNotification` because the BODID is available again in the original application area.

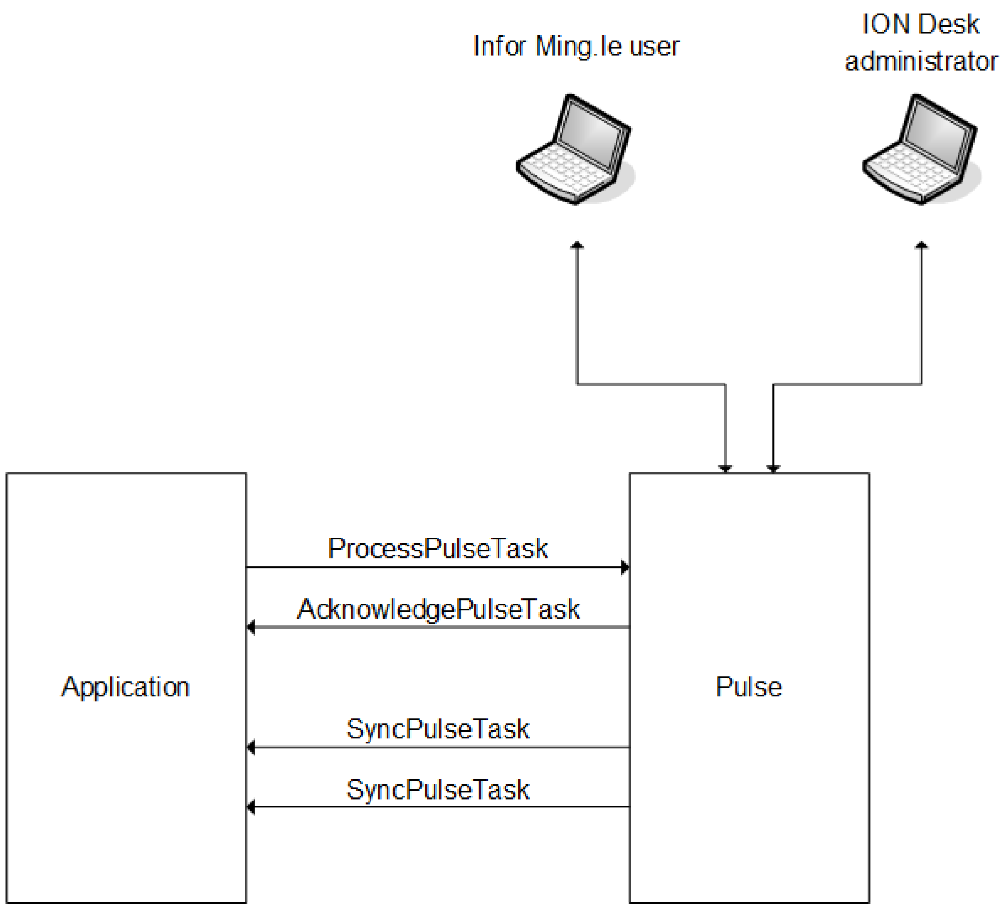
If the notification could not be created, you receive an `AcknowledgePulseNotification` BOD with action code 'Rejected'.

If the `AcknowledgePulseNotification` BOD can not be delivered, a Confirm BOD is generated.

Receiving status updates on alerts, tasks, or notifications

The Pulse engine sends Sync BODs for `PulseAlert`, `PulseTask` and `PulseNotification` to inform others about status changes in alerts, tasks and notifications.

This diagram shows a task (`PulseTask`), but the process is the same for notifications and alerts.



Pulse sends a `SyncPulseTask` when a new task is created and when the status is changed. For example, if a user picks up the task and sets it to done, a `SyncPulseTask` is sent. The application can handle this BOD to be informed about the new status. If a task is canceled through ION Desk by an administrator, a `SyncPulseTask` is also sent.

A Sync BOD is sent in these situations:

- A new task, alert or notification is created.
- An item is assigned, reassigned or unassigned.
- An item is redistributed.
- The status of an item is changed to Done or Canceled.

Pulse may not send Sync BODs for all changes to an item, such as changing a parameter value, adding a note or adding an attachment.

It is not required to handle Sync BODs when you send out Process BODs. The Acknowledge BOD tells you whether the Process request was handled successfully. When sending a `ProcessPulseNotification`, that is probably all you want to know. When creating a task, you want to know whether the task was completed and what was the outcome of the task. For example, in case of an approval task for a requisition you want to know whether the user approved or rejected the requisition. In that case you can receive the `SyncPulseTask` BODs to be informed of the task status.

It can happen that an Acknowledge Pulse BOD or a Sync Pulse BOD could be delivered to the subscribing application. In that case, a Confirm BOD is generated. The Confirm BOD contains the original Acknowledge or Sync document, and can be re-submitted later from the ION Desk UI

Receiving status updates

To receive status updates for tasks:

1 Update your application so it can handle incoming SyncPulseTask BODs. The SyncPulseAlert/DataArea/PulseTask/Source element contains the originator of the alert, so you can use this element to ignore BODs that are not relevant for your application.

2 Ensure your application is connected to ION. In the connection point, select SyncPulseTask as a document to receive.

3 Your application connection point must be used in at least one active document flow. If this is not the case, you can create a specific document flow containing an activity for your application and activate that flow.

For PulseAlert and PulseNotification BODs, the procedure is the same.

The SyncPulseAlert, SyncPulseTask, and SyncPulseNotification BODs include all elements that are relevant for the alert, task, or notification.

For the available elements, see Pulse BOD details on page 69.

Receiving information about deleted activities

In ION Desk you can delete archived monitors, archived activation policies, and archived workflows. When you delete these, the activities they have generated are also deleted. If an application has subscribed to receive status updates from Sync Pulse BODs, it also receives Sync Pulse BODs about deleted activities.

The messages about deleted activities have these characteristics:

- Depending on the type of activities being deleted, a SyncPulseAlert, SyncPulseTask, or a SyncPulseNotification is sent.
- These BODs have actionCode="Delete".
- These BODs contain a list of IDs that represent a batch of activities being deleted simultaneously.

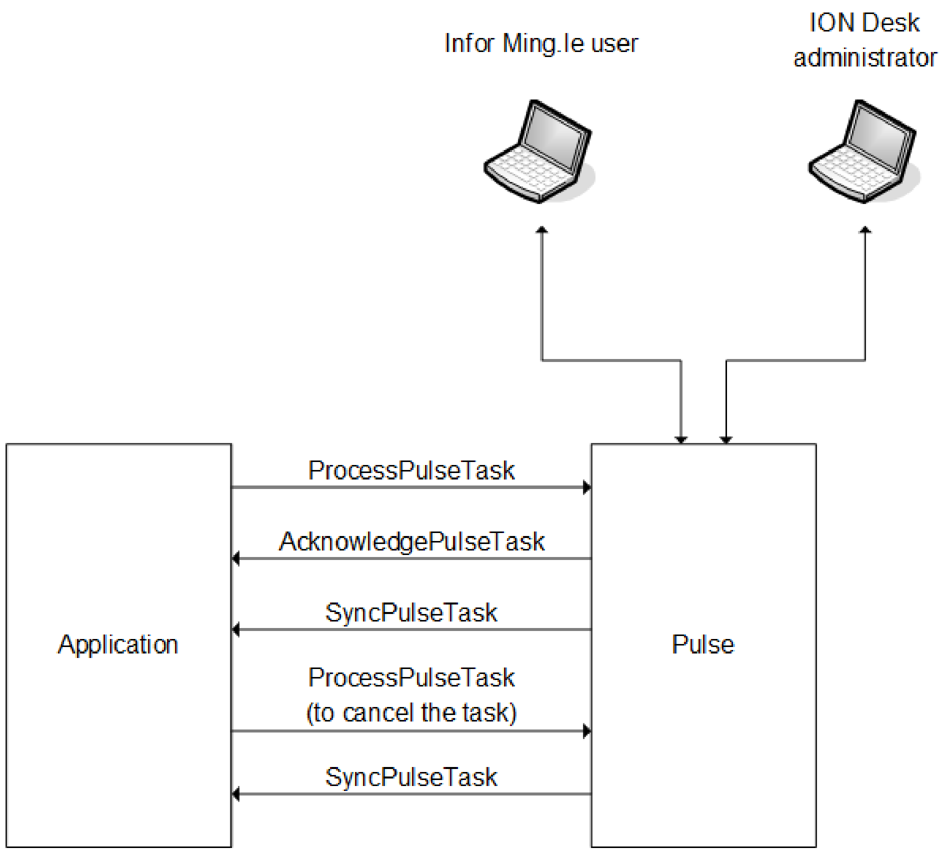
This code shows an example of a message that is sent for a set of deleted alerts:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SyncPulseAlert releaseID="10.1.3"
  xmlns="http://schema.infor.com/InforOAGIS/2">
 <ApplicationArea>
  <Sender>
   <LogicalID>infor.engine.pulse</LogicalID>
   <ConfirmationCode>OnError</ConfirmationCode>
  </Sender>
  <CreationDateTime>2018-01-18T15:16:50.397Z</CreationDateTime>
 </ApplicationArea>
 <DataArea>
  <Sync>
   <TenantID>INFOR</TenantID>
   <ActionCriteria>
    <ActionExpression actionCode="Delete"/>
   </ActionCriteria>
  </Sync>
  <PulseAlert>
   <DocumentID>
    <ID>90</ID>
   </DocumentID>
  </PulseAlert>
  <PulseAlert>
   <DocumentID>
    <ID>91</ID>
   </DocumentID>
  </PulseAlert>
  <PulseAlert>
   <DocumentID>
    <ID>92</ID>
   </DocumentID>
  </PulseAlert>
 </DataArea>
</SyncPulseAlert>
```

Canceling alerts, tasks, or notifications

In some cases, you must cancel an alert, task, or notification before the user handled it. For example, a requisition approval task is created, but now the requestor cancels the requisition. In this case you must also cancel the task in Pulse.

This diagram shows a task (PulseTask), but the process is the same for notifications and alerts.



If you created a task by sending a ProcessPulseTask with action code 'Add', you receive an AcknowledgePulseTask which contains the ID of the task. You can use this ID to send a new ProcessPulseTask request to cancel the task.

Receiving status updates

To receive status updates for tasks:

1. Update your application so it can handle incoming SyncPulseTask BODs. The SyncPulseAlert/DataArea/PulseTask/Source element contains the originator of the alert, so you can use this element to ignore BODs that are not relevant for your application.
2. Ensure your application is connected to ION. In the connection point, select SyncPulseTask as a document to receive.
3. Your application connection point must be used in at least one active document flow. If this is not the case, you can create a specific document flow containing an activity for your application and activate that flow.

For PulseAlert and PulseNotification BODs, the procedure is the same.

The SyncPulseAlert, SyncPulseTask, and SyncPulseNotification BODs include all elements that are relevant for the alert, task, or notification.

For the available elements, see Pulse BOD details on page 69.

Canceling an alert

When canceling an alert, the action code (ProcessPulseAlert/DataArea/Process/ActionCriteria/ActionExpression/@actionCode) must be "Change".

This table shows the elements you must use in the noun instance (ProcessPulseAlert/DataArea/PulseAlert):

Element	Note
DocumentID/ID	Use the DocumentID/ID as provided in the Acknowledge BOD when the item was created.
Status/Code	Use value 'CANCELLED'.

When processing the cancel request in the Pulse engine, an Acknowledge BOD is sent in reply. If the item was canceled successfully, the actionCode of the BOD is "Accepted". If the item could not be canceled, the actionCode is "Rejected".

You can only cancel an alert when:

- The alert is open. If the user completed the alert, you can no longer cancel it.
- The alert is created by a ProcessPulseAlert BOD. If the alert is created by a Monitor you cannot cancel the alert using a ProcessPulseAlert BOD.

Canceling a task

When canceling a task, the action code (ProcessPulseTask/DataArea/Process/ActionCriteria/ActionExpression/@actionCode) must be "Change".

This table shows the elements you must use in the noun instance (ProcessPulsetask/DataArea/PulseTask):

Element	Note
DocumentID/ID	Use the DocumentID/ID as provided in the Acknowledge BOD when the item was created.
Status/Code	Use value 'CANCELLED'.

When processing the cancel request in the Pulse engine, an Acknowledge BOD is sent in reply. If the item was canceled successfully, the actionCode of the BOD is "Accepted". If the item could not be canceled, the actionCode is "Rejected".

You can only cancel a task when:

- The task is open. If the user completed the task, you can no longer cancel it.
- The task is created by a ProcessPulseTask BOD. If the task is created by a Workflow you cannot cancel the task using a ProcessPulseTask BOD.

Canceling a notification

When canceling a notification, the action code (ProcessPulseNotification/DataArea/Process/ActionCriteria/ActionExpression/@actionCode) must be "Change".

This table shows the elements you must use in the noun instance (ProcessPulseNotification/DataArea/PulseNotification):

Element	Note
DocumentID/ID	Use the DocumentID/ID as provided in the Acknowledge BOD when the item was created.
Status/Code	Use value 'CANCELLED'.

When processing the cancel request in the Pulse engine, an Acknowledge BOD is sent in reply. If the item was canceled successfully, the actionCode of the BOD is "Accepted". If the item could not be canceled, the actionCode is "Rejected".

You can only cancel a notification when:

- The notification is open. If the user completed the task, you can no longer cancel it.
- The notification is created by a ProcessPulseNotification BOD. If the notification is created by a Workflow you cannot cancel the notification using a ProcessPulseNotification BOD.

Pulse BOD details

The definition of ProcessPulseAlert, AcknowledgePulseAlert, and SyncPulseAlert is available on this site:

http://schema.infor.com

This section contains an explanation on the elements that exist in these documents.

PulseAlert

This table shows the elements in PulseAlert:

Element	Note
DocumentID/ID	Unique identification of the alert.
CreationDateTime	Date when the alert was created in Pulse.
LastModificationDateTime	Date when the alert was last modified.

Element	Note
Status/Code	Status of the alert. Values are: • NEW - The initial status for a new item. • ASSIGNED - Assigned to a specific user. • UNASSIGNED - No longer assigned to a specific user. • DONE - Completed. • CANCELLED - canceled by an administrator (through ION Desk) or by the application (through a ProcessPulseAlert BOD).
IsEscalated	This indicator has value true if an alert is escalated and false otherwise. This element is only supported in Sync messages.
EscalationLevel	The number of levels in the organizational hierarchy to which the alert is escalated. If the alert is not escalated the value is 0. Otherwise the EscalationLevel is greater than 0.
DueDateTime	Date and time when the alert is due. This element is shown only if a due date was configured in the monitor that created this alert. This element is supported only in the Sync messages.
Description	The description that is displayed to the end user as the summary of the alert. You can use hash tags to make searching easier. For example: `Late shipment for #sales order 25 of customer #acme` To define a category, use `##` at the end of the message. To include values from the alert details, use square brackets around the parameter labels. To use the actual characters for square brackets, use an escape character: `\[` or `\]`
Description/@languageID	The language code of the Description field. For details, see the notes in the "Supported features" section.
Note	Notes that are added by people who handled the alert. Notes can be added, but cannot be modified or removed.
Note/@userID	The ID (personId) of the person who added the note.

Element	Note
Note/@author	Full name of the person who added the note. This attribute is required when a Note is added through a ProcessPulseAlert BOD." / "ProcessPulseTask BOD." / "ProcessPulseNotification BOD.
Note/@entryDateTime	The date/time at which the note was created.
Note/@noteID	Identification of the note within the alert.
Source/Type	BOD: the alert was created by sending a ProcessPulseAlert BOD. MONITOR: the alert was created by an event monitor.
Source/Name	If Type=BOD: the logical ID of the sender of the ProcessPulseAlert BOD. For example, `lid://infor.erp.myerp` If Type is MONITOR: the name of the monitor that created the alert. For example, `MyMonitor`.
AlertDetail	Details of the alert that are displayed to the user. Each AlertDetail group contains document references or trees. For example, an AlertDetail can contain one or more document references followed by a tree containing data for an order and its order lines.
AlertDetail/@sequence	The sequence number of the AlertDetail group. This is used for ordering the alert details when displaying them to a user.

Element	Note
AlertDetail/PulseDocumentReference	Reference to another business document. The format is the same as the standard DocumentReference, but additionally it has a sequence attribute. For example: `<PulseDocumentReference>` `type="SalesOrder" sequence="1">` `<DocumentID>` `<ID accountingEntity="infor"` `location="bvld"` `lid="lid://infor.ln.440">` `ORD0015236</ID>` `<RevisionID>123</RevisionID>` `</DocumentID>` `</PulseDocumentReference>` The RevisionID tag is only used if the document referred from this alert contains a RevisionID.
AlertDetail/PulseDocumentReference/@sequence	Sequence number to indicate the sequence in which the document references must be displayed to the user when showing the alert details.
AlertDetail/TreeNode	Node in the data tree. Alert data is a tree structure to enable multi-level data objects, such as an order header having order lines.
AlertDetail/TreeNode/@sequence	If the tree node is a child of another node, the sequence attribute defines the sequence of the child nodes relative to their parent.
AlertDetail/TreeNode/ID	Identification for this tree node within the alert.
AlertDetail/TreeNode/ParentID	Omitted if the tree node is the top-level node in the tree. Otherwise it contains the ID of the parent node. The parent node must exist within the same AlertDetail. In an AlertDetail, all TreeNodes except one will have a ParentID.
AlertDetail/TreeNode/NodeName	The name of the node. This name is displayed to the user. For example: `Sales Order`.

Element	Note

AlertDetail/TreeNode/TreeNodeParameter	A parameter in a tree node, which contains a data element that can be displayed to the user who handles the alert. Alert parameters are always read-only.
	For example:
	`<TreeNodeParameter sequence="1">`
	`  <Name>OrderNumber</Name>`
	`  <Value>12345</Value>`
	`  <DataType`
	`      listID="Pulse`
	`Datatypes">STRING`
	`  </DataType>`
	`  <Label>Order Number</Label>`
	`</TreeNodeParameter>`

AlertDetail/TreeNode/TreeNodeParameter/Name	The name that identifies the parameter within the TreeNode. This element is required for each parameter.
AlertDetail/TreeNode/TreeNodeParameter/Value	The (serialized) value of the parameter. The formatting depends on the DataType and is the same as the formatting that is normally used in BOD data elements.
AlertDetail/TreeNode/TreeNodeParameter/DataType	The data type of the parameter. This element is required for each parameter. You can use these data types: • STRING - a string value that is up to 4000 characters in length • INTEGER - a numeric type that represents a whole number • DECIMAL - a numeric type that has a floating precision. Values may be expressed using the scientific e-notation. For details about the scientific e-notation, see Wikipedia or other resources on the internet. • BOOLEAN - represents a true or false value • DATETIME - the date part and time part of a date/time stamp separated by "T" and ending with Z (is always UTC) • TIME - the time part of a date/time stamp • DATE - the date part of a date/time stamp • DURATION - time interval, starting with P followed by nM (minutes) or nH (hours) or nD (days). For example, P2D3H.

Element	Note
AlertDetail/TreeNode/TreeNodeParameter/Label	The label of the parameter that is used when displaying the parameter to a user. This element is required for each parameter.
AssignedPerson	The user to which the alert is currently assigned.
AssignedPerson/PersonReference	Reference to a person.
AssignedPerson/PersonReference/IDs/ID	Identifier of the person. The ID must be a Person ID or Contact ID as defined in IFS (case-sensitive).
AssignedPerson/PersonReference/Name	Name of the person.
AssignedPerson/PersonReference/SystemUserIndicator	Value is `true` if the person is a User in IFS. Value is `false` if the person is a Contact in IFS.
DistributionPerson	Person in the distribution list for the alert. If the alert is not assigned to a person, one of these persons can pick it up.
DistributionPerson/ID	Identification of the distribution person within the alert
DistributionPerson/PersonReference	Reference to a person.
DistributionPerson/PersonReference/IDs/ID	Identifier of the person. The ID must be a Person ID or Contact ID as defined in IFS (case-sensitive). This element is required for each distribution person.
DistributionPerson/PersonReference/Name	Name of the person.
DistributionPerson/PersonReference/SystemUserIndicator	Value is `true` if the person is a User in IFS. Value is `false` if the person is a Contact in IFS. This element is required for each distribution person.
DistributionGroup	Specify one or more distribution groups to which the alert must be distributed. This element can be used in addition to the DistributionPerson element, or instead of the DistributionPerson element.
DistributionGroup/Name	Identifier of the distribution group as defined in Infor Ming.le User Management. The alert is distributed to all users that are members of this group at the time the alert is created.
DistributionGroup/Description	Description of the distribution group. This element is optional and is not used to determine the distribution list.

Element	Note
DistributionGroup/Description/@languageID	Describe the language code of the description element. This attribute is optional and is not used for the distribution functionality.

Note: The DistributionGroup element is supported only in the ProcessPulseAlert BOD. A SyncPulseAlert BOD can be sent after the creation of the alert. In that case, the distribution list of the alert is described using a DistributionPerson element for each user from the distribution group.

PulseTask

This table shows the elements in PulseTask:

Element	Note
DocumentID/ID	Unique identification of the task.
CreationDateTime	Date when the task was created in Pulse.
LastModificationDateTime	Date when the task was last modified.
Status/Code	Status of the task. Values are: • NEW - The initial status for a new item. • ASSIGNED - Assigned to a specific user. • UNASSIGNED - No longer assigned to a specific user. • DONE - Completed. • CANCELLED - canceled by an administrator (through ION Desk) or by the application (through a ProcessPulseTask BOD).
IsEscalated	This indicator has value true if a task is escalated and false otherwise. This element is only supported in Sync messages.
EscalationLevel	The number of levels in the organizational hierarchy to which the alert is escalated. If the task is not escalated the value is 0. Otherwise the EscalationLevel is greater than 0.
Priority	The priority of the task. Values are: • HIGH • MEDIUM • LOW
DueDateTime	Date and time when the task is due. This element is shown only if a due date was configured in the workflow task properties. This element is supported only in the Sync messages.

Element	Note
Description	The description that is displayed to the end user as the summary of the task. You can use hash tags to make searching easier. For example: `Approve #requisition 25` To define a category, use `##` at the end of the message. To include values from the alert details, use square brackets around the parameter labels. To use the actual characters for square brackets, use an escape character: `\[` or `\]`
Description/@languageID	The language code of the Description field. For details, see the notes in the "Supported features" section.
Note	Notes that are added by people who handled the alert. Notes can be added, but cannot be modified or removed.
Note/@userID	The ID (personId) of the person who added the note.
Note/@author	Full name of the person who added the note. This attribute is required when a Note is added through a ProcessPulseAlert BOD." / "ProcessPulseTask BOD." / "ProcessPulseNotification BOD.
Note/@entryDateTime	The date/time at which the note was created.
Note/@noteID	Identification of the note within the task.
Note/@type	If this attribute is missing or empty "", this note is a current note of this Task. If the @type attribute is filled with a string other than "", this note is a propagated Note, which was specified in a previous Task of the same workflow.
Source/Type	BOD: the task was created by sending a ProcessPulseTask BOD. WORKFLOW: the task was created by a workflow.

Element	Note
Source/Name	If Type=BOD: the logical ID of the sender of the ProcessPulseTask BOD. For example, `lid://infor.erp.myerp` If Type is WORKFLOW: the name of the workflow definition that created the task. For example, `My Workflow`.

Parameter	Data of the task that is displayed to the user and optionally can be updated by the user.
	For example:
	`<Parameter sequence="1">`
	`  <Name>OrderNumber</Name>`
	`  <Value>12345</Value>`
	`  <DataType`
	`      listID="Pulse Datatypes">STRING</DataType>`
	`  <Label>Order Number</Label>`
	`  <ReadOnlyIndicator>true</ReadOnlyIndicator>`
	`</Parameter>`

Parameter/@sequence	The sequence number of the parameter. This is used for ordering the parameters when displaying them to a user. You are only allowed to use this attribute if there are no elements of type TreeParameter in the document.
Parameter/Sequence	You must specify this element in combination with TreeParameter/Sequence. This element is used to determine the order in which Parameters and TreeParameters are displayed to the user.
Parameter/Name	The name that identifies the parameter within the task. This element is required for each parameter.
Parameter/Value	The (serialized) value of the parameter. The formatting depends on the DataType and is the same as the formatting that is normally used in BOD data elements.

Element	Note
Parameter/DataType	The data type of the parameter. This element is required for each parameter. You can use these data types: • STRING - a string value that is up to 4000 characters in length • INTEGER - a numeric type that represents a whole number • DECIMAL - a numeric type that has a floating precision. Values may be expressed using the scientific e-notation. For details about the scientific e-notation, see Wikipedia or other resources on the internet. • BOOLEAN - represents a true or false value • DATETIME - the date part and time part of a date/time stamp separated by "T" and ending with Z (is always UTC) • TIME - the time part of a date/time stamp • DATE - the date part of a date/time stamp
Parameter/Label	The label of the parameter that is used when displaying the parameter to a user. This element is required for each parameter.
Parameter/Label /@languageID	The language code of the Label field. For details, see the notes in the "Supported features" section.
Parameter/ReadyOnlyIndicator	Indicates whether the parameter is read-only: • If the value is `true`, the user is not allowed to change the value. • If the value is `false`, the user can change the value when handling the item. This element is required for each parameter.
Parameter/Restriction	Restriction to the data type. This element can contain the name of a code as defined in ION Desk. The values that the user can specify are then restricted to the specified code.
TreeParameter	The TreeParameter consists of a TreeDefinition and TreeNodes specified after the TreeDefinition. The TreeDefinition defines a complex data structure. The TreeNodes define the data value for the structure. The TreeDefinition and the TreeNodes must be consistent.

Element	Note
TreeParameter/Sequence	Indicates the sorting sequence in which TreeParameters and Parameters must be displayed to the user in the task. If one Sequence is defined, all Parameters must specify a Sequence element.
TreeParameter/TreeDefinition	Definition for the data structure contained in the TreeParameter. Definition includes several unique TreeNodes with their ID, ParentID, NodeName, and TreeNodeParameters.
TreeParameter/TreeDefinition/TreeNode	A node in a tree using a parent relationship and containing the definition of the properties required for the node.
TreeParameter/TreeDefinition/TreeNode/Sequence	Indicates the sequence of the node within the TreeParameter. You must specify the Sequence for all TreeNodes if this element is specified for the root TreeNode. The ordering of the TreeNodes within the structure is based on the specified Sequence. If the Sequence is not specified for the root TreeNode, the structure is sorted based on the xml.
TreeParameter/TreeDefinition/TreeNode/ID	The unique identifier for this tree node within this definition of a TreeParameter.
TreeParameter/TreeDefinition/TreeNode/ParentID	The ID of the node that is the parent of this node within the tree. The root TreeNode does not have a ParentID.
TreeParameter/TreeDefinition/TreeNode/NodeName	This is the unique identification of a tree node.
TreeParameter/TreeDefinition/TreeNode/Label	The label of the parameter that is used when showing the parameter to a user. You must specify at least one label. You can specify several labels, each with a different languageID for translated labels.
TreeParameter/TreeDefinition/TreeNode/Label /@languageID	The language code of the Label field. For details, see notes from Supported Features.
TreeParameter/TreeDefinition/TreeNode /TreeNodeParameter	The TreeNodeParameter defines a property within a TreeNode level. TreeNodeParameters are optional. Values for these properties are specified in the TreeNodes that follow the TreeDefinition.
TreeParameter/TreeDefinition/TreeNode /TreeNodeParameter/Sequence	Indicates the sequence of the TreeNodeParameter within the TreeNode.
TreeParameter/TreeDefinition/TreeNode /TreeNodeParameter/Name	The name of the TreeNodeParameter is used to identify this property in the tree node instances to specify its value.

Element	Note
TreeParameter/TreeDefinition/TreeNode/TreeNodeParameter/DataType	The data type of the associated value. You can use these data types: • STRING - a string value that is up to 4000 characters in length • INTEGER - a numeric type that represents a whole number • DECIMAL - a numeric type that has a floating precision. Values may be expressed using the scientific e-notation. For details about the scientific e-notation, see Wikipedia or other resources on the internet. • BOOLEAN - represents a true or false value • DATETIME - the date part and time part of a date/time stamp separated by "T" and ending with Z (is always UTC) • TIME - the time part of a date/time stamp • DATE - the date part of a date/time stamp
TreeParameter/TreeDefinition/TreeNode /TreeNodeParameter/Label	The label of the node that is used when displaying it to a user. You must specify at least one label. You can specify several labels, each with a different languageID for translated labels.
TreeParameter/TreeDefinition/TreeNode /TreeNodeParameter/Label/@languageID	The language code of the Label field. For details, see notes from Supported Features.
TreeParameter/TreeNode	A node in a tree using a parent relationship and containing properties required for the node and their values.
TreeParameter/TreeNode/ID	The unique identifier for this tree node within this TreeParameter.
TreeParameter/TreeNode/ParentID	The ID of the node that is the parent of this node within the tree definition.
TreeParameter/TreeNode/NodeName	The unique identification of a tree node. This name must match a node name from the TreeDefinition. Several TreeNode instances with the same NodeName can exist.
TreeParameter/TreeNode/TreeNodeParameter	List of properties for this TreeNode.
TreeParameter/TreeNode/TreeNodeParameter/Name	The property name that must match with a property defined for this TreeNode in the TreeDefinition.
TreeParameter/TreeNode/TreeNodeParameter/Value	The value for this tree node property. The value must be consistent with the data type specified in the tree definition.

Element	Note

ActionParameter	Parameter that holds the actions that a user can do when closing the task.
	For example:
	`<ActionParameter>`
	`  <Name>ApprovalResult</Name>`
	`  <Value>Rejected</Value>`
	`  <Action sequence="1">`
	`    <Value>Approved</Value>`
	`    <Label>Approve</Label>`
	`  </Action>`
	`  <Action sequence="2">`
	`    <Value>Rejected</Value>`
	`    <Label>Reject</Label>`
	`  </Action>`
	`</ActionParameter>`
	An action parameter must have at least one action.

ActionParameter/Name	The name that identifies the action parameter within the task. This element is required for each action parameter.
ActionParameter/Value	The (serialized) value of the action parameter.
ActionParameter/Action/Value	The value that is assigned to the action parameter when the user selects this action.
ActionParameter/Action/Label	The label used when displaying the action button to the user. This element is required for each action parameter. Even though it is technically possible to use a string of maximum length of 255 characters, we recommend that you use short labels that are suitable for action buttons.
ActionParameter/Action/Label /@languageID	The language code of the Label field. For details, see notes from Supported Features.
AssignedPerson	The user to which the notification is currently assigned. A notification is initially distributed to all users having a DistributionPerson defined in the notification.
AssignedPerson/PersonReference	Reference to a person.
AssignedPerson/PersonReference/IDs/ID	Identifier of the person. The ID must be a Person ID or Contact ID as defined in IFS (case-sensitive).
AssignedPerson/PersonReference/Name	Name of the person.
AssignedPerson/PersonReference/SystemUserIndicator	Value is `true` if the person is a User in IFS. Value is `false` if the person is a Contact in IFS.

Element	Note
DistributionPerson	Person in the distribution list for the task. The task is distributed to all users for which a DistributionPerson is included.
DistributionPerson/ID	Identification of the distribution person within the distribution list of the task.
DistributionPerson/PersonReference	Reference to a person.
DistributionPerson/PersonReference/IDs/ID	Identifier of the person. The ID must be a Person ID or Contact ID as defined in IFS (case-sensitive). This element is required for each distribution person.
DistributionPerson/PersonReference/Name	Name of the person.
DistributionPerson/PersonReference/SystemUserIndicator	Value is `true` if the person is a User in IFS. Value is `false` if the person is a Contact in IFS. This element is required for each distribution person. For PulseTask, the value must always be `true`.
DistributionGroup	Specify one or more distribution groups to which the task must be distributed. This element can be used in addition to the DistributionPerson element, or instead of the DistributionPerson element.
DistributionGroup/Name	Identifier of the distribution group as defined in Infor Ming.le User Management. The task is distributed to all users that are members of this group at the time the task is created.
DistributionGroup/Description	Description of the distribution group. This element is optional and is not used to determine the distribution list.
DistributionGroup/Description/@languageID	Describe the language code of the description element. This attribute is optional and is not used for the distribution functionality.

Note: The DistributionGroup element is supported only in the ProcessPulseTask BOD. A SyncPulseTask BOD can be sent after the creation of the task. In that case, the distribution list of the task is described using a DistributionPerson element for each user from the distribution group.

PulseNotification

This table shows the elements in PulseNotification:

Element	Note
DocumentID/ID	Unique identification of the notification.
CreationDateTime	Date when the notification was created in Pulse.
LastModificationDateTime	Date when the notification was last modified.
Status/Code	Status of the notification. Values are: • ASSIGNED - The initial status for a new item. The notification is assigned to each of the distribution persons. • DONE - Completed. • CANCELLED - canceled by an administrator (through ION Desk) or by the application (through a ProcessPulseNotification BOD).
Description	The description that is displayed to the end user as the summary of the notification. You can use hash tags to make searching easier. For example: `#Requisition 25 is approved` To define a category, use `##` at the end of the message. To include values from the alert details, use square brackets around the parameter labels. To use the actual characters for square brackets, use an escape character: `\[` or `\]`
Description/@languageID	The language code of the Description field. For details, see the notes in the "Supported features" section.
Note	Propagated notes that are added by users who worked on Tasks from the same workflow. You cannot add or remove notes from a Notification. This field is only applicable for Sync.PulseNotification for notifications created by Workflow.
Note/@userID	The ID (personId) of the person who added the note.
Note/@author	Full name of the person who added the note. This attribute is required when a Note is added through a ProcessPulseAlert BOD." / "ProcessPulseTask BOD." / "ProcessPulseNotification BOD.
Note/@entryDateTime	The date/time at which the note was created.
Note/@noteID	Identification of the note within the notification.

Element	Note
Source/Type	BOD: the notification was created by sending a ProcessPulseNotification BOD. WORKFLOW: the notification was created by a workflow.
Source/Name	If Type=BOD: the logical ID of the sender of the ProcessPulseNotification BOD. For example, `lid://infor.erp.myerp` If Type is WORKFLOW: the name of the workflow definition that created the notification. For example, `MyWorkflow`.

Parameter	Data of the notification that is displayed to the user. Notification parameters are always read-only.
	For example:
	`<Parameter sequence="1">`
	`  <Name>OrderNumber</Name>`
	`  <Value>12345</Value>`
	`  <DataType`
	`      listID="Pulse Datatypes">STRING</DataType>`
	`  <Label>Order Number</Label>`
	`</Parameter>`

Parameter/@sequence	The sequence number of the parameter. This is used for ordering the parameters when displaying them to a user. It is only allowed to use this attribute if there are no elements of type TreeParameter in the document.
Parameter/Sequence	You must specify this element in combination with TreeParameter/Sequence. This element is used to determine the order in which Parameters and TreeParameters are displayed to the user.
Parameter/Name	The name that identifies the parameter within the notification. This element is required for each parameter.
Parameter/Value	The (serialized) value of the parameter. The formatting depends on the DataType and is the same as the formatting that is normally used in BOD data elements.

Element	Note
Parameter/DataType	The data type of the parameter. This element is required for each parameter. You can use these data types: • STRING - a string value that is up to 4000 characters in length • INTEGER - a numeric type that represents a whole number • DECIMAL - a numeric type that has a floating precision. Values may be expressed using the scientific e-notation. For details about the scientific e-notation, see Wikipedia or other resources on the internet. • BOOLEAN - represents a true or false value • DATETIME - the date part and time part of a date/time stamp separated by "T" and ending with Z (is always UTC) • TIME - the time part of a date/time stamp • DATE - the date part of a date/time stamp
Parameter/Label	The label of the parameter that is used when displaying the parameter to a user. This element is required for each parameter.
Parameter/Label /@languageID	The language code of the Label field. For details, see the notes in the "Supported features" section.
TreeParameter	The TreeParameter consists of a TreeDefinition and TreeNodes specified after the TreeDefinition. The TreeDefinition defines a complex data structure. The TreeNodes specify the data value for the structure. The TreeDefinition and the TreeNodes must be consistent.
TreeParameter/Sequence	Indicates the sorting sequence in which TreeParameters and Parameters must be displayed to the user in the notification. If one Sequence is specified, all the Parameters must specify a Sequence element.
TreeParameter/TreeDefinition	Definition for the data structure contained in the TreeParameter. This definition includes several unique TreeNodes with their ID, ParentID, NodeName, and TreeNodeParameters.
TreeParameter/TreeDefinition/TreeNode	A node in a tree using a parent relationship and containing the definition of the properties required for the node.

Element	Note
TreeParameter/TreeDefinition/TreeNode/Sequence	Indicates the sequence of the node within the TreeParameter. The Sequence must be specified for all the TreeNodes if this element is specified for the root TreeNode. The ordering of the TreeNodes within the structure is based on the specified Sequence. If the Sequence is not specified for the root TreeNode, the structure is sorted based on the xml.
TreeParameter/TreeDefinition/TreeNode/ID	The unique identifier for this tree node within this definition of a TreeParameter.
TreeParameter/TreeDefinition/TreeNode/ParentID	The ID of the node that is the parent of this node within the tree. The root TreeNode does not have a ParentID.
TreeParameter/TreeDefinition/TreeNode/NodeName	This is the unique identification of a tree node.
TreeParameter/TreeDefinition/TreeNode/Label	The label of the parameter that is used when displaying the parameter to a user. You must specify at least one label. You can specify several labels, each with a different languageID for translated labels.
TreeParameter/TreeDefinition/TreeNode/Label /@languageID	The language code of the Label field. For details, see the notes in the "Supported features" section.
TreeParameter/TreeDefinition/TreeNode /TreeNodeParameter	Defines a property within a TreeNode level. TreeNodeParameters are optional. Values for these properties are specified in the TreeNodes that follow the TreeDefinition.
TreeParameter/TreeDefinition/TreeNode /TreeNodeParameter/Sequence	Indicates the sequence of the TreeNodeParameter within the TreeNode.
TreeParameter/TreeDefinition/TreeNode /TreeNodeParameter/Name	The name of the TreeNodeParameter is used to identify this property in the tree node instances to specify its value.

Element	Note
TreeParameter/TreeDefinition/TreeNode/TreeNodeParameter/DataType	The data type of the associated value. You can use these data types: • STRING - a string value that is up to 4000 characters in length • INTEGER - a numeric type that represents a whole number • DECIMAL - a numeric type that has a floating precision. Values may be expressed using the scientific e-notation. For details about the scientific e-notation, see Wikipedia or other resources on the internet. • BOOLEAN - represents a true or false value • DATETIME - the date part and time part of a date/time stamp separated by "T" and ending with Z (is always UTC) • TIME - the time part of a date/time stamp • DATE - the date part of a date/time stamp
TreeParameter/TreeDefinition/TreeNode /TreeNodeParameter/Label	The label of the node that is used when it is displayed to a user. You must specify at least one label. You can specify several labels, each with a different languageID for translated labels.
TreeParameter/TreeDefinition/TreeNode /TreeNodeParameter/Label/@languageID	The language code of the Label field. For details, see the notes in the "Supported features" section.
TreeParameter/TreeNode	A node in a tree using a parent relationship and containing properties required for the node and their values.
TreeParameter/TreeNode/ID	The unique identifier for this tree node within this TreeParameter.
TreeParameter/TreeNode/ParentID	The ID of the node that is the parent of this node within the tree definition.
TreeParameter/TreeNode/NodeName	The unique identification of a tree node. This name must match a node name from the TreeDefinition. Several TreeNode instances with the same NodeName can exist.
TreeParameter/TreeNode/TreeNodeParameter	List of properties for this TreeNode.
TreeParameter/TreeNode/TreeNodeParameter/Name	The property name that must match with a property specified for this TreeNode in the TreeDefinition.
TreeParameter/TreeNode/TreeNodeParameter/Value	The value for this tree node property. The value must be consistent with the data type defined in the tree definition.

Element	Note
AssignedPerson	The user to which the notification is currently assigned. When a user closes the notification, he or she is removed from the list of assigned persons.
AssignedPerson/PersonReference	Reference to a person.
AssignedPerson/PersonReference/IDs/ID	Identifier of the person. The ID must be a Person ID or Contact ID as defined in IFS (case-sensitive).
AssignedPerson/PersonReference/Name	Name of the person.
AssignedPerson/PersonReference/SystemUserIndicator	Value is `true` if the person is a User in IFS. Value is `false` if the person is a Contact in IFS.
DistributionPerson	Person in the distribution list for the notification. The notification is send to each of the distribution persons in parallel.
DistributionPerson/ID	Identification of the distribution person within the notification.
DistributionPerson/PersonReference	Reference to a person.
DistributionPerson/PersonReference/IDs/ID	Identifier of the person. The ID must be a Person ID or Contact ID as defined in IFS (case-sensitive). This element is required for each distribution person.
DistributionPerson/PersonReference/Name	Name of the person.
DistributionPerson/PersonReference/SystemUserIndicator	Value is `true` if the person is a User in IFS. Value is `false` if the person is a Contact in IFS. This element is required for each distribution person.
DistributionGroup	Specify one or more distribution groups to which the notification must be distributed. This element can be used in addition to the DistributionPerson element, or instead of the DistributionPerson element.
DistributionGroup/Name	Identifier of the distribution group as defined in Infor Ming.le User Management. The notification is distributed to all users that are members of this group at the time the notification is created.
DistributionGroup/Description	Description of the distribution group. This element is optional and is not used to determine the distribution list.
DistributionGroup/Description/@languageID	Describe the language code of the description element. This attribute is optional and is not used for the distribution functionality.

Note: The DistributionGroup element is supported only in the ProcessPulseNotification BOD. A SyncPulseNotification BOD can be sent after the creation of the notification. In that case, the distribution list of the notification is described using a DistributionPerson element for each user from the distribution group.

Supported features

Most features of Pulse are supported when creating an alert, task, or notification through a process BOD. This table shows which features of Pulse are supported when creating an alert, task, or notification through a Process BOD.

Feature	Process PulseAlert	Process PulseTask	Process PulseNotification
Message (Description)	Yes	Yes	Yes
Priority	N/A	Yes	N/A
Defined sequence of data elements	Yes	Yes	Yes
Hierarchy of data	Yes	Yes	Yes
Data labels	Equal to names	Yes	Yes
Read-only values	Yes	Yes	Yes
Editable values	N/A	Yes	N/A
Data-type dependent controls	N/A	Yes	N/A
Action buttons	N/A	Yes	N/A
Distribution to (Pulse) users	Yes	Yes	Yes
Distribution to external contacts (e-mail)	Yes	N/A	Yes
Distribution to groups	Yes	Yes	Yes
Escalation policies	No	No	N/A
Translations	No See note 1.	No See note 1.	No See note 1.
Locales	Yes See note 2.	Yes See note 2.	Yes See note 2.
Hyperlinks	N/A	Yes See note 3.	Yes See note 3.

Feature	Process PulseAlert	Process PulseTask	Process PulseNotification
Document references with drill-back	Yes	N/A	N/A
Attach notes	Yes	Yes	N/A
Add attachments	Yes See note 4.	Yes See note 4.	Yes See note 4.
Start workflow from alert	No	N/A	N/A

Notes:

1 Translations are supported for Description and Label fields, in combination with the languageID attribute. An occurrence of a Description or Label field that does not have the languageID attribute is called the 'default value' and is required. Optionally, you can add additional occurrences of these fields, called 'translations', each having a languageID attribute filled with a language. At runtime, Infor Ming.le matches the display language from the user's regional settings with the language code from the translations and shows the corresponding string. If there is no match, the default value is displayed.

Language codes are the codes used by Microsoft.

See also: ISO 639 http://en.wikipedia.org/wiki/ISO_639

2 Parameter values are displayed in Infor Ming.le using the user's local settings, such as date and time format, time zone, and number format. This depends on the parameter's data type. Data in the Description element is not formatted by ION, but displayed as is. For example when the Description contains: "On 24-12-2012T14:50:40Z the total amount was 1,234.56".

3 For tasks and notifications, you can use a hyperlink as a parameter value. In that case, use data type STRING. A string is presented as a hyperlink to the Infor Ming.le user if it starts with 'http://' or 'https://'.

4 You can use attachments in Infor Ming.le widgets, also for items that are created using a Process BOD. But you cannot include them in a BOD. For tasks and notifications, you can include hyperlinks in the BOD.

Chapter 12: Creating custom metadata

The Data Catalog component is a utility component used by ION Desk.

This section explains the Data Catalog contents and how you can add metadata for your own objects or for extensions on standard application objects.

The Data Catalog contains metadata on objects that are sent through the ION Service. During the installation of ION, the Data Catalog is filled with the latest version of the Infor metadata.

If you only use standard objects from Infor, you do not have to modify the Data Catalog contents. You can use custom application objects in ION Connect, in a monitor, or in a workflow activation policy. Add the metadata for those custom application objects to the Data Catalog.

The functionality to add custom metadata in the Data Catalog is intended only for customer-specific metadata. Infor metadata should not be included as custom metadata.

Data Catalog contents

The Data Catalog contains noun metadata information organized in libraries. A library can contain one or more nouns. Libraries contain standard objects, which can either be global or application-specific. You can use the available noun metadata from all libraries in ION Desk models.

Custom objects can be of type BOD, also referred to as "custom nouns", or of type ANY, DSV, or JSON.

At each library level, each tenant level, and each noun level, a patch number is maintained. Only the latest noun metadata for both standard and custom nouns is stored in the Data Catalog.

For each noun, custom or standard, this information is stored:

- Noun Name
- Noun XSD - the XML schema definition for this noun, flattened
- Other metadata, such as the patch number (a kind of version number), the list of XPaths to key fields in the noun, the list of verbs supported by this noun, and the relationships between this noun and other nouns.

For custom objects of type ANY, only the object name is stored.

For custom objects of type JSON, the JSON schema definition for each object name is stored.

For custom objects of type DSV, the DSV schema definition for each object name is stored.

Before customizing the Data Catalog

Before you customize the Data Catalog, note these points:

- Contact Infor for information on services to assist you when developing custom objects.
- The Data Catalog contains these files:
  - XML Schema (XSD) files and XML files to describe objects of type BOD
  - JSON Schema files to describe objects of type JSON and DSV (delimiter-separated values)

  To customize the Data Catalog metadata, you must understand and adapt those files.
- If you add files to the Data Catalog, you can use customized nouns in ION Desk only. To use the customized nouns in ION Service, these nouns must be adopted by Infor applications or other applications. The definitions in the Data Catalog must always match the objects that are sent or received by applications that are connected to ION.
- In the Data Catalog XSD files, conventions are used. These are the most important conventions:
  - All elements use the type-attribute and therefore refer to named types.
  - All types, complex and simple, are explicit and have "Type" as a post fix. For example, `Status Type`.
  - The elements in a complexType containing a sequence or choice are always a `ref` to a single element with a named type.
- **Note:**
  - In customized XSDs, do not use unions such as `<xs:union memberTypes="xs:string xs:dateTime"/>` or binary data types such as `xs:hexBinary`.
  - If you create a customized version of a standard noun, it must be compatible with the standard. If you use a new name for the customized noun, such as MySalesOrder, compatibility with the standards is not required. When you create a version of your custom noun, it must be backwards compatible with previous versions.

Object Naming Conventions

Objects of all types are registered by name.

These constraints apply:

- The object name must be unique across all custom objects types and standard nouns and application specific nouns. For example, if the `InforOagis` library is imported, a custom object with the name `Sync.SalesOrder` must not exist.
- The object name is case-preserving, but case-insensitive. If 'MyDocument' exists, you cannot create 'MyDOCUMENT' except by overwriting the existing one.
- The maximum length for an object name is 100 characters.
- The object names for the metadata files cannot exceed 255 characters. This is including the path in the export file, for example: `JSON/myObject/myObject.schema.json`

Custom objects of type ANY

You can use objects of type ANY in ION File Connector and IMS Connector.

A custom object of type ANY is only defined by its name.

An object name may contain these characters:

- Any standard letter in any language
- Numbers 0-9
- Underscore (_)
- Hyphen (-)
- Period (.)

Defining a custom object of type ANY

1. To create an object of type ANY, prepare this folder structure for the import:

   FOLDER: `ANY`

   +-- FOLDER: *`<object name>`*

   +---- FILE: *`<object name>`*`.xml`

   The *`<object name>`*`.xml` file is empty for objects of type ANY.

2. To import a ZIP file containing one or more definitions for objects of type ANY, use the **Custom Objects** page in ION Desk.

   See the *Infor ION Desk User Guide*.

Custom objects of type JSON

You can use objects of type JSON in ION File Connector, IMS Connector, and ION AnySQL Connector.

A custom object of type JSON is defined by its name and a JSON schema file.

An object name may contain these characters:

- Any standard letter in any language
- Numbers 0-9
- Underscore (_)
- Hyphen (-)
- Period (.)

Defining a custom object of type JSON

1 To create an object of type JSON, prepare this folder structure for the import:

FOLDER: `JSON`

+-- FOLDER: *`<object name>`*

+---- FILE: *`<object name>`*`.schema.json`

+---- FILE: *`<object name>`*`.properties.json`

2 To import a ZIP file containing one or more definitions for objects of type JSON, use the **Custom Objects** page in ION Desk.

See the *Infor ION Desk User Guide*.

These validations are performed upon import:

- The object name must be unique and must comply to the object naming restrictions mentioned above.
- If an object with the same name is already registered with another type, the import fails.
- If an object with the same name is already registered with type JSON, the imported schema overwrites the existing schema.
- The *`<object name>`*`.schema.json` schema file is validated as follows:
  - There must be only one JSON schema file for a given object name.
  - The schema file must be valid JSON according to JSON schema definition, version draft-06.
  - The schema file must use UTF-8 encoding.
  - The “anyOf” keyword is not allowed.
  - The “oneOf” keyword is supported only to describe a type that can be null.
- The *`<object name>`*`.properties.json` file is optional and can contain additional metadata properties for the object.

  For information on how to define this file, see Defining additional object metadata properties on page 101 and Additional properties file on page 101.

Newline-delimited JSON

If your data object uses newline-delimited JSON, you must specify that in your object schema.

1 Specify the schema for a single object.

All objects in a single data object must have the same schema.

2 Include a property called `x-stream` and set it to true.

This code is a newline-delimited JSON schema example:

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
      "x-stream":true,
      "type": "object",
      "properties": {
        "field1": {
          "type": "integer"
        },
        "field2": {
          "type": "string"
        }
    }
```

This code shows the JSON data that corresponds with the schema example:

```
{"field1":123,"field2":"Some text"}
{"field1":456,"field2":"Another text"}
{"field1":789,"field2":"More text to be added"}
```

Validations that are performed on import for newline-delimited JSON follow the same rules as conventional JSON.

Custom objects of type DSV

You can use objects of type DSV in ION File Connector and IMS Connector. DSV stands for Delimiter-Separated Values.

When you register a DSV object schema in the Data Catalog, it is assigned a subtype. Based on the separator value that is provided in the schema, one of these subtypes is assigned:

- CSV - comma-separated values
- TSV - tab-separated values
- PSV - pipe-separated values
- Other

A custom object of type DSV is defined by its name and a schema file.

An object name may contain these characters:

- Any standard letter in any language
- Numbers 0-9
- Underscore (_)
- Hyphen (-)
- Period (.)

Defining a custom object of type DSV

1 To create an object of type DSV, prepare this folder structure for the import:

FOLDER: `DSV`

+--FOLDER: *`<object name>`*

+----FILE: *`<object name>`*`.schema.json`

+----FILE: `<object name>.properties.json`

2 These validations are performed upon import:

- The object name must be unique and must comply to the object naming restrictions mentioned above.
- If an object with the same name is already registered with another type, the import fails.
- If an object with the same name is already registered with type DSV, the imported schema overwrites the existing schema.
- The `<object name>.schema.json` schema file is validated as follows:
  - There must be only one DSV schema for a given object name.
  - The schema file must use UTF-8 encoding.
  - The schema file must contain a “dialect” property. This property defines the format of the delimited data object.

    The dialect property must contain a “separator” property, which defines the character that separates the values in a row within the data object.
  - The “properties” element in the schema file should list the fields, in order, that are expected to be found in the DSV data object. This section of the schema must be valid JSON according to schema definition, version draft-06.
    - The “anyOf” keyword is not allowed.
    - The “oneOf” keyword is supported only to describe a type that can be null.
- The `<object name>.properties.json` file is optional and can contain additional metadata properties for the object.

  For information on how to define this file, see Defining additional object metadata properties on page 101 and Additional properties file on page 101.

This code is an example of a DSV schema definition:

```
{
   "title": "myDelimitedFile",
   "description": "DSV Schema for myDelimitedFile",
   "dialect": {
      "separator": ",",
      "skipLines": 1,
      "enclosingCharacter": "\""
   },
   "properties": {
      "Code": {
         "description": "Customer code",
         "type": "string",
         "maxLength": 10
      },
      "Customer": {
         "description": "Customer name",
         "type": "string",
         "maxLength": 100
      },
      "PubDatetime": {
         "description": "Publication timestamp",
         "type": "string",
         "format": "date"
```

```
            }
        }
    }
```

Dialect properties for DSV objects

This table shows the dialect properties that are supported for DSV objects.

Dialect property	Type	Description
`separator`	string	The delimiter character that separates the values in a row of data. The value of the separator should be defined as follows for the different delimited file types: • Comma-separated (CSV): `"separator":","` • Tab-separated (TSV): `"separator":"\t"` • Pipe-separated (PSV): `"separator":"|"` Other separator values are allowed, but these must be single-character separators. For example, double pipes (`||`) are not supported. Required
`skipLines`	integer	Indicates the number of header rows to skip over at the top of the file before reaching the actual data. Optional
`enclosingCharacter`	string	The character that identifies the start and end of a value. If two consecutive enclosing characters are found in a data object, they are interpreted as one, therefore escaping the enclosing character. Optional

The line separator is not specified in the metadata.

These characters are regarded as the end of a line, unless they are placed within enclosing characters:

- A carriage return
- A line feed
- The combination of carriage return and line feed

The last line may or may not have a line separator.

The encoding is not specified in the metadata. It is always assumed to be UTF-8.

Using datetime formats

When dates and times are included in JSON and DSV data objects, you should define them as such in the object metadata. This makes it clear to applications that use the metadata that the values should be interpreted as dates or times. Infor applications that use dates and times must follow the ISO 8601 RFC 3339 standard formats.

JSON Schema draft-06 supports these date and time formats for string instances:

- date-time
- date
- time

When you include dates and times in your objects, it is best practice to use “date-time”, in UTC, wherever possible. Dates without a time and times without a date regularly occur in application data. Therefore, you can also use the "date" or "time" formats alone.

Standard format definitions

This table shows the standard format definitions:

Metadata definition	Expected format in data object	Example data
`“datetimeProperty”:{ “type”:”string”, “format”:”date-time” }`	yyyy-MM-ddTHH:mm:ss[.S]Z	"2017-07-04T14:08:43Z" Or "2017-07-04T14:08:43.123Z"
`“dateProperty”:{ “type”:”string”, “format”:”date” }`	yyyy-MM-dd	“2017-07-04”
`“timeProperty”:{ “type”:”string”, “format”:”time” }` This format is not supported by Data Lake services when processing data. Times without dates are treated as string values.	HH:mm:ss[.S]	“14:08:43” Or “14:08:43.123”

Custom datetime formats

To define a datetime format that is not a standard supported by JSON schema draft-06, you can use the “x-dateTimeFormat” custom property in your object metadata. This enables the interpretation of data from third-party applications that do not adhere to the ISO 8601 RFC 3339 standard.

Custom format definitions

The tables in this section show the custom datetime formats that are currently supported. Any other formats specified are not recognized by applications that use the metadata. All dates that are included in a data object are assumed to be in UTC. The value that is provided for “x-dateTimeFormat” is case-sensitive. Therefore, you must include this value in your metadata exactly as it is defined below.

This table shows the metadata definitions:

Metadata definition	Description

`“epochMillisProperty”:{`
` “type”:”integer”,`
` “x-dateTimeFormat”:”epoch-millis”`
`}`
	Datetime in epoch milliseconds for integer instances.
	Example: 1537204639000

`“americanDateTimeProperty”:{`
` “type”:”string”,`
` “x-dateTimeFormat”:”M/d/yyyy`
`h:mm:ss[.S] a”`
`}`
	American datetime format for string instances.
	Examples:
	“10/9/2017 12:42:01 PM”
	or
	“10/9/2017 12:42:01.139 PM”

`“americanDateProperty”:{`
` “type”:”string”,`
` “x-dateTimeFormat”:”M/d/yyyy”`
`}`
	American date format for string instances.
	Example: “6/23/2018”

`“integerDateProperty”:{`
` “type”:”integer”,`
` “x-dateTimeFormat”:”yyyyMMdd”`
`}`
	8-digit integer date for integer instances.
	Example: 20180601

`“threeCharMonthProperty”:{`
` “type”:”string”,`
` “x-dateTimeFormat”:”dd-MMM-yyyy`
`HH:mm:ss”`
`}`
	Three-character month for string instances, Oracle format. This format always translates to a US locale.
	Example: “01-SEP-2018 14:08:23”

This table shows the tokens for datetime formats:

Token	Description
M	Month, one or two digits: 1-12

Token	Description
MM	Month, two digits: 01-12
MMM	3-character month: JAN, FEB, MAR, etc.
D	Day of the month, one or two digits: 1-31
dd	Day of the month, two digits: 01-31
yyyy	Year, four digits: 0001-9999
HH	Hours, 24-hour, two digits: 00-23
h	Hours, 12-hour, one or two digits: 1-12
mm	Minutes, two digits: 00-59
ss	Seconds, two digits: 00-59
[S]	Optional fractional seconds, up to nine digits: 0-999999999
a	AM/PM designator when using 12-hour format: "AM", "am", "PM", or "pm"

Schema Property Order

Standard JSON Schema does not support specifying an order to the properties defined in a schema. If the fields in your JSON or DSV data objects must be displayed in a specific order, you can use the custom property “x-position” to define what that order must be. The value for "x-position" must be an integer.

Example Schema:

```
{
   "$schema":"http://json-schema.org/draft-06/schema#",
   "title":"Schema Property Order",
   "description":"Sample schema to include property order",
   "type":"object",
   "properties":{
       "divisionName":{
            "type":"string",
            "maxLength":16,
        "x-position":2
         },
         "divisionId":{
            "type":"string",
            "maxLength":250,
        "x-position":1
         },
         "updateDateTime":{
            "type":"string",
            "format":"date-time"
         },
         "variationNumber":{
            "type":"integer"
         },
         "companyId":{
            "type":"string",
            "x-position":3
         }

   }
```

Since “x-position” is optional, it is not required to include it for all properties in the schema. It is up to the applications using this metadata to determine how to handle the display of fields that do not include an “x-position” value.

Defining additional object metadata properties

When you import a schema, you can register additional metadata properties for objects of type JSON and DSV with the Data Catalog.

To register these additional metadata properties, you require an object properties file.

This is an optional file that contains extra properties that are not defined in the object schema. The file name must match the object name and have the `.properties.json` extension.

You can include these additional properties when you import the object schema in the Data Catalog UI. For information on the file structure for including additional metadata properties, see the "Zip file structure for import" section in the *Infor ION Desk User Guide*.

Additional properties file

Validation

The properties file is validated using this schema:

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "title": "Additional Object Metadata",
  "description": "Additional Object Metadata",
  "type": "object",
  "properties": {
    "IdentifierPaths": {
      "type": "array",
      "items": {
        "type": "string"
      }
    },
    "VariationPath": {
      "type": "string"
    },
    "TimestampPath": {
      "type": "string"
    },
    "DeleteIndicator": {
      "type": "object",
      "properties": {
        "path": {
          "type": "string"
        },
        "value": {
          "type":["string","boolean","number"]
        }
      },
      "additionalProperties":false,
      "required":["path","value"]
    },
    "AdditionalProperties": {
      "type": "object",
      "additionalProperties": true
    }
  },
  "additionalProperties": false
}
```

Properties

This table shows each of the properties that may be included in the file. All properties are optional.

Property	Description
`IdentifierPaths`	The set of properties of the data object that identify the object. This applies if the data object consists of a single JSON or DSV object. Alternatively, the set of properties of the data objects. This applies if the data object holds a newline-delimited list or an array of objects. Each property in the array must be a JSON path to the property that is the identifier or part of the identifier.
`VariationPath`	The path to a property that indicates a variation number or variation string. This must be a JSON path. If this property is available, it can be used for these purposes: • To determine the sequence in which multiple changes on a single object took place • To find the latest version of an object

Property	Description
`TimestampPath`	The path to a property that indicates the moment the object was last updated or created in the application that owns the data. This must be a JSON path. The property it refers to must have a date-time format. If this property is available, it can be used to determine the order in which changes on a set of objects from the same application took place. This property is less accurate than `VariationPath`, but can be used across objects.
`AdditionalProperties`	This area can be used for additional properties that are not owned or prescribed by the Data Catalog. Any custom properties for appli-cations must be added here. **Note:** When you add additional properties, ensure that the name of each property is unique. We recommend that you include the application logical ID as a prefix to the property name.
`DeleteIndicator`	The path to the property that indicates whether the object is deleted, or marked as deleted, and the property value to indicate that. The property must be a boolean, number, or string, and the specified value must match that data type. When you use `DeleteIndicator`, these properties are required: • path - The path to a property that holds the delete indicator for the object. This must be a JSON path. • value - The value of the delete indicator property to determine that the object is deleted. This value can be a string, boolean, or number. The value must match the data type of the property that “path” is pointing to.

Example

This code shows an example of an object’s properties file:

```
{
   "IdentifierPaths":[
      "$.*.myIdProperty",
      "$.*.mySecondIdProperty"
   ],
   "VariationPath":"$.*.myVariationId",
   "TimestampPath":"$.*.myTimestampProperty",
   "DeleteIndicator":{
      "path":"$.*.myDeleteIndicatorProperty",
      "value":"deleted"
   },
   "AdditionalProperties":{
      "infor_ies_searchPath":{...},
      "infor_ies_indexDefinition":{...},
      "acme_myCustomThingies":{...}
   }
}
```

Defining a custom noun

You can add the metadata for your own nouns to the Data Catalog.

Customer-defined nouns do not have to fully, adopt all practices as used in standard nouns, such as the way object IDs or references are formatted. They must meet these conditions:

- The noun content is in XML.
- The noun has an identifying attribute.
- The noun's envelope is a BOD.

Nouns are described by XML Schema (XSD) files. The Infor-delivered nouns are the officially published schemas. Customers can create their own XMLSchema for their nouns.

The name of a custom noun must be unique compared to the standard noun names that are already uploaded in the Data Catalog. We recommend that you always precede Custom nouns with "My. " For example, `MyMaterialRelease` or `MyShippingSchedule`. Using "My "avoids naming conflicts with standard Infor nouns.

The name of a custom noun may contain:

- Any standard letter in any language
- Numbers 0-9
- Underscore

To add a custom noun to the Data Catalog, you must prepare an archive file containing the metadata. You can upload several custom nouns in one archive at the same time. For each custom noun, a noun data-set composed of two files must be present and must contain the noun name in the title:

- `<nounname>.xml`
- `<nounname>.xsd`

These additional validations are performed on the archive file upon import by ION Desk:

- The archive may only contain valid XSD and valid XML files.
- The noun name of the XSD file must be the same as the noun name used inside the XSD file.
- The custom noun definition may only be defined in one XSD file.
- The XML files that defined the metadata of the custom noun must refer to this namespace:

  `xmlns="http://schema.infor.com/CloudRegistry/1"`

Upon upload of a custom noun, ION Desk assigns this noun a patch number, depending on the first available number for the current tenant. When the same custom noun is uploaded again, the patch number increases. ION Desk does not compare the noun definition contents of the new noun and the noun that already exists in the Data Catalog.

To create the metadata for a custom noun:

1 Define the custom noun schema file (XSD) and verify that this is a valid schema. Save this file with the name `<nounname>.xsd`.

Complete these steps:

a Copy the XMLSchema text below into a file with the name of your noun with a "`.xsd`" extension. For example: `MyMaterial.xsd`.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://schema.infor.com/InforOAGIS/2"
xmlns="http://schema.infor.com/InforOAGIS/2"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qual
ified" attributeFormDefault="unqualified" version="1.0.0">
   <xs:element name="MySampleDocument" type="MySampleDocumentType"/>

   <xs:complexType name="MySampleDocumentType">
      <xs:sequence>
         <xs:element name="SampleHeader" type="SampleHeaderType"
minOccurs="1"/>
         <xs:element name="SampleLine" type="SampleLineType"
minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="SampleHeaderType">
      <xs:sequence>
         <xs:element name="DocumentID" type="xs:normalizedString"
minOccurs="0"/>
         <xs:element name="DocumentDateTime" type="xs:dateTime"
minOccurs="0"/>
         <xs:element name="Status" type="xs:string" minOccurs="0"/>

         <xs:element name="Description" type="xs:string" minOc
curs="0"/>
         <xs:element name="ShipmentID" type="xs:normalizedString"
minOccurs="0"/>
         <xs:element name="IsActive" type="xs:boolean" minOc
curs="0"/>
       </xs:sequence>
   </xs:complexType>
   <xs:complexType name="SampleLineType">
      <xs:sequence>
         <xs:element name="LineNumber" type="xs:integer"/>
        <xs:element name="Amount" type="xs:decimal" minOccurs="0"/>

         <xs:element name="Note" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
</xs:schema>
```

b Open the file in an editor and replace all occurrences of "`MySampleDocument` " with "`MyMaterial`".

c Adapt the file as required to define the attributes of the custom noun. Do not refer to XSD files from the Standard folder.

2 Define the custom noun metadata properties file with the name `<nounname>.xml`. Create a file named `[NounName].xml`. For example, if your custom noun is called `MyMaterial`, the file must be named `MyMaterial.xml`.

In this file, add an entry to specify the identifier for the custom noun, verbs that are supported, and the relation it has with other nouns. Note the ‘IDXPath’ is mandatory, you can leave the ‘Relation’ element empty if there is no other noun to reference.

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<NounMetadata xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schema.infor.com/InforOAGIS/2">
   <Noun>
      <NounName>MySampleDocument</NounName>
      <NounType>Transactional</NounType>
     <IDXPath>/*/DataArea/MySampleDocument/SampleHeader/DocumentID</IDX
Path>
      <DescriptionXPath>/*/DataArea/MySampleDocument/SampleHeader/De
scription</DescriptionXPath>
     <StatusXPath>/*/DataArea/MySampleDocument/SampleHeader/Status</Sta
tusXPath>
<DocumentDateTimeXPath>/*/DataArea/MySampleDocument/SampleHeader/Docu
mentDateTime</DocumentDateTimeXPath>
      <SupportedVerbs>
          <SupportedVerb>Acknowledge</SupportedVerb>
          <SupportedVerb>Process</SupportedVerb>
          <SupportedVerb>Sync</SupportedVerb>
      </SupportedVerbs>
   </Noun>
   <Relation type="Transactional">
      <ToNoun>Shipment</ToNoun>
      <Priority>10</Priority>
      <RelationLabel>My sample document linked to shipment</RelationLa
bel>
      <RelationPaths>
         <FromNounPath>/*/DataArea/MySampleDocument/SampleHeader/Ship
mentID</FromNounPath>
          <ToNounPath>/*/DataArea/Shipment/ShipmentHeader/Documen
tID[1]/ID</ToNounPath>
      </RelationPaths>
   </Relation>
</NounMetadata>
```

3 Create a folder for each custom noun and place corresponding xsd schema file and xml file underneath it. The folder name must match the noun name.

4 Gather the custom noun metadata files into one archive file and import this archive into ION Desk. For information about importing custom nouns into ION Desk, see the *Infor ION Desk User Guide*.

Once the custom noun is uploaded to the Data Catalog, it is visible to all users of the current tenant. You can perform these actions using this custom noun:

- Select the application object in a connection point.
- Select the attributes of the object in a filter or content-based routing in an object flow.
- Select the application object and its attributes in a workflow activation policy.
- Select the application object, its references, and its attributes in a monitor.

Important notes:

When using namespaces in your XSDs, the local element name must be unique at any level in a BOD. Two elements having the same name but a different namespace must not exist within the same parent element. So, for example, this object is not supported:

```
<Customer>
  <Address xmlns="namespace1">
    …
  </Address>
  <Address xmlns="namespace2">
    …
  </Address>
</Customer>
```

Customizing an existing noun

Standard nouns contain placeholders for adding customizations: the UserAreas. Usually, UserArea elements are available in multiple locations for a noun. For example, in the header and in the lines.

You can use the UserArea in two ways:

- Using properties in the UserArea. This is the preferred method, because it is supported automatically for any BOD. No changes are required in the Data Catalog.
- Using a custom XML structure in the UserArea. This method is required if the custom data is too complex to fit in the property structure.

Both approaches are explained below.

Using properties in the UserArea

If your custom data can be organized in name-value pairs, you can use the standard 'Property' structure to include this data in the UserArea. In that case customize the application that sends a BOD so that it includes the required properties. Customize the application that receives the BOD so it can handle the data.

This code is an example of a UserArea containing data using the standard Property structure:

```
<UserArea>
   <Property>
      <NameValue name="AcmeCustomNote" type="StringType">My 
note</NameValue>
   </Property>
   <Property>
      <NameValue name="AcmeCustomQuantity" type="Numeric
Type">10.00</NameValue>
   </Property>
</UserArea>
```

Note: To avoid name clashes with properties that may already be used by some applications, we recommend that you use a prefix, such as your company name.

This table shows some of the types that you can use:

Type	Examples of Values	
StringType	This is a string	
NumericType	10.00	
IntegerNumericType	1234	
DateTimeType	2004-12-31T12:32:14.123Z	
IndicatorType	true, false	Two possible values

To use UserArea properties in ION, no changes are required in the Data Catalog. If you use a BOD that has a UserArea, you can select your properties in an event monitor or an activation policy.

Note: You cannot select your properties in ION Connect (content-based routing and filtering).

To select a UserArea property:

1 In the attribute selection window, find the location of the User Area and expand the User Area.

For example, the window can contain this code:

```
Contract
 ContractHeader
  …
  UserArea
   Property
    NameValue
```

2 Select the NameValue attribute.

3 Specify the data type for the attribute.

Normally the data type is retrieved from the Data Catalog but when using a UserArea Property you must specify the data type, because properties can have different data types.

4 For the selected NameValue attribute, define an attribute filter on the `name` to select the specific property you are interested in.

For example:

`Contract/ContractHeader/UserArea/Property/@name = "AcmeCustomQuantity"`

Note: If you use multiple properties from the same user area in an event monitor or workflow activation policy, you must define the filter for the first property before you can select the next property. Otherwise two selected attributes will have the same XPath, which is not accepted in ION Desk.

Using a custom XML structure in the UserArea

If your custom data is too complex to fit in the `standardProperty` structure, you must use a custom XML structure.

By default, the `UserArea` elements in BODs are of type 'AnyType', because they can contain any data: standard properties, your custom structure, or anything else. In these situations, you must define an XSD in the Data Catalog:

- To select your properties in ION Connect, in content-based routing or filtering.
- To select your properties in an event monitor or an activation policy.

After adding your XSD to the Data Catalog as described later, your custom elements are displayed in the attribute selection windows in ION Desk.

For example, assume you must add this custom data in the BOD UserArea:

```
<UserArea>
    <AcmeCustomContacts>
        <Contact>
            <Name>Someone</Name>
            <Email>someone@acme.com</Email>
        </Contact >
        <Contact>
            <Name>Someone Else</Name>
            <Email>someone.else@somewhere.com</Email>
        </Contact >
    </AcmeCustomContacts >
</UserArea>
```

In this case, you must complete these steps:

1 Customize the connection point that sends the BOD, to fill the `UserArea` as required. Customize the connection point that receives the BOD, to handle the data.

2 Create an XML Schema Definition (XSD) file to describe the added XML.

For example, the `AcmeCustomContacts.xsd` file can contain this code:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="AcmeCustomContacts" type="AcmeCustomContact
sType"/>
    <xs:complexType name="AcmeCustomContactsType">
        <xs:sequence>
           <xs:element name="Contact" type="ContactType" maxOccurs="un
bounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ContactType">
        <xs:sequence>
            <xs:element name="Name" type="xs:string"/>
            <xs:element name="Email" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Note: The XSD for a user area extension must have a single top-level element. To add multiple elements, put them under a single parent element.

3. In ION Desk, select **Configure > User Area Extensions > Schema Files** and add the XSD file to the Data Catalog.
4. In ION Desk, select **Configure > User Area Extensions > Extensions**. Map the schema file to the UserArea elements of the nouns that use the User Area Extension.

Note: For details about the **Configure > User Area Extensions** pages, see the *Infor ION Desk User Guide* .

Using an XSD extension for validation

By default, the UserArea is an 'AnyType', because the UserArea can contain any data: standard properties, your custom structure, or anything else. In this default situation, a BOD normally validates successfully against the BOD XSD, independent of the contents of the UserArea. If you defined a UserArea extension XSD in the Data Catalog, to use the XSD for validating your BOD XML, you must ensure the validator can find the XSD extension.

To achieve this:

1. Add namespace information to the XSD.
   For example, the `AcmeCustomContacts.xsd` file can contain this code:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schema.acme.com/userarea"
  targetNamespace=="http://schema.acme.com/userarea"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="AcmeCustomContacts" type="AcmeCustomContact
sType"/>
    <xs:complexType name="AcmeCustomContactsType">
        …
    </xs:complexType>
</xs:schema>
```

2. In the XML that contains the custom data, include a reference to the namespace:

```
<UserArea>
    <acme:AcmeCustomContacts xmlns:acme="http://schema.acme.com/user
area">
        <acme:Contact>
             <acme:Name>Someone</acme:Name>
             <acme:Email>someone@acme.com</acme:Email>
         </acme:Contact >
     </acme:AcmeCustomContacts >
</UserArea>
```

Note: The namespace, in this case "`http://schema.acme.com/userarea`", must be the same as in the XSD.

After completing these steps you can validate the BOD XML against the BOD XSD. In this case, for example, validate the BOD XML against `SyncContract.xsd`.
Note: Validation can help when you set up and test a customization, but reduces performance. Therefore, if possible, avoid validation of all messages sent or received by your application. The ION Service also does not validate the BODs against the XSDs as defined in the Data Catalog.

Chapter 13: Custom message headers

You can define custom message headers for objects that are stored in the Data Catalog. After you have defined custom message headers, they are available to attach to objects that are processed by ION services.

To define custom headers, you can use one of these methods:

1 Direct import to Data Catalog through ION Desk. On the **Object Schemas** page, you can include a custom headers file in the object import ZIP file. This method is useful if you only want to add or update custom headers for an object, but not the object itself. It is also useful when updating custom headers for multiple objects in the Data Catalog.

2 Include custom headers when adding or updating an object using the Data Catalog POST /v1/object API.
Note: This option is not viable for BODs because the API only supports JSON, DSV, and ANY objects.

3 In the Data Catalog application on an object's details page.

For more information on methods 1 and 3, see the “Configuring custom message headers” section in the *Infor ION Desk User Guide*.

Custom headers file format

ZIP file import validation

When you import custom headers in ION Desk, the file is validated using this schema:

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "http://json-schema.org/draft-06/schema#",
  "title": "Custom Headers Schema",
  "type": "object",
  "properties": {
    "customHeaders": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "objectName": {
            "type": "string"
          },
          "headers": {
            "type": "array",
            "maxItems": 3,
            "items": {
              "type": "object",
              "properties": {
                "name": {
                  "type": "string",
                  "pattern": "^Custom_[a-zA-Z0-9]+$",
                  "maxLength": 250
                },
                "dataType": {
                  "enum": [
                    "String",
                    "Integer",
                    "Decimal",
                    "Boolean",
                    "Date",
                    "DateTime"
                  ]
                }
              },
              "required": [
                "name",
                "dataType"
              ]
            }
          }
        },
        "required": [
          "objectName",
          "headers"
        ]
      }
    }
  }
}
```

This is an example of a custom headers file:

```
{
  "customHeaders": [
    {
      "objectName": "Process.PlannedTransfer",
      "headers": [
        {
          "name": "Custom_processItemID",
          "dataType": "String"
        }
      ]
    },
    {
      "objectName": "Sync.PlannedTransfer",
      "headers": [
        {
          "name": "Custom_syncItemID",
          "dataType": "String"
        },
        {
          "name": "Custom_syncUPCID",
          "dataType": "String"
        }
      ]
    },
    {
      "objectName": "Inventory_Repo",
      "headers": [
        {
          "name": "Custom_itemNumber",
          "dataType": "Integer"
        }
      ]
    },
    {
      "objectName": "MITMAS",
      "headers": [
        {
          "name": "Custom_attributeModel",
          "dataType": "String"
        }
      ]
    }
  ]
}
```

Data Catalog API validation

When you register an object with the Data Catalog through the POST /v1/object API, custom headers for that object are validated against this schema:

```
{
  "$schema": "http://json-schema.org/draft-06/schema#",
  "$id": "http://json-schema.org/draft-06/schema#",
  "title": "Custom Headers Schema",
  "type": "object",
  "properties": {
    "customHeader": {
      "type": "object",
      "properties": {
        "objectName": {
          "type": "string"
        },
        "headers": {
          "objectName": {
            "type": "string"
          },
          "headers": {
            "type": "array",
            "maxItems": 3,
            "items": {
              "type": "object",
              "properties": {
                "name": {
                  "type": "string",
                  "pattern": "^Custom_[a-zA-Z0-9]+$",
                  "maxLength": 250
                },
                "dataType": {
                  "enum": [
                    "String",
                    "Integer",
                    "Decimal",
                    "Boolean",
                    "Date",
                    "DateTime"
                  ]
                }
              },
              "required": [
                "name",
                "dataType"
              ]
            }
          },
          "required": [
            "objectName",
            "headers"
          ]
        }
      }
    }
  }
}
```

This is an example of the input body for the API that is used to register an object with custom headers:

```
{
  "name":"mySalesOrder",
  "type":"JSON",
  "schema": {...},
  "properties":{...},
  "customHeader": {
        "objectName": "mySalesOrder",
        "headers": [
            {
                "name": "Custom_salesOrderId",
                "dataType": "Integer"
            },
            {
                "name": "Custom_shipToAddress",
                    "dataType": "String"
               }
          ]
     }
}
```

Properties

This table shows the properties in the file:

Property	Description
customHeaders	An array of JSON objects. Each JSON object contains the name of an object schema and the headers to be defined for that specified object. Optional. **Note:** Applies to the zip file import only.
CustomHeader	A JSON object that contains the object name and the list of headers to be defined for that object. Optional. **Note:** Applies to the POST API only.
objectName	The name of the object for which the custom headers should be defined. For BODs, you must specify the full object name as opposed to only the noun. For example, `Sync.SalesOrder`. Required.

Property		Description
headers		An array of headers to be defined for the object. A maximum of three custom headers may exist per object. Required, but the array can be empty. If an empty array is found, any existing custom headers for the associated object are deleted.
	name	The name of the custom header. The name must begin with `Custom_` and can only contain alphanumeric characters, without any spaces. It cannot exceed 250 characters in length. Required.
	dataType	The data type that the custom header value is expected to have. These are possible data types: • String • Integer • Decimal • Boolean • Date • DateTime **Note:** These are case-sensitive and must match exactly to pass validation. Required.

Chapter 14: Application Programming Interface (API)

You can use several ION APIs.

The available ION APIs are:

- ION OneView
- Alarms
- IMS
- ION Process
- Data Catalog
- Business Rules

For more information on how to use the APIs, see the *Infor ION API Administration Guide*.

ION services such as OneView can only be used through ION API. As an ION user you must be authorized as administrator to see the ION API icon in the Infor Ming.le application menu. Click the ION API icon to find details such as the existing methods.

ION Process API

The ION Process APIs expose functionality that is related to:

- Workflows that run in the Workflow engine.
- Alerts, tasks, and notifications managed by the Pulse engine.

The ION Process APIs are located in the “Infor ION” API Suite and these endpoints:

- `process/application`

  This endpoint exposes methods that are authorized at application level and do not require a user identification. After a client application has access to this endpoint, it can call any method without further security checks being applied.
- `process/user`

  This endpoint exposes methods that are authorized at user level. Internal verification is performed to determine whether the user who calls this API has the permission to perform this call. For example, you cannot close a task that is not assigned to you.

For detailed information about the methods that are exposed in each endpoint, see the Swagger documentation of each endpoint. For more information on how to use ION APIs and how to interact with Swagger documentation for the API methods, see the *Infor ION API Administration Guide*.

Data Catalog API

The Data Catalog is an application that runs in the ION Grid. It exposes a REST Service with API methods to run one of these actions:

- Retrieve a list of all existing objects in the Data Catalog.
- Register object metadata for JSON and DSV objects.
- Retrieve metadata for JSON and DSV objects.
- Retrieve a list of BOD nouns.
- Retrieve noun properties for BODs.

Interface and consumption methods are exposed through the Data Catalog API Service registered within the ION API Suite for Infor ION.

For more information on using ION APIs and interacting with Swagger documentation for the API methods, see *Infor ION API Administration Guide*.

Additionally, the API uses the OAuth 1.0 authorization type. To use this method of authentication, you must obtain security credentials for the DataCatalog service from your system administrator. For a technical description of the API methods, see the swagger documentation of the Data Catalog endpoint on:

`https://<your server name>:9543/datacatalog/swagger.json`

Note: The port number may be different on your installation.

To check whether the `/datacatalog` endpoint is started, run the ping method:

`https://<your server name>:9543/datacatalog/ping`

You can run this method from a browser window. A successful reply returns the REST API version number, such as 1.

Generating security credentials

A client application can connect to the Data Catalog REST service using OAuth 1.0 security credentials.

To generate the OAuth security keys for a client application:

1. On the server where ION is installed, open the Grid UI.
2. Select **Security > OAuth Credentials**. Click **+ Add New**.
3. Specify a name such as **`DataCatalog_account`**.
4. In the list of roles, select **`DataCatalog`**. On the right side, select one of these check boxes:

- **datacatalog-admin** to give access to all the available API methods
- **datacatalog-reader** to give access to the read-only methods of the GET type.

5 Click **Save**.

Note: The OAuth keys are now displayed. Save these keys before you close the dialog box.

6 Copy the Secret Key and Consumer Key to a text file.

7 Click **OK**.

The client application can use these keys to connect to the data catalog.

These are the authorization details:

- Type: OAuth 1.0
- Consumer Key: See the saved value
- Consumer Secret: See the saved value
- Signature Method: HMAC-SHA256

Available REST APIs

This table shows the available API methods:

Method	Type	Description
`/datacatalog/ping`	GET	Verify whether the REST Service is running. If successful, the reply contains the API version number.
`/datacatalog/v1/document/list`	GET	Returns a list of all documents in the Data Catalog. Optionally, you can filter by document type or document name. **Note:** This API has been deprecated. Use `/datacatalog/v1/object/list` instead.

Method	Type	Description
`/datacatalog/v1/document/json`	PUT	Upload a document of type JSON with its schema and properties. For validation details, see Defining a custom object of type JSON on page 94. Documents are uniquely identified by their name. A subsequent upload overwrites the previous document definition. **Note:** This API has been deprecated. Use the POST `/datacatalog/v1/object` API instead.
`/datacatalog/v1/document/json/{name}/schema`	GET	Returns the JSON schema for the specified document name. **Note:** This API has been deprecated. Use `/datacatalog/v1/object/{name} /schema` instead.
`/datacatalog/v1/document/json/{name}/properties`	GET	Returns the JSON properties for the specified document name. **Note:** This API has been deprecated. Use `/datacatalog/v1/object/{name} /properties` instead.
`/datacatalog/v1/object`	POST	Upload an object to the Data Catalog. For validation details, see the “Defining a custom document of type …” section for the specific type. **Note:** This API cannot be used to upload a BOD object. Objects are uniquely identified by their name. A subsequent upload overwrites the previous object definition for that type. If an object has the same name as another object of a different type, the import for the new object fails.

Method	Type	Description
`/datacatalog/v1/object/list`	GET	Returns a list of all objects in the Data Catalog. Optionally, you can filter by object name or object type.
`/datacatalog/v1/object/{name}`	GET	Returns the object name, type, subtype, schema, properties, `lastUpdatedOn`, and `lastUpdatedBy` for the specified object name.
`/datacatalog/v1/object/{name}/schema`	GET	Returns the object schema for the specified object name. **Note:** This API cannot be used to retrieve BOD or ANY objects.
`/datacatalog/v1/object/{name}/properties`	GET	Returns the object properties for the specified object name. **Note:** This API cannot be used to retrieve BOD or ANY objects.
`/datacatalog/v1/object/{name}/type`	GET	Returns the object type for the specified object name.
`/datacatalog/v1/object/fetch`	POST	Returns the object name, type, subtype, schema, properties, `lastUpdatedOn`, and `lastUpdatedBy` for each of the objects specified in the API call. Up to 100 object names can be specified in the call. **Note:** This API cannot be used to retrieve BOD metadata
`/datacatalog/v1/object/fetch/audits`	POST	Returns the name, type, subtype, `lastUpdatedOn`, and `lastUpdatedBy` information for each of the objects that are specified in the API call. Up to 100 object names can be specified in the call. **Note:** This API cannot be used to retrieve BOD information.

Method	Type	Description
`/datacatalog/v1/object/summary`	GET	Returns the total object count and the most recent `lastUpdatedOn` value for objects in the Data Catalog. Optionally, you can filter by object type. **Note:** This API excludes BODs

Business Rules API

The Business Rules APIs expose functionality to request the execution of an approval matrix or a decision matrix. These APIs can only execute matrices that have been created and approved in the Business Rules UI in ION Desk. These APIs are located in the “Infor ION” API suite in the “businessrules” endpoint.

Note:

- Only the matrices that are approved can be listed and executed through the Business Rules API.
- The latest version of the matrix is used for each API call.
- To execute a matrix, values are required for the matrix input parameters. The execution of a matrix returns the same results as the Simulation operation in the Business Rules UI. For example, the results of an approval matrix execution could be a list with distributions to the same user, displayed several times. The application that calls the API decides how to handle the approval chain. For example, the application can perform one of these actions:
  - Merge duplicate distributions and send tasks one after the other, as in the workflow task chain in ION.
  - Simultaneously send parallel tasks to all users in the resulting list.
- The result of an approval matrix execution is a list with distributions to users. These users are represented by the **IFS Person ID**, **Group Names**, and **ManagerOf** properties from Infor Ming.le User Management. To retrieve the other relevant properties of these distribution elements, the application that calls the API must be integrated with Infor Ming.le User Management.
- The result of a decision matrix execution is a list of values for the output parameters from the matrix definition. The values from the first row that matched the matrix conditions, which are evaluated for the values provided for the matrix input parameters, are returned. For detailed information about the methods that are exposed in this endpoint, see the Swagger documentation.

For more information on using ION APIs and interacting with Swagger documentation for the API methods, see the *Infor ION API Administration Guide*.

Appendix A: Valid characters for document names

Custom documents can use any standard letter from any language.

Upon import, the allowed characters are verified using a regular expression that is similar to this sample expression:

```
/[\u0041-\u005A\u0061-\u007A\u00AA\u00B5\u00BA\u00C0-\u00D6\u00D8-
\u00F6\u00F8-\u02C1\u02C6-\u02D1\u02E0-\u02E4\u02EC\u02EE\u0370-
\u0374\u0376\u0377\u037A-\u037D\u0386\u0388-\u038A\u038C\u038E-\u03A1\u03A3-
\u03F5\u03F7-\u0481\u048A-\u0527\u0531-\u0556\u0559\u0561-\u0587\u05D0-
\u05EA\u05F0-\u05F2\u0620-\u064A\u066E\u066F\u0671-
\u06D3\u06D5\u06E5\u06E6\u06EE\u06EF\u06FA-\u06FC\u06FF\u0710\u0712-
\u072F\u074D-\u07A5\u07B1\u07CA-\u07EA\u07F4\u07F5\u07FA\u0800-
\u0815\u081A\u0824\u0828\u0840-\u0858\u08A0\u08A2-\u08AC\u0904-
\u0939\u093D\u0950\u0958-\u0961\u0971-\u0977\u0979-\u097F\u0985-
\u098C\u098F\u0990\u0993-\u09A8\u09AA-\u09B0\u09B2\u09B6-
\u09B9\u09BD\u09CE\u09DC\u09DD\u09DF-\u09E1\u09F0\u09F1\u0A05-
\u0A0A\u0A0F\u0A10\u0A13-\u0A28\u0A2A-
\u0A30\u0A32\u0A33\u0A35\u0A36\u0A38\u0A39\u0A59-\u0A5C\u0A5E\u0A72-
\u0A74\u0A85-\u0A8D\u0A8F-\u0A91\u0A93-\u0AA8\u0AAA-\u0AB0\u0AB2\u0AB3\u0AB5-
\u0AB9\u0ABD\u0AD0\u0AE0\u0AE1\u0B05-\u0B0C\u0B0F\u0B10\u0B13-\u0B28\u0B2A-
\u0B30\u0B32\u0B33\u0B35-\u0B39\u0B3D\u0B5C\u0B5D\u0B5F-
\u0B61\u0B71\u0B83\u0B85-\u0B8A\u0B8E-\u0B90\u0B92-
\u0B95\u0B99\u0B9A\u0B9C\u0B9E\u0B9F\u0BA3\u0BA4\u0BA8-\u0BAA\u0BAE-
\u0BB9\u0BD0\u0C05-\u0C0C\u0C0E-\u0C10\u0C12-\u0C28\u0C2A-\u0C33\u0C35-
\u0C39\u0C3D\u0C58\u0C59\u0C60\u0C61\u0C85-\u0C8C\u0C8E-\u0C90\u0C92-
\u0CA8\u0CAA-\u0CB3\u0CB5-\u0CB9\u0CBD\u0CDE\u0CE0\u0CE1\u0CF1\u0CF2\u0D05-
\u0D0C\u0D0E-\u0D10\u0D12-\u0D3A\u0D3D\u0D4E\u0D60\u0D61\u0D7A-\u0D7F\u0D85-
\u0D96\u0D9A-\u0DB1\u0DB3-\u0DBB\u0DBD\u0DC0-\u0DC6\u0E01-
\u0E30\u0E32\u0E33\u0E40-
\u0E46\u0E81\u0E82\u0E84\u0E87\u0E88\u0E8A\u0E8D\u0E94-\u0E97\u0E99-
\u0E9F\u0EA1-\u0EA3\u0EA5\u0EA7\u0EAA\u0EAB\u0EAD-
\u0EB0\u0EB2\u0EB3\u0EBD\u0EC0-\u0EC4\u0EC6\u0EDC-\u0EDF\u0F00\u0F40-
\u0F47\u0F49-\u0F6C\u0F88-\u0F8C\u1000-\u102A\u103F\u1050-\u1055\u105A-
\u105D\u1061\u1065\u1066\u106E-\u1070\u1075-\u1081\u108E\u10A0-
\u10C5\u10C7\u10CD\u10D0-\u10FA\u10FC-\u1248\u124A-\u124D\u1250-
\u1256\u1258\u125A-\u125D\u1260-\u1288\u128A-\u128D\u1290-\u12B0\u12B2-
\u12B5\u12B8-\u12BE\u12C0\u12C2-\u12C5\u12C8-\u12D6\u12D8-\u1310\u1312-
\u1315\u1318-\u135A\u1380-\u138F\u13A0-\u13F4\u1401-\u166C\u166F-
\u167F\u1681-\u169A\u16A0-\u16EA\u1700-\u170C\u170E-\u1711\u1720-
\u1731\u1740-\u1751\u1760-\u176C\u176E-\u1770\u1780-\u17B3\u17D7\u17DC\u1820-
\u1877\u1880-\u18A8\u18AA\u18B0-\u18F5\u1900-\u191C\u1950-\u196D\u1970-
\u1974\u1980-\u19AB\u19C1-\u19C7\u1A00-\u1A16\u1A20-\u1A54\u1AA7\u1B05-
\u1B33\u1B45-\u1B4B\u1B83-\u1BA0\u1BAE\u1BAF\u1BBA-\u1BE5\u1C00-\u1C23\u1C4D-
\u1C4F\u1C5A-\u1C7D\u1CE9-\u1CEC\u1CEE-\u1CF1\u1CF5\u1CF6\u1D00-\u1DBF\u1E00-
\u1F15\u1F18-\u1F1D\u1F20-\u1F45\u1F48-\u1F4D\u1F50-
\u1F57\u1F59\u1F5B\u1F5D\u1F5F-\u1F7D\u1F80-\u1FB4\u1FB6-\u1FBC\u1FBE\u1FC2-
\u1FC4\u1FC6-\u1FCC\u1FD0-\u1FD3\u1FD6-\u1FDB\u1FE0-\u1FEC\u1FF2-
\u1FF4\u1FF6-\u1FFC\u2071\u207F\u2090-\u209C\u2102\u2107\u210A-
\u2113\u2115\u2119-\u211D\u2124\u2126\u2128\u212A-\u212D\u212F-\u2139\u213C-
\u213F\u2145-\u2149\u214E\u2183\u2184\u2C00-\u2C2E\u2C30-\u2C5E\u2C60-
\u2CE4\u2CEB-\u2CEE\u2CF2\u2CF3\u2D00-\u2D25\u2D27\u2D2D\u2D30-
\u2D67\u2D6F\u2D80-\u2D96\u2DA0-\u2DA6\u2DA8-\u2DAE\u2DB0-\u2DB6\u2DB8-
\u2DBE\u2DC0-\u2DC6\u2DC8-\u2DCE\u2DD0-\u2DD6\u2DD8-
\u2DDE\u2E2F\u3005\u3006\u3031-\u3035\u303B\u303C\u3041-\u3096\u309D-
\u309F\u30A1-\u30FA\u30FC-\u30FF\u3105-\u312D\u3131-\u318E\u31A0-
\u31BA\u31F0-\u31FF\u3400-\u4DB5\u4E00-\u9FCC\uA000-\uA48C\uA4D0-
\uA4FD\uA500-\uA60C\uA610-\uA61F\uA62A\uA62B\uA640-\uA66E\uA67F-\uA697\uA6A0-
\uA6E5\uA717-\uA71F\uA722-\uA788\uA78B-\uA78E\uA790-\uA793\uA7A0-
\uA7AA\uA7F8-\uA801\uA803-\uA805\uA807-\uA80A\uA80C-\uA822\uA840-
\uA873\uA882-\uA8B3\uA8F2-\uA8F7\uA8FB\uA90A-\uA925\uA930-\uA946\uA960-
\uA97C\uA984-\uA9B2\uA9CF\uAA00-\uAA28\uAA40-\uAA42\uAA44-\uAA4B\uAA60-
\uAA76\uAA7A\uAA80-\uAAAF\uAAB1\uAAB5\uAAB6\uAAB9-\uAABD\uAAC0\uAAC2\uAADB-
\uAADD\uAAE0-\uAAEA\uAAF2-\uAAF4\uAB01-\uAB06\uAB09-\uAB0E\uAB11-
\uAB16\uAB20-\uAB26\uAB28-\uAB2E\uABC0-\uABE2\uAC00-\uD7A3\uD7B0-
\uD7C6\uD7CB-\uD7FB\uF900-\uFA6D\uFA70-\uFAD9\uFB00-\uFB06\uFB13-
\uFB17\uFB1D\uFB1F-\uFB28\uFB2A-\uFB36\uFB38-
\uFB3C\uFB3E\uFB40\uFB41\uFB43\uFB44\uFB46-\uFBB1\uFBD3-\uFD3D\uFD50-
\uFD8F\uFD92-\uFDC7\uFDF0-\uFDFB\uFE70-\uFE74\uFE76-\uFEFC\uFF21-
\uFF3A\uFF41-\uFF5A\uFF66-\uFFBE\uFFC2-\uFFC7\uFFCA-\uFFCF\uFFD2-
\uFFD7\uFFDA-\uFFDC]+
```