Sterling Selling and Fulfillment Foundation

# Customizing APIs

*Version 9.1*

Sterling Selling and Fulfillment Foundation

# Customizing APIs

*Version 9.1*

**Note**
Before using this information and the product it supports, read the information in "Notices" on page 49.

# Contents

# Chapter 1. Checklist for Customization Projects

## Customization Projects

Projects to customize or extend Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales vary with the type of changes that are needed. However, most projects involve an interconnected series of changes that are best carried out in a particular order. The checklist identifies the most common order of customization tasks and indicates which guide in the documentation set provides details about each stage.

## Prepare Your Development Environment

Set up a development environment that mirrors your production environment, including whether you deploy your application on a WebLogic, WebSphere®, or JBoss application server. Doing so ensures that you can test your extensions in a real-time environment.

You install and deploy your application in your development environment following the same steps that you used to install and deploy it in your production environment. Refer to your system requirements and installation documentation for details.

You have an option to customize your application with Microsoft COM+. Using Microsoft COM+ has advantages such as increased security, better performance, increased manageability of server applications, and support for clients of mixed environments. If this is your choice, see the *Customization Basics Guide* about additional installation instructions.

## Plan Your Customizations

Are you adding a new menu entry? Or customizing the sign-in screen or logo? Or customizing views or wizards? Or creating new themes or new screens? Each type of customization varies in scope and complexity.

For background, see the *Customization Basics Guide*, which summarizes the types of changes that you can make and provides important guidelines about file names, keywords, and other general conventions.

## Extend the Database

For many customization projects, the first task is to extend the database so that it supports the other UI or API changes that you make later. For instructions, see the *Extending the Database Guide*, which includes information about the following topics:

- Important guidelines about what you can and cannot change in the database.
- Information about modifying APIs. If you modify database tables so that any APIs are impacted, you must extend the templates of those APIs or you cannot store or retrieve data from the database. This step is required if table modifications impact an API.
- How to generate audit references so that you improve record management by tracking records at the entity level. This step is optional.

## Make Other Changes to APIs

Your application can call or invoke standard APIs or custom APIs. For background about APIs and the services architecture of service types, behavior, and security, see the *Customizing APIs Guide*. This guide includes information about the following types of changes:

- Invoke standard APIs for displaying data in the UI and for saving changes made in the UI to the database.
- Invoke customized APIs for executing your custom logic in the extended service definitions and pipeline configurations.
- APIs use input and output XML to store and retrieve data from the database. If you don't extend these API input and output XML files, you may not get the results you want in the UI when your business logic is executing.
- Every API input and output XML file has a DTD and XSD associated to it. Whenever you modify input and output XML, you must generate the corresponding DTD and XSD to ensure data integrity. If you don't generate the DTD and XSD for extended XMLs, you may get inconsistent data.

## Customize the UI

IBM® applications support several UI frameworks. Depending on your application and the customizations you want to make, you may work in only one or in several of these frameworks. Each framework has its own process for customizing components such as menu items, logos, themes, and so on.

Depending on the framework you want, consult one of the following guides:

- *Customizing the Console JSP Interface Guide*
- *Customizing the Swing Interface Guide*
- *Customizing User Interfaces for Mobile Devices Guide*
- *Customizing the Rich Client Platform Guide* and *Using the RCP Extensibility Tool Guide*
- *Customizing the Web UI Framework Guide*

Depending on the framework you want, consult one of the following guides:

- *Customizing the Console JSP Interface Guide*
- *Customizing the Swing Interface Guide*
- *Customizing User Interfaces for Mobile Devices Guide*
- *Customizing the Rich Client Platform Guide* and *Using the RCP Extensibility Tool Guide*
- *Customizing the Web UI Framework Guide*

## Extend Transactions

You can extend and enhance the standard functionality of your application by extending the Condition Builder and by integrating with external systems. For background about transaction types, security, dynamic variables, and extending the Condition Builder, see the *Extending Transactions Guide* and *Extending the Condition Builder Guide*. These guides includes information about the following types of changes:

- Extend the Condition Builder to define complex and dynamic conditions for executing your custom business logic and using a static set of attributes.

- Define variables to dynamically configure properties belonging to actions, agents, and services configurations.
- Set up transactional data security for controlling who has access to what data, how much they can see, and what they can do with it.
- Create custom time-triggered transactions. You can invoke and schedule custom time-triggered transactions in much the same manner as you invoke and schedule the time-triggered transactions supplied by your application.
- Coordinate your custom, time-triggered transactions with external transactions and run them either by raising an event, calling a user exit, or invoking a custom API or service.

## Build and Deploy your Customizations or Extensions

After performing the customizations that you want, you must build and deploy your customizations or extensions.

1. Build and deploy your customizations or extensions in the test environment so you can verify them.
2. When you are ready, repeat the same process to build and deploy your customizations and extensions in your production environment.

For instructions about this process, see the *Customization Basics Guide* which includes information about the following topics:

- Building and deploying standard resources, database extensions, and other extensions (such as templates, user exits, and Java™ interfaces).
- Building and deploying enterprise-level extensions.

# Chapter 2. Extending Services

## About Extending Services

InSterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales terminology, a service is a core business logic component that is stateless and does not contain presentation logic. Each service (either provided out-of-the-box by the application or those that are custom created using the Service Definition Framework) represents a logical unit of processing that can be independently performed without any loss of data integrity and within one transaction boundary. Using the Service Definition Framework, one or more services can be aggregated into larger composite services which can in turn be used to create other services. This provides a way to build small reusable components that can be linked together to provide complex business processing.

All services within the Service Definition Framework can be invoked bidirectionally either through internal application business processes or through external systems. Services deployed in the Service Definition Framework are stateless, each having their own transaction commitment boundaries.

A service can be invoked by the application by associating the service with an event through an action. You can use a standard interoperability event handler or implement your own custom event handler. You can then configure the application to invoke the event handlers when certain events are raised and conditions are met. For more information about configuring events, conditions, and actions, see the Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *Configuration Guide Application Platform Configuration Guide*.

The application provides several user exits to extend business logic. User exits invoked from within transactions can be associated to a service when configuring transactions. Note that templates are not supported for user exits. For more information on configuring user exits, see the Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *Configuration Guide Application Platform Configuration Guide*.

Once services have been configured, they can also be invoked programmatically by a client.

The service invocation configuration depends on the location of the client invoking the service in relation to the location of the Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales installation, as described in the following situations:

- If the invoking client **does not** have Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales installed - Configure for remote invocation.
- If the invoking client **does** have Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales installed - Configure for local invocation.

## Invoking Services Synchronously or Asynchronously

Depending upon the mode of invocation, services can be classified into two major categories:

- Synchronously invoked services (on demand) - These services can perform all their processing and return the result in single call.
- Asynchronously invoked services (message driven)

### Synchronously Invoked Services

These services can perform all their processing and return the result in a single call, on demand.

### Asynchronously Invoked Services

These services automatically perform all their processing whenever triggered by a message from an external system or from within Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales. The trigger could be in the form of a file, a database record or a message in a message queue depending upon the mode of integration. These services do not return any value and are purely used for background processing such as sending out emails or automatically receiving updates from or sending updates to an external system.

In general, asynchronous services provide a lower cost to performance ratio than synchronous services and should be preferred wherever possible. However, asynchronous services queue up and process messages in the order they are received. The time to process a certain transaction after it's been queued can vary widely depending upon peaks in your processing cycle and a host of other factors. Therefore, they are not suitable for certain specific scenarios where an SLA (service level agreement) requires that a transaction has to be processed within a specified short time frame. However, these scenarios are rare for most businesses and business processes and asynchronous processing is efficient enough for the majority of transactions at a significantly lower cost while still providing a high service level.

## Business Functions To Use In Services

A service typically consists of one or more messaging components (or components that define how messages to and from the service are handled), one or more utility components (such as email or alert handlers) and one or more business processing components. For information about the utility and messaging components available for services defined in the application, see the Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *Configuration Guide Application Platform Configuration Guide*. This section describes how to work with services, customize and extend the business processing components, and make them usable in services.

The application is shipped with an extensive out-of-the-box business function library. Each function in this library is known as a standard API. For detailed information on the input, output, and behavior of each standard API, see the Javadocs.

You can also write your own business functions and use them in services. Each such function is known as an extended API.

While standard APIs can be aggregated and linked together to form more complex services, for most cases the API provides all the functionality that is required for a business transaction and is therefore not required to be linked together with other components or APIs. To ease working with this most common scenario, all of the standard APIs are automatically available for synchronous invocation without the need to model each one as a service using the Service Definition Framework. You can think of these APIs as "automatically defined" synchronous services. However, extended APIs and asynchronous invocation of these APIs requires that you explicitly model them as services first using the Service Definition Framework.

## Message Size For Asynchronous Services

If a database table reaches its maximum size and a send() function attempts to insert a message in the table, the Service Definition Framework throws an exception. See the size limitations noted in the following table:

| Mechanism | Data Type | Size |
|---|---|---|
| Database (Oracle) | CLOB | 4 GB |
| Database (Microsoft SQL Server) | TEXT | 4 GB |
| JMS Queue | TextMessage | 4 GB |
| MSMQ Queue | Message | 2 GB |

## Exception Handling and Services

The Alert Console displays all exceptions logged by the Service Definition Framework. It also enables you to reprocess exceptions that occur in transactions configured to be asynchronous. When using a database or queue, calls are asynchronous.

The Service Definition Framework uses the log4j utility for logging exception information. The log4j utility writes both trace and debug information to a log file. You can configure the logger to send different categories of messages to different destinations. Categories are organized hierarchically, which permits inheritance. Each category can be configured with a priority indicating a severity level. If a category is not configured with a priority, it inherits the priority of its closest ancestor with an assigned priority.

All exceptions that occur during an API call or during use of an event handler are logged.

# Chapter 3. Understanding APIs

## About APIs

You can use both standard APIs that are supplied by the application and any extended (custom) APIs that you have created. Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales provides standard APIs to handle the most common business scenarios. For example, there are APIs that create an order, allocate an order, and report shipment confirmation. Standard APIs can be invoked directly or aggregated into more complex services.

## API Behavior

Each API takes an XML document as input and returns another XML document as output. The YFSEnvironment input parameter represents a runtime state under which this API is being invoked. It is used for the following tasks:

- Security audits and logging
- Transaction control
- Achieving invocation-specific API behavior

For an asynchronous service,Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales automatically creates an instance of this object and passes it to each API part of the service. To programmatically invoke a synchronous service, you have to create an instance of this environment by calling the createEnvironment() API.

**Note:** In general, input to APIs should not contain any "BLANK" elements or attributes. A blank element can be defined as an element containing all the attributes with blank values. If a blank element is passed, the API behavior is unpredictable.

All APIs (whether standard or extended) have the same signature with respect to input parameters and return values. This signature is of the form

```
org.w3c.dom.Document APIName(YFSEnvironment env, org.w3c.dom.Document input);
```

In order for custom APIs to access custom values, the API should implement the com.yantra.interop.japi.YIFCustomApi interface. If entered, these name/value pairs are passed to the Custom API as a Properties object. See the Javadocs for more information about the com.yantra.interop.japi.YIFCustomApi interface.

## Types of APIs

An API processes records based on key attribute values, processing records with a primary key first. If the primary key is not found, the API then searches for the logical keys and then processes those records. For example, the ChangeOrder() API first looks for the OrderHeaderKey key attribute and then for the combination of the OrderNo and EnterpriseCode key attributes.

### Select APIs

Typically prefixed with "get," select APIs return one record for an entity (for example, the getOrderDetails() API returns the details of one order). They do not update the database.

Since select APIs return only one record, they require unique key attributes to be passed in the input XML. If a unique key attribute is not passed in the input XML, the API uses blanks for those attributes in the criteria to select the record. There can be more than one unique key combination, and in that combination you must pass any one of the multiple combinations.

For example, an order is uniquely identified either by the OrderHeaderKey key attribute or by a combination of the OrderNo and EnterpriseCode attributes. So, when calling the getOrderDetails() API, you must pass either the OrderHeaderKey attribute or the combination of the OrderNo, EnterpriseCode and DocumentType key attributes. If you pass only OrderNo, the API returns the order that matches OrderNo and has a blank enterprise code. In order to identify the unique key combinations for each API, see the Javadocs.

However, getOrderDetails() API uses a select for update on YFS_ORDER_HEADER so that its internal processes such as user exits, events, etc., have a lock on the order elements while the thread working on it is active. This enables to maintain a transaction cache until the final commit. Hence, you need to avoid using nested transactions to overcome the locking mechanism by performing:

1. Commit or rollback only once for all event of the order. Keep in mind that all the events are set to rollback if one of them fails.
2. Select the order for each event and process. Also keep in mind that if age of the orders having multiple events are higher it could have an impact on the performance.

### List APIs

Typically prefixed with "get," list APIs return a list of records for an entity that match the criteria specified through the input XML, for example, the getOrderList() API returns a list of orders. If any attribute in the input XML has a blank value, it is ignored. List APIs do not update the database. You can also get the paginated data from a list API by calling the getPage API and passing the list API as the input to the getPage API. For more information about the getPage API, see the Javadocs.

### Update APIs

Update APIs insert new records into the database. They also modify or delete existing records in the database. Update APIs that modify or delete existing records use the same logic as select APIs to identify which record to update. If no record is found, update APIs throw an exception.

## API Security

When calling an API, you must pass through the following two levels of security:

1. Authentication with a user ID, a certificate or both. The login API is called before any other API is called.
2. Authorization, which verifies which API that you can access.

This security procedure is for every API call that is made through an application server process. By default, agent and integration servers always have full access to APIs.

Once you have passed the authentication check, an authorization check determines what APIs and resources you can access. This authorization check is in addition to the user interface (UI) security. For example, the UI security might allow you access to a screen that lists users. To generate a list of users at the screen, you might also have to pass an authorization check for the getUserList API that lists the users.

Other examples of authorization checks include:

- If you use the getCommonCodeList API for display purposes, you should not be able to get user information that is explicitly restricted from the output of the API.
- If you call the getUserList API before assigning an alert, you should not be able to get user passwords.
- If you use the UserHierarchy API to change your password:
  - You should not be able to change your own IsSuperUser flag.
  - You should not be able to modify another user's information.
  - You should not be able to subscribe to additional user groups, which would give you more system access.

This security is implemented using the apisecurity specific template files. These apisecurity template files are XML files that documents the input and output elements to which (by default) all APIs are restricted. These files are automatically generated during XAPI deployment, even when document generation is turned off.

**Note:** Services do not use the apisecurity file.

Templates are used for the input and output authorization checks. These templates override the regular templates.

For example, an input template with the lines `OrganizationCode=#PROHIBITED#` and `IsSuperUser=#PROHIBITED#` would prevent you from subscribing to more user groups and gaining more permissions.

The output template supplements the filtering performed by the default documentation-based template. If an element is restricted because it is not configured in the apisecurity file, it will never be returned in the output, even if present in the documentation-based template.

**Note:** At certain points in the input and output, APIs like multiApi and getPage have authorization access for any element. But other APIs that are called by these APIs must go through the authorization check.

Access to API security and the permission level are controlled in the following properties in the yfs.properties file. All authorization failures are logged to a logging category named sci.apisecurity.

- api.security.enabled
  - Y (default)—Enable API security
  - N—Do not enable API security
- api.security.mode

- STRICT—If any validation fails, throw an exception. This is appropriate for production systems, if all permissions are configured properly.
- LAX—Filter out and log invalid input, but continue processing. The filtering allows the system to mostly work despite incorrect input or output, while the logging helps to identify places that need change.

  **Note:** The system may still throw an exception when the filtering produces an ambiguous behavior.
- DEBUG—Log invalid input and output, but do not filter anything or throw exceptions. This is only appropriate during initial development, to identify the permissions required by various processes.

If you do not specify a security mode, there is no filtering, thrown exceptions, or authorization checking. There is limited logging.

- api.security.override.*apiName*.mode

  Use this setting to override permissions on individual APIs. This property uses the same values as api.security.mode.
- api.security.smc.enabled
  - Y—Enable API security for the Applications ManagerConfigurator
  - N (default)—Do not enable API security for the Applications ManagerConfigurator
- api.security.console.enabled
  - Y—Enable API security for the Application Console
  - N (default)—Do not enable API security for the Application Console

When upgrading, you should initially disable this feature and grant all access through properties. In an upgraded system, you can phase in this feature by enabling security one API at a time, as you define and test permissions. If enabled, only the system user group has grant permission to the APIs; for all other custom user groups, appropriate permission has to be given. For information about user group permissions, see the Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *Configuration Guide Application Platform Configuration Guide*.

# Include the apisecurity File in the Documentation

## About this task

To include the apisecurity file in the documentation, and package it in the EAR (enterprise archive) file, do the following:

## Procedure

1. Place the extensions in the *INSTALL_DIR*/xapidocs/extn/input directory.
2. Rebuild the documentation (including the apisecurity files) by running the following command:

   ```
   deployer.sh -t xapideployer -l info [new target]
   ```
3. Build the EAR file. The apisecurity template files will be packaged from: *INSTALL_DIR*/repository/xapi/template/merged/apisecurity.

## Date and Time Handling by APIs

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales handles values for both date-time and date. Date-time refers to values that contains a date and time component, where Date refers to values that contain only a date component.

Date values can be made nullable by specifying `Nullable="true"` in the entity XML. Thereby the Date values in the table is blanked out. The expected behavior of a date column is marked as Y is described in the following table.

| Action | Description |
|---|---|
| Insert | When the field is not populated in a database object (is null), the database infrastructure automatically inserts a null value into the column in the database. |
| Update | When the application nulls out a date, it sets the corresponding field to null value in the database. |
| Select or List | When a column is defined as nullable and the date from the database is returned as null, it is automatically nulled out. So, the corresponding get method returns a null. |
| Search by date | Can pass null value as needed when specified to do so. |

**Note:** If you have specified the date value as 01/01/2400 in versions prior to Release 8.5, those values are now treated as null. The dates with special significance are:

- Null date - 01/01/2400
- High date - 01/01/2500
- Low date - 01/01/1900

## Specifying Time Zones

Dates and times are time zone aware. Time zones are relative to the Coordinated Universal Time (UTC).

For example, if an order is created on the system on 06/15/2003 at 16:00:00 in New York, (USA/New York time zone) a user in Chicago who examines that order observes that order creation date-time as 06/15/2003 at 15:00:00, (USA/Chicago time zone).

For a time published from Boston that is -5:00 hours from UTC, the string literal "-5:00" is appended to the current date-time attribute published from APIs. The input "2003-04-23T14:15:32-05:00" gives the date, time, and time zone reference for a transaction.

The yfs.install.localecode parameter in the yfs.properties file determines the Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales time zone. For example, `yfs.install.localecode=en_US_EST`

To configure the time zone, set the yfs.install.localecode property to `en_US_EST` in the *INSTALL_DIR*/properties/customer_overrides.properties file.

## Using Date-Time Syntax

All APIs, user exits, and events that use date-time fields have a uniform syntax (a combination of the basic and extended formats of the ISO 8601 specification). This syntax is the expected format for all input as well as output.

### Date Only Syntax

*YYYY*-*MM*-*DD*

### Date-Time Syntax

*YYYY*-*MM*-*DD***T***HH*:*MI*:*SS*+*HH*:*MM*

Values in bold above are placeholders for literals. For example, the format for March 5, 2003, 11:30:59 p.m. is 2003-03-05T23:30:59.

**Note:** This syntax is an ISO Date-Time syntax and not the database syntax. Using a syntax other than the ISO Date-Time format may cause problems. For example, the time element in the Date-Time syntax may be overlooked or calculated incorrectly.

For example, if you provide the Date-Time input as "2007-05-18-19.10.28.000000", the system may interpret it as just "2007-05-18" because the T symbol is missing in the input.

### Syntax Parameters

| Parameter | Description |
|---|---|
| *YYYY* | Required. Four-digit year. Used in both date-time and date fields. |
| *MM* | Required. Two-digit month. Used in both date-time and date fields. |
| *DD* | Required. Two-digit day of the month. Used in both date-time and date fields. |
| T | Required. The literal value T, which separates the date and time component. Used only in date-time fields. |
| *HH* | Required. Two-digit hour of the day. For example, 11 p.m. is displayed as 23:00:00. Used only in date-time fields. |
| *MI* | Required. Two-digit minutes of the hour. For example, 59 minutes is displayed as 00:59:00. Used only in date-time fields. |
| *SS* | Required. Two-digit seconds of the minute. For example, 21 seconds is displayed as 00:00:21. Used only in date-time fields. |
| *+HH:MI* | Optional. Two-digit hours and minutes, separated by a colon (":"). Indicates how many hours from UTC, using - to indicate earlier than UTC and + to indicate later than UTC. If this value is not passed in input, the time zone of the application is assumed. |

# Chapter 4. Input XML Files for APIs

## About Input XML Files for APIs

APIs retrieve data using input XML files that define which records need to be selected or used. When extending the database to include additional fields, you need to also extend the input XML to populate those fields.

**CAUTION:** Do not pass a blank element (an element containing all the attributes with blank values) to an API. Also, do not pass attributes that have leading or trailing spaces. The result of either situation is not predictable.

The following example shows an input XML modification.

### Example of Input XML Modification

The following example modifies the input XML file for the YFS_createOrder() API:

```
<Orders AuthenticationKey="">
   <Order EnterpriseCode="DEFAULT" OrderNo="DB04"
OrderName="DB04" OrderDate="20010803" OrderType="Phone" PriorityCode="1"
PriorityNumber="1" ReqDeliveryDate="20010810" ReqCancelDate=""
ReqShipDate="20010810" SCAC="FEDEX" CarrierServiceCode="Express Saver Pak"
CarrierAccountNo="112255" NotifyAfterShipmentFlag="N"
NotificationType="FAX" NotificationReference="" ShipCompleteFlag="N"
EnteredBy="Iain " ChargeActualFreightFlag="Y" AORFlag="Y"
SearchCriteria1="Search" SearchCriteria2="Search Again" >
      <OrderLines>
         <OrderLine PrimeLineNo="1" SubLineNo="1"
OrderedQty="1" ReqDeliveryDate="20010810" ReqCancelDate="20010810"
ReqShipDate="20010810" SCAC="FEDEX" CarrierServiceCode="Express
Saver Pak" PickableFlag="Y" HoldFlag="N" CustomerPONo="11" >
            <Extn ExtnAcmeLineType="Type1"/>
            <Item ItemID="ITEM1" ProductClass="A" ItemWeight="1"
ItemDesc="paintball gun" ItemShortDesc="pball gun"
UnitOfMeasure="EACH" CustomerItem="Spectra Flex" CustomerItemDesc="GEGRG"
SupplierItem="Spectra Flex @ supplier" SupplierItemDesc="Spectra
Flex Desc @ supplier" UnitCost="15.99"  CountryOfOrigin="CA"/>
            <PersonInfoShipTo Title="Mr" FirstName="Quigley"
MiddleName="Al" LastName="Johns" Company="Company" JobTitle="Project
Clert" AddressLine1="Address Line 1 -3 Main Street" AddressLine2="ShipTo
Address line 2" AddressLine3="ShipTo Address line 3" AddressLine4="ShipTo
Address line 4" AddressLine5="ShipTo Address line 5" AddressLine6="ShipTo
Address line 6" City="Acton" State="MA" ZipCode="01720" Country="US"
DayPhone="978-635-9242" EveningPhone="978-635-9252" MobilePhone="978-888-8888"
Beeper="" OtherPhone="other555-5555" DayFaxNo="" EveningFaxNo=""
EMailID="jquigley@maine.com" AlternateEmailID="hfournier@ontario.com"
ShipToID=""/>
         </OrderLine>
         <NumberOfOrderLines/>
      </OrderLines>
      <PersonInfoShipTo Title="MR" FirstName="s"
MiddleName="X" LastName="T" Suffix="T" Department="T" Company="SD"
JobTitle="SS" AddressLine1="SS" AddressLine2="SS" AddressLine3="SS"
AddressLine4="SS" AddressLine5="SS" AddressLine6="SS" City="REDWOOD"
State="CA" ZipCode="01852" Country="USA" DayPhone="3456789234"
EveningPhone="3456789234" MobilePhone=""  EveningFaxNo="SS" />
       <PersonInfoBillTo Title="mj" FirstName="m"
MiddleName="JJ" LastName="KK" Suffix="lll" Department="l" Company="kj"
JobTitle="k" AddressLine1="HJHKK" AddressLine2="HJKHK" AddressLine3="HKHJ"
AddressLine4="" AddressLine5="" AddressLine6="" City="UUU" State="IUI"
```

```
ZipCode="78787" Country="USA"  />
  </Order>
  <NumberOfOrders/>
</Orders>
```

**Important:** In order for the factory setup scripts to operate properly, when you add a column to a database table, be sure that the column is not null and that it has a default value. If you need to make the column nullable, the default value must not be present.

Also, when you are specifying XML Name and XML Group, keep in mind that the values should be valid Document Object Model (DOM) strings. (The values must not contain spaces or special characters that are not supported by the DOM specification.)

The following example XML file adds a column to the YFS_ORDER_LINE table:

```
<?xml version="1.0" encoding="UTF-8" ?>
<DBSchema>
 <Entities>
  <Entity TableName="YFS_ORDER_LINE">
   <Attributes>
    <Attribute ColumnName="EXTN_ACME_LINE_TYPE" DecimalDigits=""
Default Value="' '" Size="10" Type="CHAR" XMLGroup="Extn"
XMLName="ExtnAcmeLineType"/>
   </Attributes>
  </Entity>
 </Entities>
</DBSchema>
```

# Guidelines for Forming API Input

When coding API input parameters, follow the stated guidelines for using literals and formatting API input.

Do not pass a blank element (an element containing all the attributes with blank values) to an API. Also, do not pass attributes that have leading or trailing spaces. The result of either situation is not predictable.

## Using Literals in Maps and XMLs

Using literals enables you to write code with fewer bugs because the compiler catches the use of incorrect names in the *name=value* pair. In addition, using literals simplifies the maintenance of your code; if you change the *name*, all you need to do is recompile your code instead of editing one or more *name* instances within it first.

## Using Special Characters

The fields that are a part of the logical key for any record in the Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales schema (such as OrganizationCode and OrderNo) have some restrictions. For such fields, the application does not support the use of special characters listed in the following table.

| Special Character | Description |
|---|---|
| & | Ampersand |
| > | Greater Than |
| < | Less Than |

| Special Character | Description |
|---|---|
| % | Percent |
| " | Quotation Mark |
| + | Plus sign |
| ' | Single quotation or Apostrophe |
| ( | Parenthesis |
| ) | Parenthesis |
| [ | Square Bracket |
| ] | Square Bracket |

**Note:** You can use the plus (+) and ampersand (&) signs only in the ItemID field.

**Note:** A forward slash (/) must not be used in attributes that form part of a path in another attribute. For example, Category ID must not include a forward slash, as Category ID is used to form the attribute Category Path, which uses the forward slash as a path separator.

In addition,IBM recommends against using third-party vendors' reserved special characters. For example, in certain situations, data with underscore characters ("_") on an Oracle database could result in unexpectedly slow query performance because the database deciphers the underscore as a single character wild-card.

The following fields have no restrictions and support all characters:

- All description fields (for example, item description)
- All name fields (for example, organization name)
- All address fields (for example, billing address)

  **Note:** However when creating address fields through the UI, the information entered after the quotation mark (") is truncated and appears as a new entry in YFS_PERSON_INFO table. To work around this problem, use apostrophe (') instead of quotations.
- All instruction fields (for example, gift wrapping)
- All text fields (for example, reasons and comments)

**Note:** The Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales Mobile Device does not support the use of the ampersand (&) character.

## XML-Based APIs

The following table lists special characters and the escape sequences that should be used in place of them in XML.

| For This Character | Enter This Sequence |
|---|---|
| quotation mark (") | &quot; |
| single quotation (') | &apos; |
| greater than symbol (>) | &gt; |
| less than symbol (<) | &lt; |
| ampersand (&) | &amp; |

## Support for CreateTS and ModifyTS in Input and Output XML Files

CreateTS and ModifyTS can be used in getAPIs(input or output) if an entity in the input or output XML file for which these attributes are requested have corresponding tables. These attributes indicate when a record was created or modified in the database.

# Forming Queries in the Input XML of List APIs

The input XML of list APIs enable queries on conditions such as starts with, contains, is greater than, and so forth. The following example shows a fragment of the input XML that returns a list of items at a specific shipping node that fall within a specific weight range and to be shipped during a specific date range.

### Example: getOrderList API Input XML with Query Type Values

```
<Order ReqShipDateQryType="DATERANGE" FromReqShipDate="20010113" ToReqShipDate="20030113" /Order>
<OrderLine ShipNode="Atlantic" /OrderLine>
<Item ItemWeightQryType="BETWEEN" FromItemWeight="2" ToItemWeight="20"/>
<OrderRelease CarrierServiceCodeQryType="FLIKE"
CarrierServiceCode="Priority" />
```

**Note:** Some APIs do not support QryType for an input attribute. These APIs are:

- getAssignedPricelistHeaderList
- getCarrierServiceList
- getNodeSCACAccountList
- getOrderLineStatusList
- getPaymentStatusList
- getPriceListForOrdering
- getQueryTypeList
- getReceiptLinesForTransaction
- getRegionList
- getServerList
- getShipmentListForOrder
- getSurroundingNodeList
- getTaskQueueDataList
- getTraceableComponentList
- getTraceList
- getZoneListForDiscount

# To Form Queries

### About this task

To form queries:

### Procedure

1. Edit the custom input XML of any list API, and append `QryType` to any attribute you want to query on. Any attribute that is not appended with QryType can also be queried on, using the default query type value EQ, as shown for ShipNode in the example in the topic on forming queries in the input XML of list APIs.

2. For attributes appended with QryType, specify a query type value from the following table. This is case sensitive.
3. Specify the values that are applicable to your search criteria.

   The values for the QryType attributes vary depending on the datatype of the field. The following table lists the supported query type values for each datatype.

| Field DataType | Supported Query Type Values |
|---|---|
| Char/VarChar2 | • EQ - Equal to<br>• FLIKE - Starts with<br>• LIKE - Contains<br>• GT - Greater than<br>• LT - Less than |
| Number | • BETWEEN - Range of values<br>• EQ - Equal to<br>• GE - Greater than or equal to<br>• GT - Greater than<br>• LE - Less than or equal to<br>• LT - Less than<br>• NE - Not equal to |
| Date | • DATERANGE - Range of dates<br>• EQ - Equals<br>• GE - Greater than or equal to<br>• GT - Greater than<br>• LE - Less than or equal to<br>• LT - Less than<br>• NE - Not equal to |
| Date-Time | • BETWEEN - Range of dates<br>• EQ - Equals<br>• GE - Greater than or equal to<br>• GT - Greater than<br>• LE - Less than or equal to<br>• LT - Less than<br>• NE - Not equal to |
| Null | • ISNULL - Return records that are null.<br>• NOTNULL - Return records that are not null.<br>**Note:** These two query types are used when the column or attribute is set to Nullable in the entity XML. |

# Setting Query Timeouts for XAPIs

## About this task

You can add individual query timeouts for APIs. To do this, specify the query timeout value in seconds for the API in the API's input XML. For example:

```
<ApiInput QueryTimeout="10">
...
...
...
</ApiInput>
```

**Note:** The value of the QueryTimeout attribute overrides the value of the yfs.ui.queryTimeout property in the yfs.properties file. But the value of this attribute is valid only for a single API call. After the API execution is complete, the query timeout is set to the old value based on the value of the yfs.ui.queryTimeout property in the yfs.properties file.

## Sorting Through OrderBy Element in the Input XML of List APIs

The input XML of list APIs supports sorting based on the OrderBy element.

To form queries:

Edit the custom input XML of any list API, and add the `OrderBy` element. Add the `Attribute` child element, and in the `Name` attribute specify the name of the field based on which you want to sort the results.

The OrderBy element supports ordering of the attributes in both ascending and descending order. By default, the results are sorted in ascending order. If you want to sort the results in descending order, add the `Desc` attribute to the Attribute element and set it to `Y`.

You can also do nested sorting using the OrderBy element.

### getOrganizationList API Input XML with OrderBy Element

The following example shows a fragment of the input XML that returns a list of organizations and results are sorted by the OrganizationName attribute.

```
<Organization IgnoreOrdering="N" MaximumRecords="5000">
  <OrderBy>
     <Attribute Name="OrganizationName"/>
   </OrderBy>
</Organization>
```

### getOrganizationList API Input XML with Nested OrderBy Element

The following example shows a fragment of the input XML that returns a list of organizations and results are sorted by OrganizationName and LocaleCode attributes.

```
<Organization IgnoreOrdering="N" MaximumRecords="5000">
   <OrderBy>
     <Attribute Name="OrganizationName"/>
     <Attribute Name="LocaleCode"/>
    </OrderBy>
</Organization>
```

### getOrganizationList API Input XML with OrderBy Element and Desc Attribute

The following example shows a fragment of the input XML that returns a list of organizations and results are sorted by the OrganizationName attribute in the descending order.

```
<Organization IgnoreOrdering="N" MaximumRecords="5000">
   <OrderBy>
      <Attribute Name="OrganizationName" Desc="Y"/>
   </OrderBy>
</Organization>
```

# Chapter 5. Output XML Files for APIs

## About Output XML Files and Templates for APIs

APIs return data using two types of output XML files that define which elements and attributes are required by an API.

- Output XML File - Defines the outer limits of the data an API can return. Do not modify output XML files.
- Template XML File - Defines the data returned by an API for the record specified in the input XML file and restricts the amount of data to a subset of the output XML. You can modify this file to incorporate a subset of the attributes and elements from the output XML.

### Output XML Templates

Many APIs use a corresponding output template. The output template is in XML format and is read in by an API in order to determine the elements and attributes for which it should return. The standard output template defines the elements and attributes returned for any specific API. (To see the entire range of possible values an API can return, see its output XML in Javadocs.) The standard template can be a subset of the entire range of values returned, as determined by the output XML in the Javadocs.

**Note:** Ensure that when adding elements and attributes to the output template, use only those that are documented in the Javadocs. While the APIs can output additional elements and attributes, only those that are documented in the Javadocs are supported.

For example, the standard output template of the getOrderList() API returns the header-level information of an order and the standard output template of the getOrderDetails() API returns in depth information about an order.

Besides the standard output XML template, you can create custom output templates for APIs to use for your own business requirements, such as different output for different document types.

### Document Types

If you use a variety of business-related document types such as orders, planned orders, purchase orders, and returns, you can use custom templates that enable an API to return the values that pertain to each unique document type.

For example, you can use one template with the getOrderDetails() API to return information about Planned Orders and another template for the getOrderDetails() API to return different information about Orders.

### Standard Output Template Behavior

The set of values that the standard output template returns covers a variety of business scenarios. With such a large range of possibilities, an API using the standard output template may return much more data than you need for your business purposes (and take much more time to process than you prefer).

If you want to customize the information returned by an API, you can do so by creating and using a custom template, using our guidelines and procedures.

# Extending an Output XML Template

## About this task

Many APIs use an output XML template to define what is returned. Each API has its own XML template, which is picked up from the *INSTALL_DIR*/repository/xapi/template/merged/api/*apiName*.xml file. The files in this directory are part of the product and should not be altered. However, these templates can be overridden by implementing template extensions.

To extend a template file:

## Procedure

1. Copy the template *INSTALL_DIR*/repository/xapi/template/merged/api/*apiName*.xml file to the *INSTALL_DIR*/extensions/global/template/api/ directory, keeping the same file name.

   If the /global/template/api/ directory does not exist, create the required directory structure.
2. Modify the copied file, as needed. To extend a template file, add the `Extn` tag under the entity tag. For example, if you have added a column `EXTN_COLOR` to YFS_ITEM table, you also must add the tag `Extn` under the tag `Item` in the getItemDetails.xml file as follows:

   ```
   <Item ItemKey=""....>
      <PrimaryInfo MasterCatalogID="" .../>
        ...
       <Extn ExtnColor=""/>
   </Item>
   ```

   **Note:** If you are extending an output XML template, place your extended files in the *INSTALL_DIR*/extensions/global/template/api folder. But when providing the name of the template.api file during service definition, the path should be /global/template/api/*CUSTOM-TEMPLATE-API*.

# Best Practices for Creating Custom Output XML Templates

Whenever you call an API, you need to pass your own customized template, not the sample provided by Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales. This section helps guide your decision-making processes in planning how to design custom output templates.

In general, when you customize an output template, you do so by editing a copy of the standard template. You cannot modify the standard output template.

There are two ways of customizing and calling the output template. Which function you choose depends on the size and type of the data set you want returned by the API.

## Gather Information Relevant to the API

Custom output templates provide the flexibility to return whatever data you wish, so it is important to understand that it is possible to modify an output template in such a way that it returns information that is not quite relevant to the API.

For example, it is possible to modify the output template of the getOrderList() API in such a way that it returns detailed information about an order rather than just header-level information. You should modify an output template in such a way that it takes advantage of the unique aspects of its corresponding API. Keeping each template unique to its API prevents any ambiguity about which API to use in any specific situation.

## Gather Information Relevant to Your Business Needs

Since the standard output template returns all attributes, even for empty elements in the template, you might want to tailor information to your specific business needs. If you don't exclude the attributes you don't require, you receive more data than you need and the extra data may slow the performance of the API.

For example, if you are using the getOrderDetails()API to return only OrderLine attributes but your custom output template includes Schedule attributes, all attributes for OrderLine and Schedule are returned.

## Choose an Appropriate Template Mechanism

In general, the format of any template should follow the same structure as the standard template. Keeping this general rule in mind, there are two ways to customize the standard template, differentiated by the amount of data they return and how they can be called:

- Static templates
- Dynamic templates

Static templates provide the ability to add new elements but not remove any of the defaults. A static template is pervasive, as it is picked up by default by an API whenever that API is invoked.

Dynamic templates provide the ability to add new elements and remove any of the default elements from the standard template. A dynamic template is an instance, as it is picked up only for a specific API call, such as when configured to do so during user interface extensibility.

A comparison of the differences between the two types of template mechanisms is summarized in the following table.

| Template Types | Allowed XML Elements | Behavior |
|---|---|---|
| Static Template | Default template elements cannot be removed.<br><br>New elements can be added. | Pervasive. Picked up by default by an API. |
| Dynamic Template | Default template elements can be removed.<br><br>New elements can be added. | Instance. Picked for a specific API call, as configured during user interface extensibility. |

Choose which of these mechanisms best fits your business needs and adhere to it.

Remember that when you define a dynamic template, all possible values are returned. In order to return the smallest amount of data for an element, when you are pruning away elements you don't need, you need to include its parent with at least one of its attributes.

If you leave an element blank or include unwanted attributes in the parent element all values are returned, as illustrated in the following example of a poorly pruned dynamic template.

## A Poorly Pruned Dynamic Template

```
<!-- getOrderDetails Output XML -->
<Order>
   <OrderLines>
     <!--1 or more order line-->
       <OrderLine>
          <Item CountryOfOrigin="" ItemDesc="" ItemID=""/>
          <Schedules>
             <Schedule Attr1 ..... />
          </Schedules>
       </OrderLine>
   </OrderLines>
<Order>
```

Since the poorly pruned dynamic template specifies all OrderLine attributes as well as a few Item and Schedule attributes, the API returns values similar to the following.

```
<OrderLine AllocationDate="03/28/2002" CarrierAccountNo="112233"
CarrierServiceCode="Next Day Air" Createprogid="CustomTester"
Createts="03/28/2002" Createuserid="CustomTester" CustomerLinePONo="999"
CustomerPONo="111" DeliveryCode="AIR" DepartmentCode="Clothing"
ExtendedFlag="" ExternalReference1="" ExternalReference2=""
ExternalReference3="" ExternalReference4="" ExternalReference5=""
FreightTerms="Buyer" HoldFlag="N" HoldReasonCode="HoldReas"
ImportLicenseExpDate="08/08/2002" ImportLicenseNo="225588"
InternalReference1="" InternalReference2="" InternalReference3=""
InternalReference4="" InternalReference5="" KitCode=""
LineClass="" LineSeqNo="1.1" LineType="Single" Lockid="1" MarkForKey=""
Modifyprogid="CustomTester" Modifyts="03/28/2002"
Modifyuserid="CustomTester" OrderClass="NEW"
OrderHeaderKey="200203281036245174" OrderLineKey="200203281036245175"
OrderedQty="5.00" OrigOrderLineKey="" OriginalOrderedQty="5.00"
OtherCharges="0.00" OtherChargesPerLine="0.00" OtherChargesPerUnit="0.00"
PackListType="Bill" PersonalizeCode="PersCode" PersonalizeFlag=""
PickableFlag="Y" PricingDate="01/01/2500" PrimeLineNo="1"
Purpose="Purpose" ReceivingNode="B1N1" ReqCancelDate="01/01/2500"
ReqDeliveryDate="04/04/2002" ReqShipDate="03/30/2002"
ReservationID="" ReservationPool="" SCAC="UPS" ShipNode="E1N1" ShipToID=""
ShipToKey="" ShipTogetherNo="Y" SplitQty="0.00" SubLineNo="1"
TotalDiscountAmount="0.00" TotalOtherCharges="0.00">
<Item CountryOfOrigin="" ItemDesc="" ItemID=""/>
<Schedules>
<Schedule ExpectedDeliveryDate="" ExpectedShipmentDate=""
TagNumber="" OrderHeaderKey="" OrderLineKey="" OrderLineScheduleKey=""
ScheduleNo="" ShipByDate="" Quantity=""  PromisedApptStartDate=""
PromisedApptEndDate=""/>
</Schedules>
</OrderLine>
</OrderLines>
</Order>
```

## A Carefully Pruned Custom Output Template

In this carefully pruned custom output template, the dynamic template has been trimmed down, keeping in mind the following guidelines:

- The structure of the custom output template mirrors the structure of the standard output template.

- Excess elements (regarding kits, schedules, addresses, and so forth) are pruned away.
- Parent elements are populated with one attribute in order to suppress excess detail. For example, specifying the OrderNo attribute for the Order element suppresses all of the other Order attributes.

```
<!-- getOrderDetails Output XML -->
<Order OrderNo="">
   <OrderLines>
      <!--1 or more order line-->
        <OrderLine PrimeLineNo="">
            <Item CountryOfOrigin="" ItemDesc="" ItemID=""/>
     </OrderLine>
    </OrderLines>
</Order>
```

Since this carefully pruned custom output template specifies only a few Item attributes and only one attribute for its parent element, the getOrderDetails() API returns only the following values:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Order OrderNo=Y00000765>
   <OrderLines>
      <OrderLine PrimeLineNo="1">
          <Item CountryOfOrigin="IN" Item Description"
ItemDesc="Green Sari" ItemID="GNSARI5LT" />
       </OrderLine>
       <OrderLine PrimeLineNo="3">
          <Item CountryOfOrigin="CA" ItemDesc="Pink Scarf"
ItemID="PKSCARF4LT" />
       </OrderLine>
    </OrderLines>
</Order>
```

This method of pruning the templates improves the performance as database access to order schedules and other unwanted elements has been prevented.

## Develop Useful Templates

The supplied templates located in the *INSTALL_DIR*/repository/xapi/template/merged/api/ directory are sample guides. Use them to help you develop your own output XML templates. Using your own customized templates gives you much more flexibility, greater performance, and more assurance of appropriate data output. You can either pass your template through the env or put it in the extension folder.

## Keep Performance Needs in Mind

Besides tailoring the templates to your business needs, it is important to keep technological considerations in mind. For performance-related information about using output, see the Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *Performance Management Guide*.

# Defining and Deploying a Static Template for Output XML

### About this task

If you want to use a template that has more elements in addition to those in the standard output template, create a static output template. This function enables you to create a template that includes all of the elements in the standard output template plus any new ones you add. For example, you may need to add UI fields

for any database columns you have added. Note that if you use this function, you cannot remove any elements that exist in the standard template.

To define and deploy a static template:

### Procedure

1. Copy the standard output template for the API that you want to modify from the *INSTALL_DIR*/repository/xapi/template/merged/api/*FileName*.xml file to *INSTALL_DIR*/extensions/global/template/*FileName*[.*DocType*].xml.
   - Keep the file name of your new template the same as the standard template.

     The name of the output template corresponds with the name of the API or event associated with it. For example, the getOrderDetails() API takes the output template file getOrderDetails.xml.
   - If the template references a document type, include the document type code in the filename.

     For example, to create an output template for the getOrderDetails() API for an Order (0001) document type, the name of the template XML is getOrderDetails.0001.xml.
2. Modify the copied template in the /extensions/global/template/api/ directory as required, keeping in mind the best practices for creating custom output XML templates.

   **Note:** You may add any elements you wish, but you cannot remove any of the elements present in the standard output template.
3. Call the API as typical and it automatically picks up the custom output template from the directory containing the custom templates.

## Defining and Deploying a Dynamic Template for Output XML

### About this task

If you want to use a template that contains a subset of the elements in the standard output template, create a dynamic output template. If you want the ability to remove some elements from the standard template and perhaps add your own elements, you do that by passing your XML data or a file name into the YFSEnvironment object.

To define and deploy a dynamic template:

### Procedure

1. Copy the standard output template for the API that you want to modify from the *INSTALL_DIR*/repository/xapi/template/merged/api/*FileName*.xml file to *INSTALL_DIR*/extensions/global/template/api/*FileName*.xml.

   When naming your new template file, use the same name as the standard template

   The name of the output template corresponds with the name of the API or event associated with it. For example, the getOrderDetails() API takes the output template file getOrderDetails.xml.
2. Modify the copied template in the /extensions/global/template/api directory as required, keeping in mind the best practices for creating custom output XML templates.

3. During user interface extensibility, call the setApiTemplate() function on the YFSEnvironment object. This enables you to specify an output template before calling an API, using one of the following functions:
   - XML *data* as a variable - as in the following example:

     ```
     YFSEnvironment env = createEnv();
     Document doc = getTemplateDocument();
     env.setApiTemplate("getOrderDetails", doc);
     private YFSEnvironment createEnv() {
     //create new environment by passing the user id, program id, etc.
     }
     private Document getTemplateDocument() {
     //create a Document object containing the desired template XML.
     }
     ```

   - XML *file* as a variable - as in the following example:

     ```
     YFSEnvironment env = createEnv();
     env.clearApiTemplates();
     env.setApiTemplate("getOrderDetails", "myOrderDetails");
     private YFSEnvironment createEnv() {
     //create new environment by passing the user id, program id, etc.
     }
     ```

     The API then uses the template passed in through YFSEnvironment to produce the output XML document. For details about the YFSEnvironment interface, see the Javadocs.

# Sequence of Precedence for Output XML Templates

Since Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales enables you to define multiple types of templates, in addition to the standard templates that you cannot modify, it is important to understand the order of precedence in which Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales implements when reading API and event templates.

## API Templates

The following table shows the sequence of precedence for determining which output template is used by an API. (Events use a similar sequence of precedence.) Note that templates are not supported for user exits.

| Priority | Output Template Path and File Name |
|---|---|
| 1 | setApiTemplate(*file* \| *xmlDocument*) to `YFSEnvironment`<br><br>When a file is specified, it is picked up from the *INSTALL_DIR*/extensions/global/template/api directory. |
| 2 | *INSTALL_DIR*/extensions/global/template/api/apiName.docType.xml |
| 3 | *INSTALL_DIR*/extensions/global/template/api/apiName.docType.xml (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales sample template; not for use in production.) |
| 4 | *INSTALL_DIR*/extensions/global/template/api/apiName.xml |
| 5 | *INSTALL_DIR*/extensions/global/template/api/apiName.xml (Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales sample template; not for use in production.) |

## Event Templates

Event templates help determine what elements and attributes should be present in the output XML of an event. For events raised, see the relevant transactions in the Javadocs.

To see which events take output templates, see the files in the *INSTALL_DIR*/repository/xapi/template/merged/event/ directory. These templates can be overridden by files you place in the *INSTALL_DIR*/extensions/global/template/ directory.

The naming convention for templates is BaseTxnName.eventName.xml. For example, the on_success event of the createOrder() API uses the ORDER_CREATE.ON_SUCCESS.xml event template.

**Note:** Templates are not supported for user exits.

# Chapter 6. DTDs, XSDs, and Complex Queries

## DTD and XSD Generator

Every Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales API uses standard input, output, and error XMLs. These XMLs conform to the related Document Type Definition (DTD). For example, consider the following XML:

```
<?xml version="1.0" encoding="UTF-8">
<Order EnterpriseCode="DEFAULT" OrderNo="S100" />
```

The corresponding DTD for this XML is:

```
<!ELEMENT Order>
<!ATTLIST Order OrderNo CDATA #IMPLIED>
<!ATTLIST Order EnterpriseCode CDATA #REQUIRED>
```

To create such DTDs for the extended Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales XML, a tool called the xsdGenerator.xml is provided in the *INSTALL_DIR*/bin directory. This tool converts a specially-formatted XML file into a DTD and XML schema definition (XSD). The command for running the tool is:

```
sci_ant.sh -f xsdGenerator.xml generate
```

You can also pass the following properties as command line arguments:

- `xsdgen.use.targetnamespace`
- `xsdgen.use.datatypeimport`

For example,

```
sci_ant.sh  -Dxsdgen.use.targetnamespace=N
-Dxsdgen.use.datatypeimport=N -f xsdGenerator.xml generate
```

The following table contains information about the XSD Generator properties:

| Fields | Description |
|---|---|
| xsdgen.use.targetnamespace | Optional. The default value is Y. If set to Y, the XSD files are generated with a defined target namespace. |
| xsdgen.use.datatypeimport | Optional. The default value is Y. If set to Y, all the XSD files reference a single common XSD file containing all the common data type definitions. If set to N, each XSD file is created with a copy of the database definitions embedded within it. |

The input XML files should be placed in the *INSTALL_DIR*/xapidocs/extn/input directory. The resulting DTD and XSD files are placed in the *INSTALL_DIR*/xapidocs/extn/output/dtd and *INSTALL_DIR*/xapidocs/extn/output/xsd directories respectively.

**Note:** When the xsdgen.use.datatypeimport is set to 'Y' it will generate the updated datatypes.xsd file in the <INSTALL_DIR>/xapidocs/extn/output/xsd directory based on the merged datatypes.xml including the data type extensions.

Consider the following sample XML that could be placed in the input directory and converted to an XSD and DTD:

```
<Item yfc:DTDOccurrence="REQUIRED" ItemKey="" ItemID="REQUIRED"
OrganizationCode="REQUIRED" UnitOfMeasure="">
   <PrimaryInformation Description="" ItemType="" />
   <AdditionalAttributeList>
        <AdditionalAttribute Name="" Value=""/>
    </AdditionalAttributeList>
    <Extn ExtnAttr1="" ExtnRefId="">
       <CSTItemDataList yfc:DTDOccurrence="ZeroOrOne">
         <CSTItemData yfc:DTDOccurrence="ZeroOrMany" ItemDataKey=""
Description="">
            <CSTItemExtraData yfc:DTDOccurrence="ZeroOrOne" CodeType=""
DataType="" />
            <YFSCommonCode yfc:DTDOccurrence="REQUIRED" CodeName=""
CodeType="" CodeValue="" />
         </CSTItemData>
       </CSTItemDataList>
    </Extn>
</Item>
```

The following table contains descriptions of special attributes for XML:

| Fields | Description |
|---|---|
| yfc:QryTypeSupported | This attribute determines whether or not the query type functionality is supported for the attributes in this element. If set to Y, it takes effect for all the elements. |
| yfc:ComplexQuerySupported | This attribute specifies whether or not a complex query type is supported. This attribute can only be present in the root element. |
| yfc:XSDType | The name of the type to use for the root element schema definition. |
| yfc:DTDOccurrence | This attribute can contain any of the following values:<br>• REQUIRED - This element must be present if the parent element is present.<br>• ZeroOrOne - This element is optional, but may occur only once.<br>• ZeroOrMany - This element is optional, but may occur multiple times.<br>• OneOrMany - This element is required, and may occur multiple times. |
| yfc:UseEntityOrdering | This attribute determines whether or not all the first-level children of an element are ordered in the sequence they are found in the entity xmls. This attribute can contain any of the following values:<br>• true - All the first-level children of an element are ordered in the sequence they are found in the entity xmls.<br>• false - The first-level children of an element are not ordered in the sequence they are found in the entity xmls. |
| xmlns | The namespace to use for the targetNameSpace in the output XSD. This attribute takes effect only if it is present in the root element. |

The attributes with values of `REQUIRED` are generated as required attributes in the DTD and XSD. However, an existing required attribute cannot be marked as optional.

The attribute values can also be specified to supply additional constraints. A list of options is separated by a vertical bar (`|`). The value of the attribute must be one of the given options. This is only supported for data types based on the strings. The values are trimmed of the whitespace character if the value itself is entirely spaces, in which case the enumerated option remains unchanged.

For example, `SomeAttr="A | B | C | |"` results in valid options of "A", "B", "C", " ", and "".

**Note:** The DTDs do not support enumerated values containing only whitespace characters. Therefore, restrictions of this type cannot be represented in the DTD.

The default input and output XMLs that can act as a base for your custom XML are located in the *INSTALL_DIR*/xapidocs/xmlstruct/ directory. Also note that the DTDOccurrence and REQUIRED data provided for the standard tables are inferred from the base file in the xmlstruct directory and do not need to be supplied. If they are provided, the existing information is overridden by any new information present in the custom XMLs. Any required datatype and relationship information are obtained from the entity XMLs.

**Note:** Do not put your custom XMLs in the xmlstruct directory.

Therefore, when the tool is run these base XML files serve as a default to your custom XML files, which need only contain the changes made by you such as the extended elements and attributes. This allows future upgrades to safely modify the XML files in the xmlstruct directory. Re-running the XSD generation tool automatically picks up these updates.

The appropriate XML file in the xmlstruct directory associated with your custom XML is identified by the file name. Your custom XML may start with an optional prefix followed by an under-score and the base file name. For example, a custom XML file named Custom_File_YFS_getOrderDetails_input.xml refers to the YFS_getOrderDetails_input.xml file in the xmlstruct directory.

However, the naming convention is optional. For example, you can also name your custom XML sampleCustomApi.xml but no base file is used. In this case, the tool outputs an informational message to indicate that no base XML is found.

**Note:** If you want to use our base XML file for conversion, the naming convention of your custom XML must be suffixed appropriately. For example, Custom_File_YFS_getOrderDetails_input.xml would use the base file named YFS_getOrderDetails_input.xml.

The generated XSD specifies the target namespace as shown below:

```
<xsd:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
targetNamespace="http://www.sterlingcommerce.com/documentation"
xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:yfc="http://www.sterlingcommerce.com/documentation">
```

This namespace is picked up from the xmlns attribute on the root element of the input XML and defaults to http://www.sterlingcommerce.com/documentation.

The XSD and DTD files contain query type attributes used in list APIs when `QryTypeSupported="Y"` is set in the root element of the input XML. Similarly, the complex query types defined for getItemList() and getOrganizationList() APIs are represented in the XSD and DTD files when `ComplexQuerySupported="Y"` is set.

However, in APIs the following exceptions are exhibited in the DTDs since these constraints cannot be represented in a pure DTD, XSD or both:

- If an XML contains multiple Extn attributes, the generated DTD-only (not generated XSD) defines a single Extn element which appears as the union of all possible Extn elements.
- Conditionally required attributes. For example, you need to specify a group of attributes or another group of attributes such as OrderHeaderKey or EnterpriseCode/OrderNo.
- Mandatory condition of a node depends on some attribute value. For example in the createOrder() API, the OrderLine node is required if the `DraftOrderFlag="N"`.

## Defining Complex Queries

Complex queries help to narrow a detailed listing obtained as output from an API. To generate the desired output, you can pass queries using `And` or `Or` operators in the input XML of an API.

For example, you can query the getItemList API based on the unit of measure, item group code or any parameters provided in the API definition, using the complex query operators, `And` or `Or`.

Complex queries are supported for the following APIs:

- deletePricelistAssignmentList
- deletePricingRuleAssignmentList
- getAttributeAllowedValueList
- getClassificationPurposeList
- getCustomerContactList
- getInventoryReservationList
- getItemList
- getOrderLineList
- getOrderList
- getOrganizationList
- getSearchIndexTriggerList
- getShipmentList

**Note:** Only item, organization, order, order line, shipment and shipment line entities are supported for performing complex queries. The attributes for complex query must map directly to valid database columns of these entities and should be within the same XML element.

For more information about these APIs, see the Javadocs. For more information on valid database columns, see the Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales ERDs.

### Example: Adding Complex Queries in getItemList API

Consider the following scenario for adding complex queries to the getItemList API.

The getItemList API returns a list of items based on the selection criteria specified in the input XML such as item attributes, aliases, category, and so on. You can create complex queries in the getItemList input XML as shown in the following example:

```
<Item OrganizationCode="DEFAULT" ItemGroupCode="PS" >
  <PrimaryInformation PricingQuantityStrategy="IQTY">
    <ComplexQuery Operator="OR">
       <And>
          <Or>
            <Exp Name="UnitOfMeasure" QryType="ISNULL"/>
            <Exp Name="UnitOfMeasure" Value="HR" QryType="FLIKE"/>
          </Or>
        <And>
             <Exp Name="ManufacturerName" Value="XYZ"/>
        </And>
        </And>
    </ComplexQuery>
  </PrimaryInformation>
</Item>
```

`OrganizationCode` and `ItemGroupCode` are the two attributes of the `<Item>` element and `PricingQuantityStrategy` is the attribute of the `<PrimaryInformation>` element considered in this example. However you can include any or all of the attributes in the getItemList API. All the attributes in the API are interpreted with an implied `And` along with the complex query operator.

Apply the following rules when including complex queries:

- You can define only one ComplexQuery under a single element. For example, you cannot have two ComplexQuery operator under an Item element.
- You cannot add a single complex query against two different tables. For example, in getShipmentList API you cannot use ChainedFromOrderHeaderKey and ShipmentLineNo in the same query, since the former belongs to YFS_ORDER_LINE table and the latter is an attribute of the YFS_SHIPMENT_LINE table.
- The attribute with no value is not considered in the complex query, like `Attribute=""`.
- For attributes appended with `QryType`, specify a query type value from the folllowing table. This is case sensitive.

The values for the QryType attributes vary depending on the datatype of the field. The following table lists the supported query type values used by List APIs for each datatype.

| Field DataType | Supported Query Type Values |
|---|---|
| Char/VarChar2 | • EQ - Equal to<br>• FLIKE - Starts with<br>• LIKE - Contains<br>• GT - Greater than<br>• LT - Less than |

| Field DataType | Supported Query Type Values |
|---|---|
| Number | • BETWEEN - Range of values<br>• EQ - Equal to<br>• GE - Greater than or equal to<br>• GT - Greater than<br>• LE - Less than or equal to<br>• LT - Less than<br>• NE - Not equal to |
| Date | • DATERANGE - Range of dates<br>• EQ - Equals<br>• GE - Greater than or equal to<br>• GT - Greater than<br>• LE - Less than or equal to<br>• LT - Less than<br>• NE - Not equal to |
| Date-Time | • BETWEEN - Range of dates<br>• EQ - Equals<br>• GE - Greater than or equal to<br>• GT - Greater than<br>• LE - Less than or equal to<br>• LT - Less than<br>• NE - Not equal to |
| Null | • ISNULL - Return records that are null.<br>• NOTNULL - Return records that are not null.<br>**Note:** These two query types are used when the column or attribute is set to Nullable in the entity XML. |

- There can be only one element under the ComplexQuery namely, `And` or `Or`.
- `And` or `Or` elements can have one or many child elements as required.
- `And` or `Or` elements can have other `And` or `Or` expression elements as child elements.

This example can be interpreted as the following logical expression:

```
(OrganizationCode="DEFAULT" AND ItemGroupCode="PS") AND
((PricingQuantityStrategy="IQTY") OR ( ( UnitOfMeasure = "EACH"
OR UnitOfMeasure="HR" ) AND ( ManufacturerName = "XYZ") ))
```

By following the above example you can include complex queries to achieve desired results from your database using the above mentioned APIs.

**Note:** Use of Complex Query in conjunction with Case Insenstive search for a column is not supported. However this can be achieved by using the shadow column in Name Attribute of Exp element of complex query. For example, if the shadow column for ORGANIZATION_CODE in table YFS_ORGANIZATION is ORGANIZATION_CODE_LC, then getOrganizationList api can be called with below input to perform case insenstive search on OrganizationCode whose value is either org1 or org2.

```
<Organization OrganizationCode="org1" OrganizationCodeQryType="LIKE">
   <ComplexQuery Operator= "OR"
       <Or>
<Exp Name="OrganizationCodeLC" QryType="LIKE" Value="org2"/>
   </ComplexQuery>
</Organization>
```

# Chapter 7. Creating Extended APIs

## Invoking Extended APIs

### About this task

Extended APIs are APIs that you provide; they are sometimes called custom APIs. You can use an extended API to invoke a Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales API or third-party API, as well as to perform custom processing through the Service Definition Framework.

To invoke an extended API:

### Procedure

1. Code a class.
2. Code a function that has exactly two parameters of types `YFSEnvironment` and `Document` and ensure that the function returns a document.

   ```
   public Document <method-name> (YFSEnvironment env, Document doc)
   ```

3. Configure a service that contains an API node. When configuring an API node, use the properties described in the following table.

| Property | Description |
|---|---|
| **General Tab** | |
| Extended API | Select this option if a custom API is to be invoked. |
| API Name | Select or enter the API to be called.<br><br>**Note:** This field is for integration purposes only. |
| Class Name | Specifies the class you coded in the first step. |
| Method Name | Specifies the function to be called as coded in the previous step. |
| **Arguments Tab** | |
| Argument Name | You can pass name/value pairs to the API by entering the values in the Arguments Tab.<br><br>In order for custom APIs to access custom values, the API should implement the interface `com.yantra.interop.japi.YIFCustomApi`.<br><br>If entered, these name/value pairs are passed to the custom API as a properties object. |
| Argument Value | Enter the argument value. |
| **Template Tab** | |
| XML Template | Select this radio button to construct the XML to be used for the API output. Enter the template root element name and click OK. You can then construct the XML. |
| File Name | Select this radio button to enter the filename of the XML file to be used as the API output template. This file should also exist in your CLASSPATH. |

| Property | Description |
|---|---|
| **Facts Tab**<br><br>You can configure the Fact Lookup for Database and custom APIs by using the Facts tab. You can define Name-Value pairs for Fact lookup. The Value can be an XML Path. | |
| Fact Name | Enter the fact name. |
| Fact Value | Enter the fact value. |

When connecting the nodes within a service, keep in mind the API node connection properties as listed in the following API Node Connection Properties table:

| Connection | Node Connection Rules |
|---|---|
| Can be the first node after the start node | Only for services invoked synchronously |
| Can be placed before | • Any transport node except FTP or File IO<br>• Any other component node |
| Can be placed after | • Start node<br>• Any transport node except FTP or File IO<br>• Any other component node |
| Passes data unchanged | Yes |

4. Make sure the class is in the CLASSPATH of the Service Definition Framework.
5. Make sure that the class implements a method with a signature that takes in exactly two parameters, a `YFSEnvironment` and a `Document.`

   The following example shows how to implement a class:

```
import com.yantra.yfs.japi.YFSEnvironment;
import org.w3c.dom.Document;
public class Bar {
  public Bar () {
    }
  public Document foo(YFSEnvironment env, Document doc)
    {
    //write your implementation code here
    }
}
```

6. To access the extended API you created, invoke the service containing your extended API.

   For details and sample code that show how to access properties specified when the custom API is configured, see the YIFCustomAPI interface in the Javadocs.

## Implementing the Error Sequence User Exit

You can configure the Service Definition Framework to call a user exit that checks for prior errors for the exception group to which the API belongs. This user exit is called before any processing of the message starts. A Java interface is supplied for its implementation. This interface definition is in the com.yantra.interop.japi.YIFErrorSequenceUE class. The user exit computes the Message Key based on user defined custom code.

YIFErrorSequenceUE defines two functions. The function definitions are:

```
1) public Document getExceptionGroupReference(Document document, String apiName)
throws Exception
2) public void setExceptionGroupFinder (YIFExceptionGroupFinder finder)
```

The getExceptionGroupReference() function takes two parameters:

- Document - The input XML document retrieved by the Integration Adapter
- String - The API for which the Integration Adapter retrieved the XML

The setExceptionGroupFinder() function sets the YIFExceptionGroupFinder() interface. Use the implementation of this interface to retrieve the exceptionGroupId if prior errors exist.

An example implementation of this function is:

```
public void setExceptionGroupFinder (YIFExceptionGroupFinder finder){
   this.finder = finder;
}
```

## Implementing the YIFExceptionGroupFinder Interface

This interface defines the findExistingError() function that takes in `Document` as the input parameter.

For example, the input XML document that the user exit passes to the findExistingError() function would contain:

```
<?xml version="1.0"?>
<ExceptionGroupReference messageKey="xyz"/>
```

## Exception Handling in Extended APIs

The client always has the option of throwing an exception to the Service Definition Framework instead of handling it when it occurs. Depending on the configuration, the Service Definition Framework either sends the exception to the Alert Console or logs the exception.

## Locking Records in Extended APIs

You can lock a record in a custom table when executing a custom entity API in Service Definition Framework (SDF). To lock a record, you must pass the SelectMethod attribute as part of the input XML to the custom entity API. The locking happens within the transaction boundary of the custom API call.

The value of the SelectMethod attribute will determine what (if any) type of locking to be used. You can pass one of the following values for the SelectMethod attribute:

- WAIT—The record is locked for SELECT FOR UPDATE operation.
- NO_WAIT—The record is locked for SELECT FOR UPDATE NOWAIT operation.
- NONE—Locking mechanism is not used.

**Note:** If you pass any other value for the SelectMethod attribute, an error is thrown indicating that the "SelectMethod" attribute value is not valid.

**Note:** If the SelectMethod attribute does not exist or if it is set to NONE in the input XML, the locking mechanism is not used.

# Chapter 8. Invoking APIs and Services

## Invoking APIs from the Client Environment

In order to call standard APIs from the client, ensure that the client environment is set up correctly. The client environment must have appropriate CLASSPATH settings and JAR files as described in this section for a basic configuration. SSL, JNDI, or other security requirements may affect your configuration.

**Note:** It is recommended that you do not invoke a local API in the application JVM before the server initialization. Also, if you are making a local API call, you must add the following code after the local API invocation in the block that has YIFClientFactoryImpl.getLocalApi and api.invoke:

```
YFCRemoteManager.setIsLocalInvocation(false);
```

The *INSTALL_DIR*/resources/ directory must contain the yifclient.properties file.

If you are calling in local mode, the client CLASSPATH must contain all JAR files referred to in the *INSTALL_DIR*/properties/dynamicclasspath.cfg file.

When invoking APIs through EJB or HTTP, the client CLASSPATH must contain the following files in *WAS_HOME*/AppClient/properties directory:

- xapi.jar
- log4j-1.2.15.jar
- platform_afc.jar
- xercesImpl.jar
- xml-apis.jar
- ejbstubs.jar (or equivalent .jar file containing EJB stubs).

The client CLASSPATH must also contain the following files from *INSTALL_DIR*/jar/:

- install_foundation.jar
- smcfs/9.1/smcfsshared.jar
- platform_afc_demo.jar (for demonstration application)

### Application Server-specific Files

In addition to the files listed above, the following files are required:

| Application Server | Required Files |
|---|---|
| IBM WebSphere | For WebSphere, use *ejb.jar available in the EAR file to retrieve the ejbstubs.jar.<br>• j2ee.jar<br>• com.ibm.ws.ejb.thinclient_*version*.jar<br>• com.ibm.ws.orb_7.0.0.jar<br>• com.ibm.ws.sib.client.thin.jms_*version*.jar<br>• com.ibm.ws.wcom.jar |

| Application Server | Required Files |
|---|---|
| Oracle WebLogic | • wlfullclient.jar<br>• Consult your application server documentation for other CLASSPATH requirements. |
| JBoss | The CLASSPATH should include entries for the following jar files:<br>• log4j-1.2.15.jar<br>• jboss-j2ee.jar<br>• jnpserver.jar<br>• jboss-common-client.jar<br>• concurrent.jar<br>• jboss.jar<br>• jboss-serialization.jar<br>• jboss-remoting.jar<br>• jbosssx.jar<br>• jboss-transaction.jar |

## Invoking Services and Standard APIs Programmatically

Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales provides sample code that demonstrates how the standard APIs and services of the application can be invoked programmatically. See the sample files in the *INSTALL_DIR*/xapidocs/code_examples/ directory.

**Note:** Use the executeFlow() method of the YIFApi interface to run a service defined within the Service Definition Framework.

API and service transactions that are outbound from the application can be configured through the Service Builder, as described in the Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales: *Configuration Guide Application Platform Configuration Guide*.

API and service transactions that are inbound to Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales can be invoked through the following protocols:

- EJB
- HTTP and HTTPS
- LOCAL
- Web Services
- COM+

### EJB

Use EJB for server-side execution of the code. Java call. All the methods in Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales take a YFSEnvironment and a document, and return a document. Since EJBs are designed to be called remotely, each of these documents is serialized on one end and unserialized on the other. However, the application uses an EJB, where each API takes two string parameters and returns a string. Thereby, forcing any document implementation to serialize and unserialize using a standard well-defined interface.

For example, a new EJB is created with method signatures like:

```
String createOrder(String env, String inputXML) throws YFSException, RemoteException;
```

where `env` is an XML that should be a valid input to `createEnvironment` variable. The return value is the output XML.

When calling an API using YIFClientFactory.getInstance().getApi("EJB") the call is made using this String-based EJB. With this type of call you can pass a YFSEnvironment and document, and get a document in return. The Sterling Business CenterSterling Selling and Fulfillment FoundationSterling Field Sales code performs the conversion transparently.

**Note:** The DOM-based EJB is deprecated. Hence, moving forward you need to use the String-based EJB for server-side execution.

### HTTP

Use HTTP for server side execution of code. Java call.

### LOCAL

Use Local for client side execution of code. COM or Java call.

### Web Services

Use Web Services for client side execution of code. COM or Java call.

### COM+

Use COM for client side execution of VB or C++ code. COM or Java call.

Using COM requires setting up your server and runtime clients.

**Note:** Exceptions encountered when making synchronous API calls through EJB, COM, or HTTP transport protocols are not queued for reprocessing.

## Configuring Service Invocation

### About this task

To configure service invocation:

### Procedure

1. Rename the *INSTALL_DIR*/resources/yifclient.properties.in file to *INSTALL_DIR*/resources/yifclient.properties.
2. Ensure that the CLASSPATH contains the following:
   - log4j-1.2.15.jar
   - xercesImpl.jar
   - install_foundation.jar
   - platform_afc.jar
   - resources.jar
   - entities.jar
   - xapi.jar

- JARs requried by your application server
- JARs required by user exits and custom APIs

3. Set your java command line property to:

   ```
   -Dlog4j.configuration=resources/log4jconfig.xml
   ```

4. Make sure that the *INSTALL_DIR* directory is in your CLASSPATH.
5. Set the log4j properties in the log4jconfig.xml file to the appropriate values for your environment. If these properties are not specified correctly, the Service Definition Framework does not initialize correctly.
   - If you are using the EJB protocol and Oracle WebLogic, make sure that `weblogic.jar` is in your CLASSPATH environment variable. In addition, `xercesImpl.jar` and `xalan.jar` must precede `weblogic.jar` in your CLASSPATH.
   - If you are using the EJB protocol and JBoss, make sure that *JBOSS_HOME*/client/jbossall-client.jar is in your CLASSPATH environment variable.
   - If you are using the EJB protocol and IBM WebSphere, make sure that the CLASSPATH environment variable contains the necessary JAR files. For information about the WebSphere JAR files, see the IBM documentation. Make sure that the CLASSPATH environment variable contains the appropriate properties directory.

     **Note:** If you are invoking the service or API from the machine on which the server is running, make sure that the CLASSPATH environment variable contains the *WAS_HOME*/AppServer/properties/ directory.

     If you are invoking the service or API from a different machine, make sure that the CLASSPATH environment variable contains the *WAS_HOME*/AppClient/properties/ directory.
   - If you are configuring a COM+ protocol call, use one of the following COM signatures that you need:

     ```
     createEnvironment(VARIANT *lEnvHandle, BSTR sProgID,
     BSTR sUserID, int *iRetval)
     ```

     Signature for calling standard APIs:

     ```
      <SterlingAPI>(VARIANT *lEnvHandle, BSTR inXML,
     VARIANT *outXML, VARIANT *errXML, int *retval)
     ```

     Signature for calling services:

     ```
      executeFlow(VARIANT *lEnvHandle, BSTR flowName,
     BSTR flowMsg, VARIANT *outXML, VARIANT *errXML, int *retval)
     ```

     For examples of VB code, see the samples in the *INSTALL_DIR*/xapidocs/code_examples/complus directory.

# Directing API Calls to Specific Servers

## About this task

The application provides the ability to route custom API calls to a particular server or group of servers when these APIs are invoked either remotely or locally.

To enable this, the server(s) and the protocol must be specified in the yifclient.properties file and in the required APIs. The endpoint is the configured server(s) or a protocol which is used for routing the API calls.

If the XAPI client is configured for multiple URLs, an attempt to connect is made to each server in the group in turn. If the connection works but something else fails, no other server is tried and an exception is thrown.

To direct API calls to a specific server or group of servers, perform the following steps:

## Procedure

1. Specify the `endpoint` attribute in the yifclient.properties file under the directory *INSTALL_DIR*/resources/. Modify the yifclient.properties file to include the declaration and usage of endpoint in the following format:

   ```
   endpoint.Server_Name.apifactory.protocol=HTTP
   endpoint.Server_Name.httpapi.url=http://server:port/context_root/
   interop/InteropHttpServlet
   ```

   To configure the XAPI client for multiple URLs, define the group as a comma-separated list:

   ```
   endpoint.Server_Name.httpapi.url=http://server1:port1/context_root1/
   interop/InteropHttpServlet,http://server2:port2/context_root2/
   interop/InteropHttpServlet...
   ```

   The expression `endpoint.Server_Name` specifies a server with the name *Server_Name*. For example, `endpoint.INBOXSERVER` creates an endpoint with the name INBOXSERVER. You can assign properties to the endpoint name, which takes precedence over any other assigned property.

   The protocol configured for the endpoint is HTTP as specified in the line: `endpoint.Server_Name.apifactory.protocol=HTTP`

   The settings specified in the second line, `endpoint.Server_Name.httpapi.url`, are used to connect to the server specified in the endpoint.
2. Configure a protocol to be used for connecting to the specified server. The protocols such as HTTP, HTTPS, EJB, LOCAL, AUTO are reserved endpoint names. If any of these are configured for the endpoint, the system uses the default connection settings (as applicable) for routing the API calls.
3. For each API, specify the endpoint to be used:

   ```
   yfs.api.apiname.endpoint=ENDPOINT
   ```

   The API then calls the server(s) specified in the endpoint attribute.

   **Note:** If an endpoint is not configured for an API, then it uses the default (local) server or a general one configured for all APIs. The property set at the API level takes precedence over other common properties.

# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing*

*IBM Corporation*

*North Castle Drive*

*Armonk, NY 10504-1785*

*U.S.A.*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing*

*Legal and Intellectual Property Law*

*IBM Japan Ltd.*

*19-21, Nihonbashi-Hakozakicho, Chuo-ku*

*Tokyo 103-8510, Japan*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be

incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*IBM Corporation*

*J46A/G4*

*555 Bailey Avenue*

*San Jose, CA 95141-1003*

*U.S.A.*

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© IBM 2014. Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 2014.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

## Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at http://www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium and the Ultrium Logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Connect Control Center®, Connect:Direct®, Connect:Enterprise®, Gentran®, Gentran®:Basic®, Gentran:Control®, Gentran:Director®, Gentran:Plus®, Gentran:Realtime®, Gentran:Server®, Gentran:Viewpoint®, Sterling Commerce™, Sterling Information Broker®, and Sterling Integrator® are trademarks or registered trademarks of Sterling Commerce®, Inc., an IBM Company.

Other company, product, and service names may be trademarks or service marks of others.

Product Number: xxxx-xxx

Printed in USA