Introducing InterSystems Supply Chain Orchestrator

Version 2024.1
2024-07-02

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

Introducing InterSystems Supply Chain Orchestrator
InterSystems IRIS Data Platform Version 2024.1 2024-07-02

For Support questions about any InterSystems products, contact:

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

Table of Contents

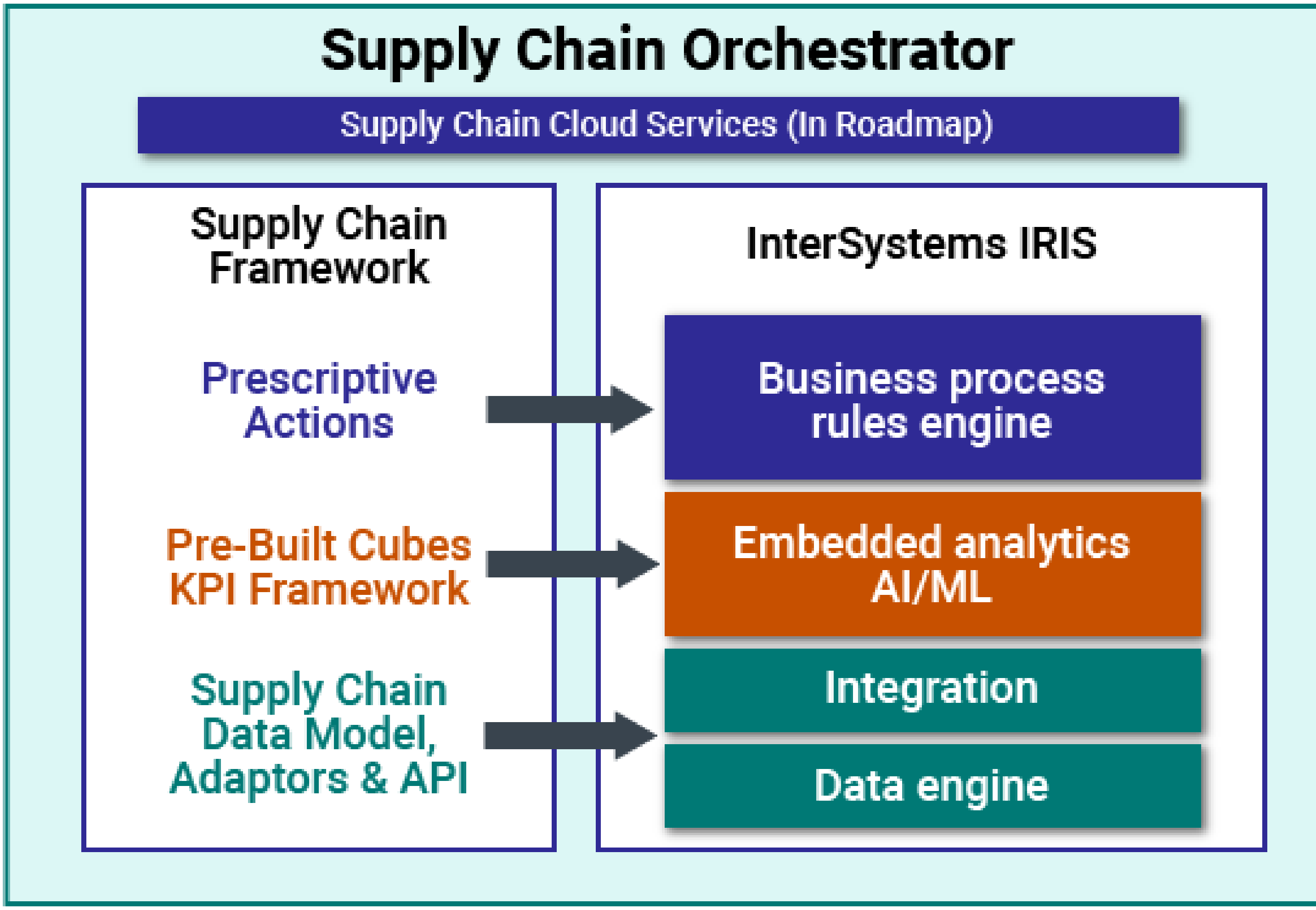
1

Introduction to InterSystems Supply Chain Orchestrator

InterSystems Supply Chain Orchestrator™ provides an intelligent data platform together with a set of supply-chain-specific cloud services to provide end-to-end visibility and unmatched predictive and prescriptive capabilities that transform supply chain performance and agility.

The core part of Supply Chain Orchestrator is named InterSystems IRIS for Supply Chain, which consists of InterSystems IRIS® and the separately released InterSystems Supply Chain Framework. As shown in the figure below, the Supply Chain Framework extends key parts of InterSystems IRIS.

In addition to this core, a series of cloud services are planned. Customers and partners can also create and market their own cloud services to interoperate with this ecosystem.


Supply Chain Orchestrator
Supply Chain Cloud Services (In Roadmap)
Supply Chain Framework
Prescriptive Actions
Pre-Built Cubes KPI Framework
Supply Chain Data Model, Adaptors & API
InterSystems IRIS
Business process rules engine
Embedded analytics AI/ML
Integration
Data engine

1.1 Purpose of Supply Chain Orchestrator

Supply Chain Orchestrator (specifically the core part, IRIS for Supply Chain) is designed to meet the following objectives:

- Provide the basic infrastructure required for most supply chain problems, such as supply-chain-wide KPIs and issue management.
- Provide common supply chain features, such as supply chain data model and related analytics cubes.
- Provide a set of APIs to access supply chain capabilities, which can be easily used for external consumption such as UI development.
- Provide a set of utilities which make the adoption of the framework easy. For example, messaging capabilities and helper classes are provided to simplify business process design to achieve no-code or low-code development.

Supply Chain Orchestrator is *not*:

- A supply chain solution to replace any existing supply chain operation applications, such as TMS, WMS, or OMS. The product does, however, make it easy to connect data and/or processes from these systems, bridge data visibility gaps, and provide intelligent actions to minimize risk impacts on the overall supply chain.

- A predefined set of solutions for specific use cases, such as business processes or business rules for any use case, or KPIs for any part of the supply chain. The product does provide simple mechanisms for you to define your own use case specific solutions with minimum or no coding effort.
- An end-user application for supply chain practitioners to interact with directly. The solution or outcome from the framework is expected to be exposed to other applications, such as a control tower UI, a BI tool, or other supply chain applications.

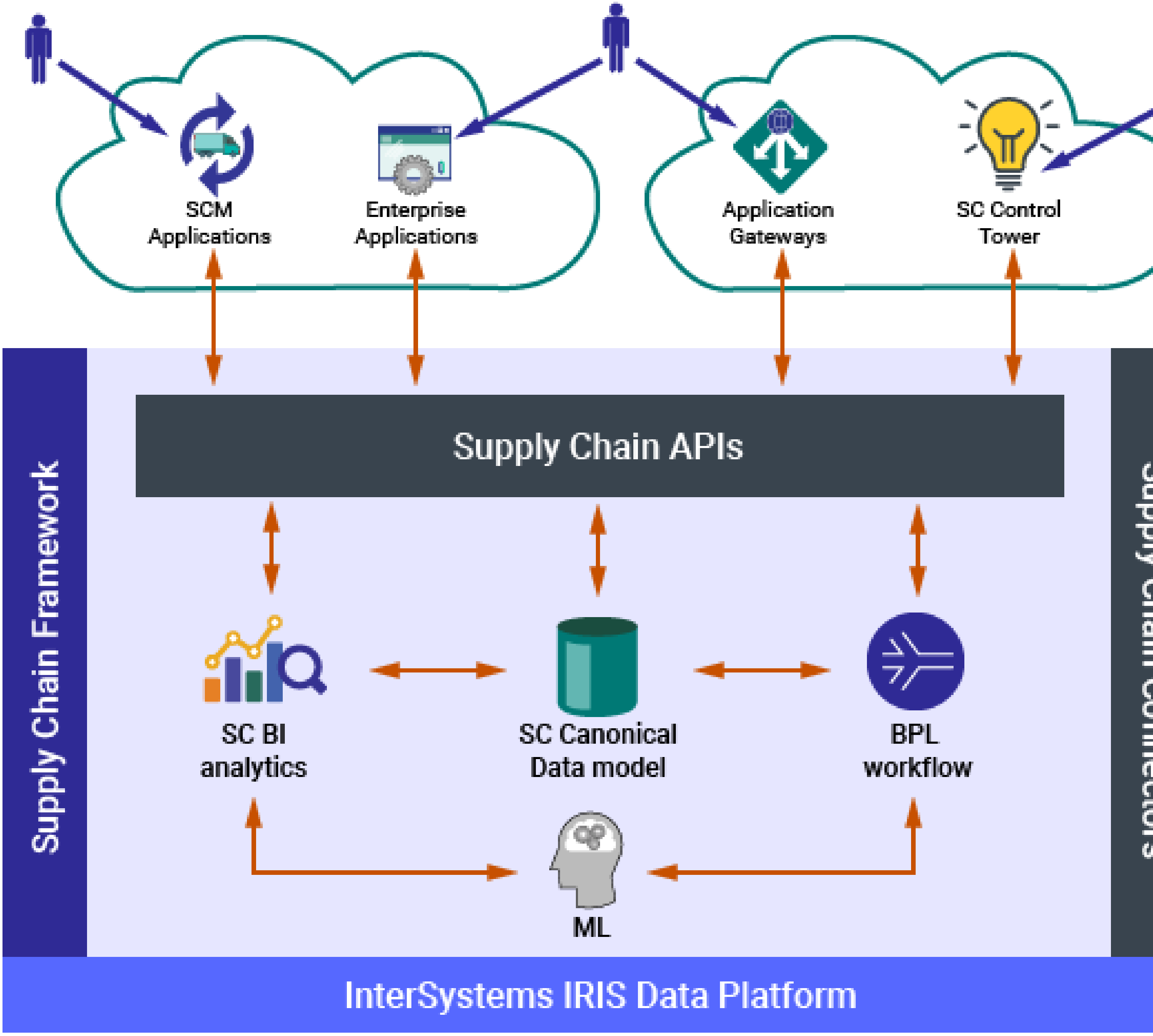
1.2 Features

Supply Chain Orchestrator (specifically the core part, IRIS for Supply Chain) provides the following features:

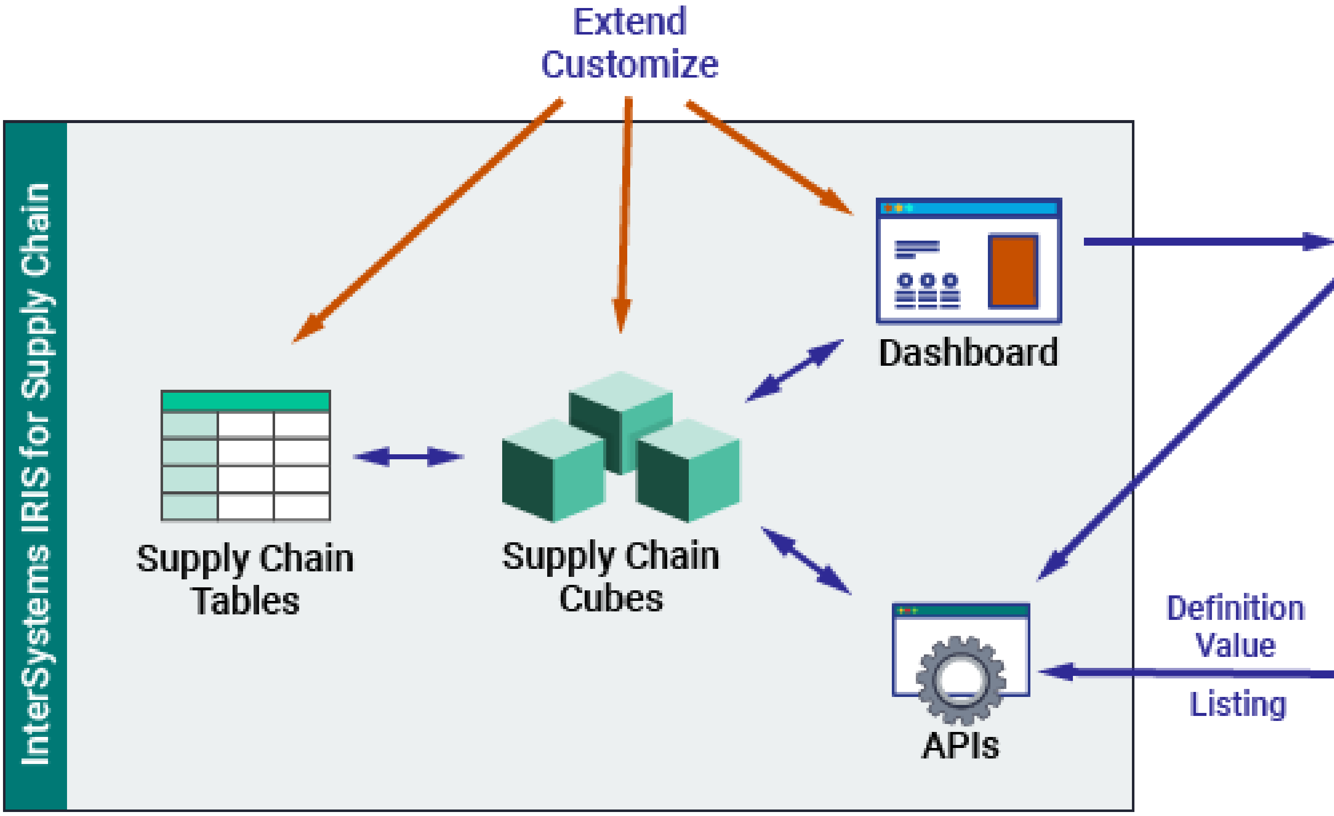
- Extensible supply chain data model, a canonical supply chain data model implemented on InterSystems IRIS data platform, which can be extended/customized using either ObjectScript code or APIs.
- Data model APIs for data model discovery and live documentation, such as listing all supply chain data objects, or getting the detailed definition of a supply chain data object.
- Data access APIs, which support both CRUD operations and advanced search capabilities with sorting and pagination support.
- Analytics cubes, prebuilt Business Intelligence cubes based on the supply chain data model. The built-in cubes can be extended as needed, for example to include custom attributes.
- KPI framework and related APIs, support for KPI management (KPI configurations and life cycle management) and KPI values and listings access. The current KPI framework is based on the predefined analytics cubes.
- Issue life cycle management, the component which is responsible for issue identification, issue analysis, and action recommendations.

1.3 Architecture Overview

The core of Supply Chain Orchestrator is InterSystems IRIS for Supply Chain (an InterSystems IRIS® server with the addition of the InterSystems Supply Chain Framework), as shown above. The following figure shows more detail:



The next picture shows the analytics support in more detail:



1.4 Available Online Learning

In addition to the documentation website, see the online learning path Building Smart Real-Time Supply Chain Applications, in which you will explore use cases and architecture of this platform and learn how to install and start implementing it. You will also learn to create KPIs and use business logic for issue resolution.

2

Installation and Upgrade

A supply chain application or solution built on InterSystems Supply Chain Orchestrator™ consists of the following components:

- Core InterSystems IRIS® data platform
- InterSystems Supply Chain Framework features delivered through InterSystems Package Manager (IPM)

  Together with InterSystems IRIS, this forms the core of Supply Chain Orchestrator.
- Application or solution implementation code

The installation and upgrade process depends on which components are being installed or updated. This page provides detailed instruction for different scenarios.

Also see the online learning path Building Smart Real-Time Supply Chain Applications.

2.1 New Installation

To deploy a new instance, there are three steps:

1. Deploy InterSystems IRIS data platform.
2. Deploy InterSystems Supply Chain Framework, thus creating the core of Supply Chain Orchestrator.
3. Deploy application code.

2.1.1 InterSystems IRIS Deployment

There are different ways to deploy InterSystems IRIS: in containers, on Kubernetes, using installation kits, and so on. For more details on each approach, see Deployment.

2.1.2 Adding the Supply Chain Framework

After deploying an InterSystems IRIS server, use the following steps:

1. Create a new namespace reserved for use by Supply Chain Orchestrator. This documentation uses the namespace name `SC`, but you can use a different name.

   Note: Make sure that the namespace is interoperability-enabled and analytics-enabled. The latter option enables you to use the Business Intelligence Analyzer so that you can explore the cubes.

2. Start the Terminal application and complete the rest of the steps in the Terminal.

   Or on Unix®, where the Terminal application is not available, start the ObjectScript shell.

3. Change to the newly created namespace.

4. Check to see if IPM is installed. To do so, enter the command **zpm**. If IPM is not installed, this command results in an error. If IPM is installed, the Terminal prompt then looks like this:

   Terminal

   ```
   zpm:SC>
   ```

   Where the part after the colon is the namespace in which you are working.

5. If InterSystems Package Manager (IPM) is not installed, install it as follows:

   ```
    set r = ##class(%Net.HttpRequest).%New()
    set r.Server="pm.community.intersystems.com",
    set r.SSLConfiguration="ISC.FeatureTracker.SSL.Config"
    do r.Get("/packages/zpm/latest/installer"),
    do $system.OBJ.LoadStream(r.HttpResponse.Data,"c")
   ```

   Tip: For convenience, the following single line of code represents all the preceding lines.

   ```
   set r =
   ##class(%Net.HttpRequest).%New(),r.Server="pm.community.intersystems.com",r.SSLConfiguration="ISC.FeatureTracker.SSL.Config"
    d r.Get("/packages/zpm/latest/installer"),$system.OBJ.LoadStream(r.HttpResponse.Data,"c")
   ```

   Then start IPM. To do so, enter the command **zpm**

6. If you do not yet have an IPM token, login to Pm.InterSystems.com using your InterSystems account credentials and obtain a token.

7. At the IPM prompt, use the following command to log in to the IPM repository:

   ```
   repo -n registry -r -url https://pm.intersystems.com/ -token YOUR_IPM_TOKEN
   ```

   Where *YOUR_IPM_TOKEN* is the IPM token from the previous step.

8. Still at the IPM prompt, install InterSystems Supply Chain Framework. To install the latest framework:

   ```
   install isc-supply-chain
   ```

   Or to install a specific version of the framework (in this example version 1.0.0):

   ```
   install isc-supply-chain 1.0.0
   ```

9. After the installation is complete, enter `q` to exit the IPM prompt.

2.1.3 Application Deployment

Any application code can be deployed using DevOps tools. Application code may include (but not limited to):

- Extensions to the supply chain data model
- Custom analytics cube design
- Business processes
- Any DTL implementations
- Custom APIs

- Others

2.2 Contents of the InterSystems Supply Chain Framework

After you install the InterSystems Supply Chain Framework into a namespace, you will see the following new packages in that namespace:

- The SC package.
  - The SC.Data subpackage contains the data model classes that you can modify.
  - The SC.Core.BP subpackage contains classes you refer to when creating business processes to manage issues.
  - The SC.Core.Tasks subpackage contains the tasks to be scheduled in the Task Manager.

  Other code in this package is purely internal.
- The datamodelAPI, scbi, and utils packages, which are purely for internal use.

After installation, you will also see new tasks in the Task Manager schedule, specifically the tasks in the SC.Core.Tasks subpackage.

Also see Guidelines for Safe Customizations.

2.3 Upgrade

As mentioned above, there are three deployable components involved: InterSystems IRIS, InterSystems Supply Chain Framework, and application code. Each of these components can be upgraded, together or independently. If you need to upgrade more than one of the deployable components at the same time, do so in the following order:

1. Upgrade core InterSystems IRIS data platform.

   See the InterSystems IRIS upgrade instructions, and be sure to recompile the %SYS namespace as instructed.
2. Upgrade InterSystems Supply Chain Framework.

   To upgrade the InterSystems Supply Chain Framework, use the same command as for the initial deployment:

   ```
   zpm:SC> install isc-supply-chain
   ```

   Optionally, to upgrade to a specific version of the framework (in this example version 1.1.0):

   ```
   zpm:SC> install isc-supply-chain 1.1.0
   ```
3. Upgrade application code. Use client DevOps tool to do this.

2.4 Reconfiguring the Documentation Links

Because the base product is InterSystems IRIS, the documentation links are specific to that product, and you should update your web server to redirect to the Supply Chain Orchestrator documentation.

If you are using the private web server, update the *install-dir*\httpd\conf\httpd-doc.conf file, which might initially look like this:

```
Redirect /csp/docbook/ http://docs.intersystems.com/iris20231/csp/docbook/
```

Change this file to the following instead (to the content for this specific release):

```
Redirect /csp/docbook/ http://docs.intersystems.com/supplychain20231/csp/docbook/
```

Or to the following (to the latest content, which will change over time):

```
Redirect /csp/docbook/ http://docs.intersystems.com/supplychainlatest/csp/docbook/
```

If you are using a different web server, adapt the above instructions as applicable for that server.

2.5 See Also

- Supply Chain Solution Overview
- Guidelines for Safe Customizations
- Building Smart Real-Time Supply Chain Applications

3

Supply Chain Solution Overview

This page summarizes what is needed for a solution built with InterSystems Supply Chain Orchestrator™, which is designed to let you easily build the following:

- KPIs that give visibility across the entire supply chain.
- Issue management: generating issues, analyzing them, with user input if needed, and resolving them, perhaps with calls to external APIs.

With the underlying foundation of InterSystems IRIS®, it is possible for the solution to do much more such as loading data, performing data transformations, providing its own APIs, supporting additional analytics options, and more.

3.1 Prerequisite

The prerequisite is to set up the core system for Supply Chain Orchestrator:

1. Deploy an InterSystems IRIS instance.
2. Create an InterSystems namespace reserved for use by Supply Chain Orchestrator — the *supply chain namespace*. This documentation uses the namespace name SC, but you can use a different name.
3. Install the InterSystems Supply Chain Framework onto that instance.

See Installation and Upgrade.

3.2 Basic Implementation Requirements

After setting up the core system, you will need to perform the following steps, not necessarily in a particular order:

- If necessary, extend the supply chain tables.
- Load your data into the supply chain tables. To do this, you can use the APIs provided by the product; you can also do this within the production that you create in the supply chain namespace, as described below.
- Using the Task Manager, add the following tasks to the schedule, so that they run at suitable times for your business needs:
  - SC.Core.Tasks.AnalyzeAllNewIssues

  - SC.Core.Tasks.BuildCubes
  - SC.Core.Tasks.ConsolidatedInventoryTask
  - SC.Core.Tasks.PredictInventory
  - SC.Core.Tasks.SynchCube
  - SC.Core.Tasks.SynchIssueCube
  - SC.Core.Tasks.UpdateKPIssue

  These tasks should all run in the supply chain namespace.

- Create a production in the supply chain namespace, as described below. Together with the issue-related tasks listed above, the production is responsible for issue management.
- Use the Business Intelligence Analyzer to get acquainted with analytics cubes provided by the product. Their purpose is to enable you to analyze the supply chain data and to create KPIs.
- If necessary, copy the analytics cubes and modify the copies, or create your own analytics cubes from scratch. This would be necessary only if the predefined cubes do not define the data you need. Another option is to define subject areas, to provide access to subsets of the data. Subject areas can be used in all the same ways as cubes.
- Define one or more KPIs. This task primarily consists of examining the requirements, using the analytics cubes to obtain the needed values, creating a simple specification, and using an API to add the KPI to the system. A KPI can generate issues, but does not have to do so.

3.3 Production Requirements

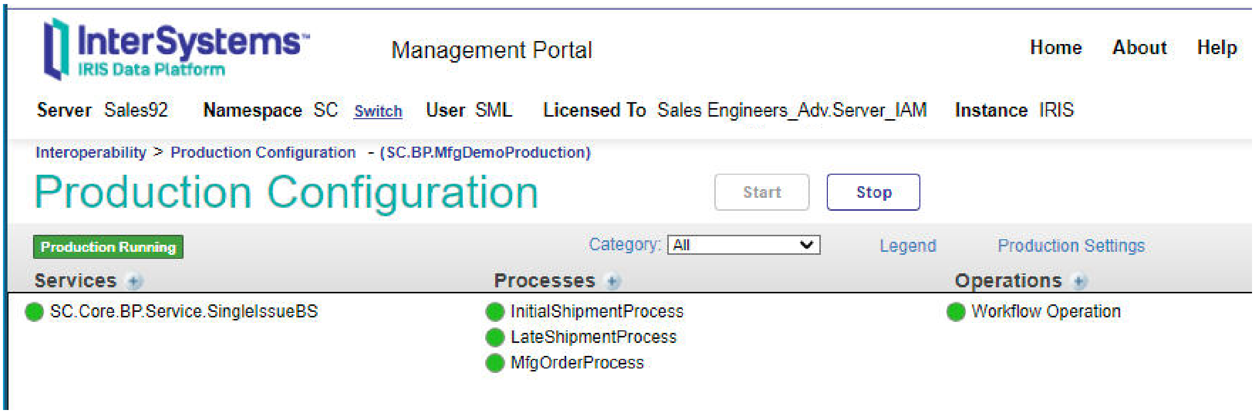
A core feature of Supply Chain Orchestrator is issue management, which generally includes your custom business logic, which can include automation, calls to third-party APIs, and the use of InterSystems workflow technology to involve users where appropriate. Your custom business logic is provided by a production that you create, configure, and manage within the Management Portal. The requirements of that production are as follows:

- The production must be in the supply chain namespace.
- It should include the business service SC.Core.BP.Service.SingleIssueBS, which is available out of the box.
- It should include one business process for each type of issue analysis you need to perform. Typically each type of issue requires its own analysis. For example, a late shipment may be a type of issue; you would generally analyze any late shipment the same way, and that logic would not apply to other types of issues.

  For issues generated automatically from KPIs, there is typically one business process for each KPI definition.

  To create these business processes, you use a graphical editor within the Management Portal, which enables you to create complex logic without coding. See the tutorial for details.
- If you need to use workflow, the production must include the business operation EnsLib.Workflow.Operation, which is available out of the box. Make a note of the configuration name of this business host, because you need to use that name within the business processes. Where applicable, the business processes can use the <call> element to call this business operation.

The following shows an example:



Server Sales92 **Namespace** SC **User** SML **Licensed To** Sales Engineers_Adv.Server_IAM **Instance** IRIS

Production Configuration - (SC.BP.MfgDemoProduction)

Category: All

Services: SC.Core.BP.Service.SingleIssueBS

Processes: InitialShipmentProcess, LateShipmentProcess, MfgOrderProcess

Operations: Workflow Operation

You can also use the production to load some or all of the data into the supply chain tables.

3.4 Guidelines for Safe Customizations

When creating your application, remember the following guidelines so that your additions and customizations will survive any upgrade to InterSystems IRIS or to the add-on InterSystems Supply Chain Framework:

- You *can* edit the classes in the SC.Data package, which represents the supply chain data model. If you do, be sure to make no manual changes to the `Storage` section of those classes. Let InterSystems IRIS do the updates for you as you modify the classes; the class compiler expands the storage definition safely to keep your data accessible.
- *Do not change any other classes* provided by the framework; for example, do not change anything under packages SC.Core.*

  Apart from the classes in SC.Data, all classes may be replaced upon an upgrade.
- *Do not add classes* to any of the packages provided by the framework.
- You *can* create your own classes in other packages, following the standard InterSystems IRIS naming conventions applicable to a namespace that is interoperability-enabled.

3.5 See Also

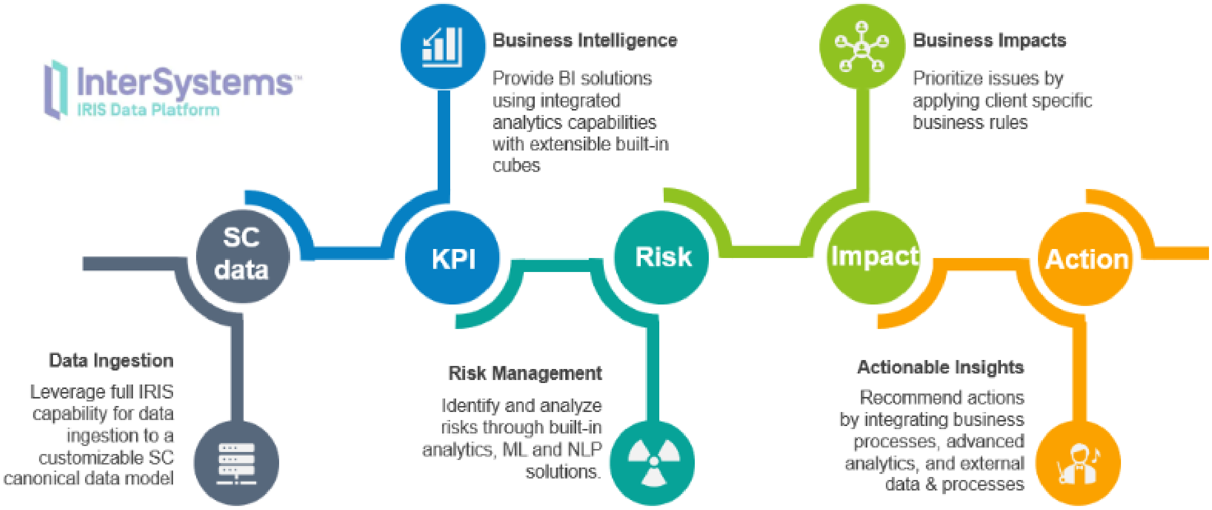
- Installation and Upgrade
- Building Smart Real-Time Supply Chain Applications
- Using the Task Manager (generic)
- Formal Overview of Productions (generic)
- Overview of Workflow within Productions (generic)

4

Issue Management

In InterSystems Supply Chain Orchestrator™, an *issue* is an InterSystems IRIS® object that is used to capture any risk, concerns, or things which may impact the supply chain. Most issues may also require some kind of action to minimize the impact. Some examples are late shipments, inventory stock out, labor shortage, and so on. An issue can be linked to an specific instance of a supply chain object, such as an order, a shipment, inventory and so on, but the relationship is not always required. Issues can be tracked by category, associated data object, time frame, status, and so on.

The following diagram shows how supply chain data is processed in Supply Chain Orchestrator, and related technologies used. Risks in the diagram are modeled using the issue object, and related impacts as well as actions are also linked to the issue object (that is, the issue API also returns associated impact and action data if that exists).



Business Intelligence

Provide BI solutions using integrated analytics capabilities with extensible built-in cubes

Business Impacts

Prioritize issues by applying client specific business rules

Data Ingestion

Leverage full IRIS capability for data ingestion to a customizable SC canonical data model

Risk Management

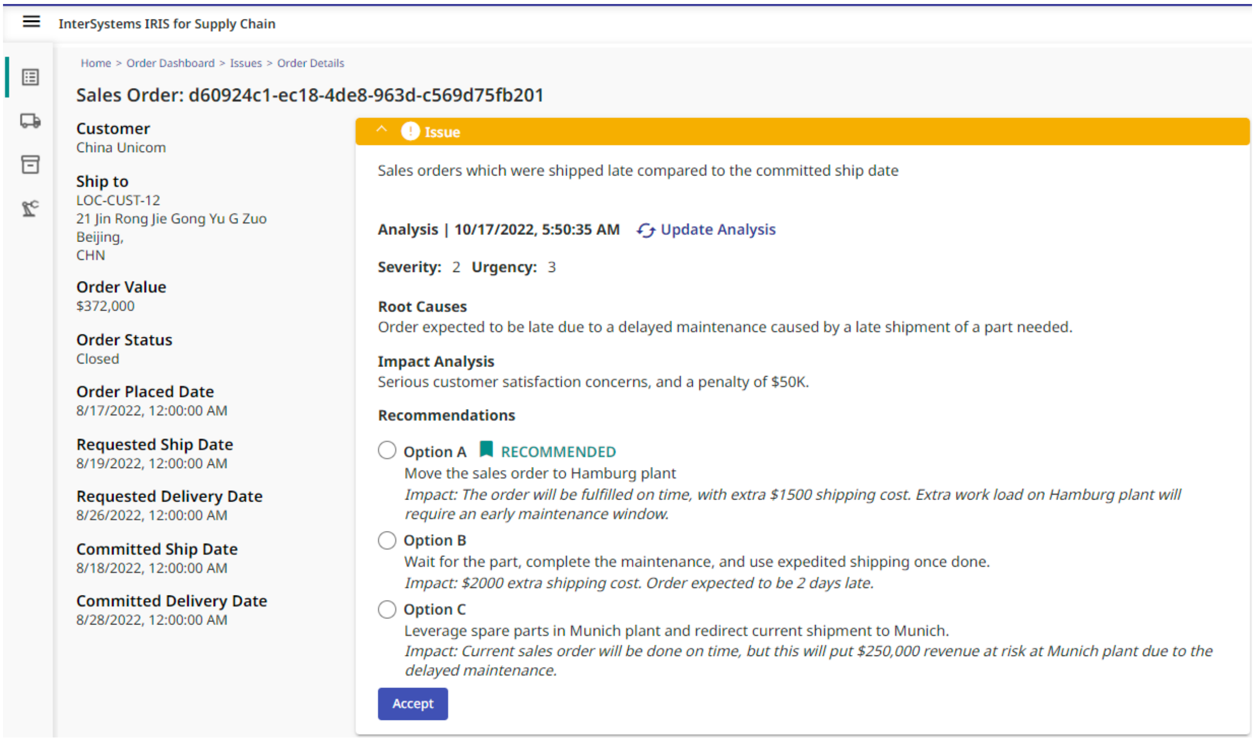
Identify and analyze risks through built-in analytics, ML and NLP solutions.

Actionable Insights

Recommend actions by integrating business processes, advanced analytics, and external data & processes

4.1 Example Issue Analysis

Issue management is at the core of Supply Chain Orchestrator. Issue management in turn includes issue analysis and resolution. The following is an example of an issue resolution as shown in a demo UI:


InterSystems IRIS for Supply Chain
Home > Order Dashboard > Issues > Order Details
Issue
Update Analysis
Accept

Sales Order: d60924c1-ec18-4de8-963d-c569d75fb201

Customer
China Unicom

Ship to
LOC-CUST-12
21 Jin Rong Jie Gong Yu G Zuo
Beijing,
CHN

Order Value
$372,000

Order Status
Closed

Order Placed Date
8/17/2022, 12:00:00 AM

Requested Ship Date
8/19/2022, 12:00:00 AM

Requested Delivery Date
8/26/2022, 12:00:00 AM

Committed Ship Date
8/18/2022, 12:00:00 AM

Committed Delivery Date
8/28/2022, 12:00:00 AM

Sales orders which were shipped late compared to the committed ship date

Analysis | 10/17/2022, 5:50:35 AM

Severity: 2 **Urgency:** 3

Root Causes
Order expected to be late due to a delayed maintenance caused by a late shipment of a part needed.

Impact Analysis
Serious customer satisfaction concerns, and a penalty of $50K.

Recommendations

○ Option A RECOMMENDED
Move the sales order to Hamburg plant
Impact: The order will be fulfilled on time, with extra $1500 shipping cost. Extra work load on Hamburg plant will require an early maintenance window.

○ Option B
Wait for the part, complete the maintenance, and use expedited shipping once done.
Impact: $2000 extra shipping cost. Order expected to be 2 days late.

○ Option C
Leverage spare parts in Munich plant and redirect current shipment to Munich.
Impact: Current sales order will be done on time, but this will put $250,000 revenue at risk at Munich plant due to the delayed maintenance.

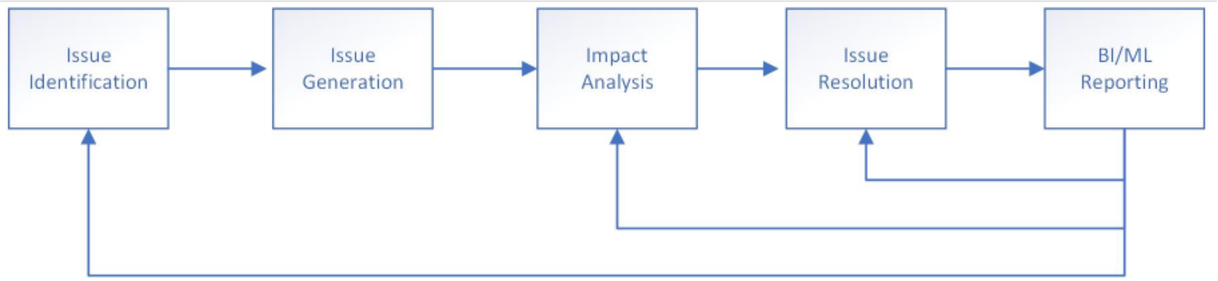
As shown in the example above, an issue analysis can cover the following, but not limited to, aspects:

- Root cause analysis
- Impact analysis
- Issue rating by severity and urgency
- Issue resolution/recommendation
- Actions related to the resolution, such as sending out a notification, or place an order in ERP system, and so on

You implement the analysis in an InterSystems IRIS business process by using a graphical UI that is designed for business logic.

4.2 Issue Life Cycle

The following diagram shows the life cycle of issues in Supply Chain Orchestrator.



There are different ways to identify and generate issues. One common approach is to auto-generate issues using KPIs, i.e., an issue is generated for every record that meets the KPI conditions. For example, a late-delivery KPI for sales orders can generate an issue for every late-delivery order.

KPIs are only one way to generate issues. Other ways to generate issues include

- Through BPL/DTL processes in Supply Chain Orchestrator. Any data integration or business process can be used to generate issues for specific conditions.
- By a Smart Data Service (SDS). For example, a SDS can be implemented to subscribe to external events, and determine the impacts on current supply chain. Such impacts can be captured in the form of issues and persisted in InterSystems IRIS.
- Sent in from external systems. Supply chain risks can be determined outside Supply Chain Orchestrator. For example, a manufacturing application may determine a part shortage for certain product at a plant, and the information can be sent to Supply Chain Orchestrator as an issue through API for issue resolution and reporting purpose.

Each issue can have a business process associated with it for impact analysis and issue resolution logic. Business impacts include setting up the severity and urgency levels, estimated the time and money impacts, as well as other implications for the issue.

The same business process can also be used to implement resolution logic. Issues can be automatically resolved through a process, or it may require business user to review the options and make a final decision (in which case InterSystems IRIS Interoperability workflow is used).

4.3 Generating Issues from KPIs

A KPI can generate issues. To enable issue generation for a KPI, specify the following in the KPI definition:

```
"issueKpi": true,
```

If this value is set to `true`, you can also add the following optional attributes to the KPI definition:

- `defaultIssueSeverity` specifies the default severity level for all issues generated from this KPI
- `analysisService` specifies the BPL name for running and issue analysis and resolution logic

There are automatically run system tasks to update issues for all issue-enabled KPIs:

- SC.Core.Tasks.UpdateKPIIssues generating issues as needed for KPIs. Specifically, this task does the following for each issue-enabled KPI:
  1. Identify the records that triggered the KPI condition (such as the late orders)
  2. Check to see if the database contains issues that correspond to these records in combination with this KPI. If the issues do not yet exist, the task creates them, using information contained in the KPI definition, including the name of the business process that analyzes this kind of issue by default. The new issues have `open` status.
  3. Check to see if the database contains additional records associated with this KPI (records not currently triggering the condition but that previously did so). If so, the task updates those issues to close them; specifically it sets the status to `closed`, and `resolutionType` to `noLongerValid`.
- SC.Core.Tasks.AnalyzeAllNewIssues starts issue analysis. This task looks at all non-closed issues that do not yet have an issue analysis record. For these issues, the task creates a new issue analysis. In the Visual Trace for the production, you will see SC.Core.BP.Service.SingleIssueSS receiving an issue analysis request message. SC.Core.BP.Service.SingleIssueSS will then send a message to the business process that is named by the issue.

> If the business process includes workflow, the Visual Trace will also show those components as applicable.

There are also API calls to run the KPI issue update logic.

4.4 Introduction to the Issue Management API

Supply Chain Orchestrator provides an data API, which includes options for working with issues:

Creating an issue

```
POST {{IRIS-SERVER}}/api/scbi/v1/issues
```

With the issue value in JSON format added to the body of the request.

Get issues

```
GET {{IRIS-SERVER}}/api/scbi/v1/issues
```

Additional query parameters based on any issue attributes can be added in the form of HTTP parameters, such as `status=open`.

Update issues for a KPI

```
POST {{IRIS-SERVER}}/api/scbi/v1/kpiissues
```

Forces an issue update for the given KPI. This applies to KPIs that are issue-enabled. In this case, specify KPI information in the JSON body data, for example:

```
{
    "kpi": "SalesOrderLateShipVsCommitted"
}
```

Force a new analysis for an issue

```
POST {{IRIS-SERVER}}/api/scbi/v1/analyzeissue/[ISSUE-ID]
```

This option applies if a process is associated with an issue. In this case, specify the process to use by including the following JSON body data:

```
{
    "processName": "YourProcessName"
}
```

4.5 See Also

- Supply Chain Solution Overview
- Building Smart Real-Time Supply Chain Applications
- Formal Overview of Productions (generic)
- Building BPL Business Processes (generic)
- BPL Tutorial for Supply Chain Orchestrator

- Adding Workflow Scenarios
- API Reference

5

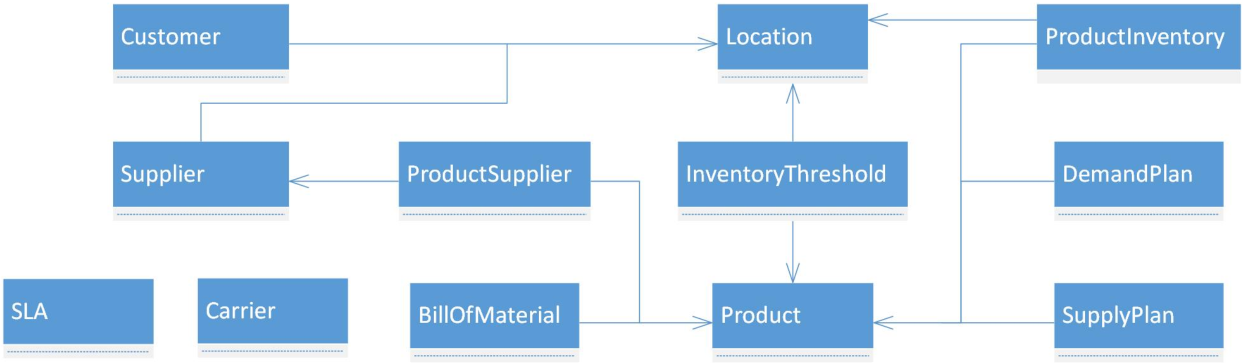
Supply Chain Data Model

This page introduces the data model provided by InterSystems Supply Chain Orchestrator™.

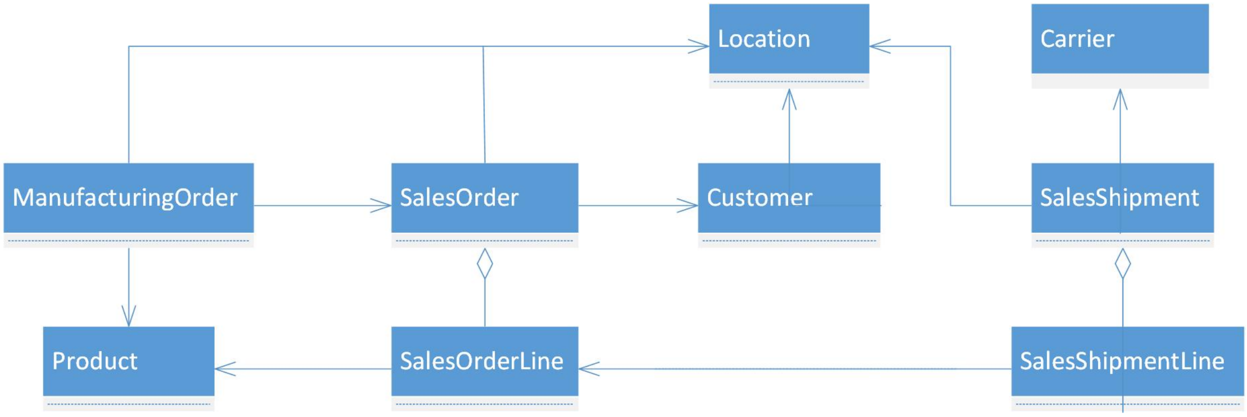
5.1 Supply Chain Data Objects

The following diagrams show the data objects supported in the current model, and their relationships.

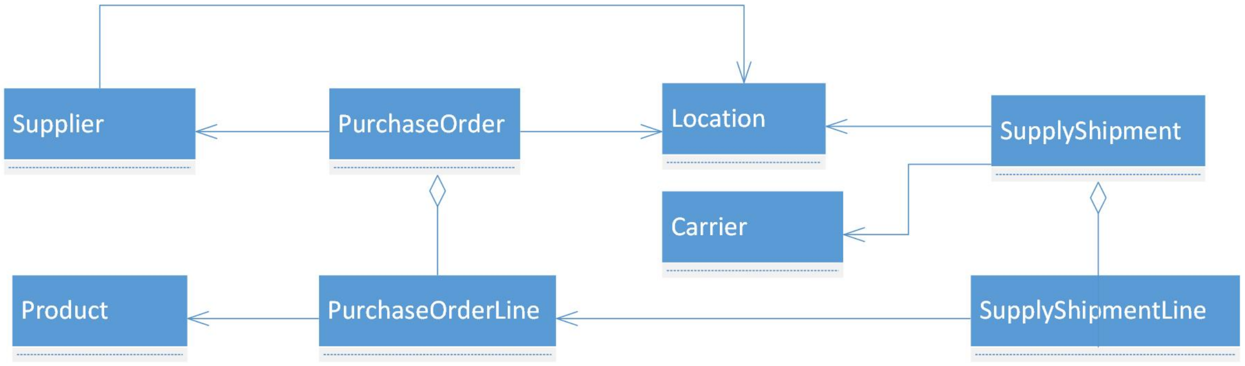
5.1.1 Master and Reference Supply Chain Data



5.1.2 Outbound and Manufacturing Data



5.1.3 Inbound Supply Chain Data



5.2 Introduction to Data Model Classes

All supply chain objects supported by the model are in the package SC.Data. These classes can be customized to suit the business needs of any clients. These data model classes leverage core implementations provided in the package SC.Core.Data, but no client should change anything in the core package. To help the consumption of the data model (such as data integration, UI developments, and so on), APIs are provided as live documentation of the data model.

5.3 Customizing the Data Model

You can extend the data model in different ways, depending on your technical skills.

5.3.1 Customizing via Coding

All supply chain data objects have an InterSystems class definition in the package SC.Data. These are classes used for all supply chain applications and services. These classes can be modified to add new attributes to the existing data model. As shown in the example below, these classes are mostly empty out of the box because the standard model attributes are all defined in the parent class. You can add new properties or indexes to these class for customization.

```
///This is a custom implementation of the Customer class
Class SC.Data.Customer Extends SC.Core.Data.Customer
{

Property MyNewProp as %String;

}
```

5.3.2 Customizing via the API

For people without class programming knowledge, or who just need to make a change without direct access to the IRIS server, there is an API call you can use to add new custom attributes to any supply chain object. As a result of such API calls, the corresponding object definition classes are updated (as if manually coded), database tables are adjusted automatically, and the attributes are available to be used anywhere on the InterSystems IRIS® platform, such as in analytics cubes, or in a business process.

5.4 See Also

- Supply Chain Solution Overview
- Supply Chain Data Model API
- Supply Chain Data API
- API Reference

6

Supply Chain Data Model API

This page introduces the data model API provided by InterSystems Supply Chain Orchestrator™.

Supply Chain Orchestrator includes a data model API that you can use for model discovery and customization, including ways to

- List all supply chain data objects
- Retrieve details of any objects, with attributes, data type, size limit, and so on
- Extend the data model by adding custom attributes to existing supply chain objects

6.1 List of Supply Chain Data Objects

Use this API call to find all supply chain objects in the data model:

```
GET {{IRIS-SERVER}}/api/scdata/v1/objects
```

The response looks like the following example:

```
[
    {
        "objectName": "BOM",
        "className": "SC.Data.BOM",
        "description": "Object for bill of material. This object is commonly
                        used in manufacturing to specify the parts required
                        to make or assemble a product."
    },
    {
        "objectName": "Carrier",
        "className": "SC.Data.Carrier",
        "description": "This object is used to capture shipment
                        carrier information."
    },
...
]
```

To find the details of each object, use the next API call with the `objectName` value returned in this API response.

6.2 Get Object Definition

To get the details of an object, first get the object name using the above API call. Then use this API call:

```
GET {{IRIS-SERVER}}/api/scdata/v1/objects/[ObjectName]
```

For example, the following is part of the response returned for `Customer` object:

```
{
    "objectName": "Customer",
    "className": "SC.Data.Customer",
    "objectName": "This object is used to capture the master
                  data for a customer.",
    "attributes": [
        {
            "name" : "uid",
            "description" : "unique ID of a customer",
            "dataType" : "String",
            "required" : 1,
            "maxLength" : "256",
            "isCustom" : 0
        },
        {
            "name" : "name",
            "description" : "official name of the customer",
            "dataType" : "String",
            "required" : 0,
            "maxLength" : "256",
            "isCustom" : 0
        },
...
    ]
}
```

6.3 Adding Custom Attributes

To add a new attribute to an object in the supply chain model, use this API call:

```
POST {{IRIS-SERVER}}/api/scdata/v1/attributes/[ObjectName]
```

with the attribute definition JSON in the API body, such as:

```
{
    "name" : "customProperty",
    "description" : "Custom attribute added for testing",
    "dataType" : "String",
    "maxlength" : 120,
    "required" : 1
}
```

Once an attribute is added through this API, data access APIs can be used to load data into the attributes, or search for records based on the newly created attributes. No additional step or waiting is needed.

Important: This API does not support more advanced customizations as can be made by modifying the class directly, such as adding indexes or validation rules. Also, there is no API to update or delete custom attributes, so use this API with caution.

6.4 See Also

- Supply Chain Solution Overview
- Supply Chain Data Model
- Supply Chain Data API
- API Reference

7

Introduction to the Data API

InterSystems Supply Chain Orchestrator™ provides an API for creating, updating, deleting, and retrieving data. This page provides an introduction to this API.

7.1 API URL Patterns

All the API calls follow the same URL pattern:

```
GET {{IRIS-SERVER}}/{{DATAMODEL-PATH}}/OBJECT_PATH?parameters
POST {{IRIS-SERVER}}/{{DATAMODEL-PATH}}/OBJECT_PATH
GET/PUT/DELETE {{IRIS-SERVER}}/{{DATAMODEL-PATH}}/OBJECT_PATH/uid
```

Where:

- The {{IRIS-SERVER}} part is the server information of the InterSystems IRIS® instance. In a locally deployed server, it may look like `http://localhost:52773`
- The {{DATAMODEL-PATH}} part is the API base URL, such as `/api/scdata/v1`
- The OBJECT_PATH part is simply the object name in lower case plural form, such as `salesorders` or `customers`. See the table below for all object values.
- The first URL pattern gets data for an object, or searches for objects using a set of parameters. Any attributes of the object can be used in a search criterion.
- The second URL pattern creates a new object record. The body of the request should include the JSON string for the new object.
- The third URL pattern retrieves, updates, or deletes an object record by its `uid` (which is the external primary key).

Some examples:

- Finding all sales orders with status `Open` or `PartialShip`:

  ```
  GET {{IRIS-SERVER}}/{{DATAMODEL-PATH}}/salesorders?orderStatus=Open,PartialShip
  ```

- Creating a new customer:

  ```
  POST {{IRIS-SERVER}}/{{DATAMODEL-PATH}}/customers
  ```

with the following JSON in the request body:

```
{
    "uid" : "CUST-TEST-101",
    "name" : "Google",
    "type" : "HighTech",
    "contact" : "Ming",
    "url" : "https://google.com"
}
```

- Retrieving a supply shipment record with `uid` value `SUP-SHIP-1001`:

  ```
  GET {{IRIS-SERVER}}/{{DATAMODEL-PATH}}/supplyshipments/SUP-SHIP-1001
  ```

- Updating a location record:

  ```
  PUT {{IRIS-SERVER}}/{{DATAMODEL-PATH}}/locations/LOC-PLANT-002
  ```

  with new location JSON data in the request body.

7.2 Search APIs

Searches are supported through GET APIs with one or multiple URL parameters; each parameter maps to one search condition. If more than one parameter are used, the parameter conditions are combined with logical `AND` operation. For example, the following searches all Lenovo laptop products:

```
GET /products?brand=Lenovo&category=laptop
```

Each parameter name should match exactly an attribute name of the primary object. In some cases, attributes from secondary objects (contained or referenced objects) can also be used. For example, some product attributes can be used in an order search.

The search parameter value can be a single value, a list of values, or a range of values, as follows:

- *Single value*. This is the simplest search criterion, using the format `parameterName=value`. If the parameter is a string or a list of strings, the value matching is case-insensitive. So `brand=lenovo` is the same as `brand=Lenovo`.
- *List of values*. For a condition that requires matching any of the values in a list, you can provide a list of values as a comma-separated string. For example, to find products of either Lenovo brand or Dell brand, you can use `brand=Lenovo,Dell`. The list can be used for all data types, string, date, or numbers.
- *Range of values*. For date and numerical values (including currency), a range can be used in a search parameter. A range is specified by two values, separated by `..` (two dots), in format of `min..max`. Both values are inclusive for the range. For example, `price=100..200` finds all records with price in range of 100 (including 100) and 200 (including 200). The two range boundary values are not always required, and if one is missing, it means that boundary does not exist. For example, `price=..200` means any record with price less than or equal to 200, while `price=100..` means any record with price greater than or equal to 100.
- *Null value*. For any data type, you can search for records with an attribute not set (that is, a null value), or an attribute with some value set (that is, not null). For such cases, you can use two special string values `NULL` and `NOTNULL` for any attribute, such as `attr1=NULL&attr2=NOTNULL`

For example, this call finds all laptop computers from either Lenovo or Dell with price range between 500 and 1000:

```
GET /products?category=laptop&brand=Lenovo,Dell&price=500..1000
```

7.3 Sorting of Results

When more than one record is returned by an API call, you can define the order of the response data by specifying the sorting parameter `sortBy`. The value of this parameter must be a comma-separated list of attribute names of the primary object returned.

For example, the following API call defines how a list of orders should be sorted in the response:

```
GET /salesorders?sortBy=customer,orderValue
```

which defines the sorting to be first by customer, and then by order value.

By default, sorting is done in ascending order. To change the order to descending, use the `-` sign before the attribute name, such as the following example, which first sorts by customer in ascending order, and then by `orderValue` in descending order:

```
GET  /salesorders?sortBy=customer,-orderValue
```

7.4 Default Sorting

If no sorting parameter is provided in the request URL, a default sorting is applied, based on the primary object returned. The following table lists the default sorting parameters for each object type:

Object	URL path	Default sorting attributes
Carrier	carriers	name
Customer	customers	name
Supplier	suppliers	name
Product	products	name
Location	locations	locationName
BillOfMaterial	billofmaterials	productId, parentItemId
InventoryThreshold	inventorythresholds	siteLocationId, productId
Milestone	milestones	ID
ProductInventory	productinventories	siteLocationId, productId
ProductSupplier	productsuppliers	productId
SupplyPlan	supplyplans	locationId, productId
DemandPlan	demandplans	locationId, productId
SalesOrder	salesorders	-orderPlacedDate
SalesOrderLine	salesorderlines	orderId, lineNumber
SalesShipment	salesshipments	-actualShipDate
SalesShipmentLine	salesshipmentlines	shipmentId, lineNumber
PurchaseOrder	purchaseorders	-orderPlacedDate

Object	URL path	Default sorting attributes
PurchaseOrderLine	purchaseorderlines	orderId, lineNumber
SupplyShipment	supplyshipments	-actualShipDate
SupplyShipmentLine	supplyshipmentlines	shipmentId, lineNumber
ManufacturingOrder	manufacturingorders	-orderEntryDate

7.5 Pagination of Results

By default, any API call that returns multiple records returns the first 100 records that match the given criteria. You use pagination parameters to obtain additional records or a larger set of records. The pagination parameters give you the control needed to, for example, populate a table in a web UI, without excess resource consumption (memory, network bandwidth, and so on) or performance issues.

The pagination parameters are as follows:

- `pageSize`. This parameter defines the maximum number of record returned in the API call. The actual number of records returned can be less or equal to this value. The default value is 100, and the maximum allowed value is 1000.
- `pageIndex`. This parameter specified the page index (starting from 0), to be returned in the response. The default is 0.

For example, the following API call skips the first 200 orders and returns the next 100 orders, sorted by order value:

```
GET  /salesorders?sortBy=orderValue&pageSize=100&pageIndex=2
```

When a response returns only partial records using pagination, the following HTTP header parameters are populated in the response. Use these values to control the pagination logic in the UI.

- `pageSize`. The same value if specified in the request path parameter. Returns the default value if not explicitly specified in the request.
- `pageIndex`. The same value if specified in the request path parameter. Returns the default value if not explicitly specified in the request.
- `returnCount`. Number of records returned in the current response. This value is less than or equal to `pageSize`

7.6 Date and Date/Time Formats

For any date or date/time attribute, you must specify the value in ISO 8601 format, whether you provide this in the message JSON body or in the HTTP parameter.

Here are some examples. For date attributes:

```
2021-02-28
```

For date/time attributes (the following two values are equivalent):

```
2021-12-15T13:23:15-05:00
2021-12-15T18:23:15Z
```

If the time zone information is missing, the UTC time is assumed. For example, `2021-05-24T08:30:00` is treated the same as `2021-05-24T08:30:00Z`

7.7 See Also

- Supply Chain Solution Overview
- Supply Chain Data Model
- API Reference

8

Analytics Cubes for InterSystems Supply Chain Orchestrator

This page describes the Business Intelligence cubes provided by InterSystems Supply Chain Orchestrator™. You can use these cubes in multiple ways, including:

- To define KPIs
- For simple exploration via the Business Intelligence tools provided within InterSystems IRIS®, such as the Business Intelligence Analyzer
- To create pivot tables accessible via the Business Intelligence REST API

These cubes are available in the supply chain namespace. The cubes are available once you have loaded data into the supply chain tables, and the automated system tasks have built the analytics cubes.

Tip: If the Analyzer is not available, make sure that the namespace is analytics-enabled.

8.1 Available Data

This section summarizes the data available in the analytics cubes.

8.1.1 Order Insights

Supply Chain Orchestrator provides data and tools needed for insights to all sales orders, including:

- Order counts, total order revenue, mean/average revenue per order
- Order fulfillment performance (on-time, in full, and so on)
- By customers (region/country/company)
- By product (category/family/brand/product)
- By time (year/month/day)
- By status

Similarly, for supply/purchase orders:

- Order counts, total order revenue, mean/average revenue per order
- Supplier performance
- By supplier (region/country/company)
- By product (category/family/brand/product)
- By time (year/month/day)
- By status

8.1.2 Shipment Insights

Supply Chain Orchestrator also provides the following information for shipments, both inbound and outbound:

- Shipment count, shipment value
- ETA and related status for in transit shipments
- Performance for delivered shipments
- By customer/supplier
- By carrier, transportation mode, route, and so on
- By origin, destination
- By time (year/month/day)

8.1.3 Inventory Insights

Supply Chain Orchestrator also provides the following insights for inventories:

- Stock levels: stock out, approaching stock out, excess inventory
- Plan versus actual
- By product (category/brand/etc.)
- By location

8.2 Available Cubes

Supply Chain Orchestrator provides the following cubes:

- *SalesOrder cube*. This cube is the primary source for sales order related information.
  - Measures: order count, total revenue, average order value, mean order value
  - Dimensions: customer (country -> name), sales region, order status, order placed date (year->month->day), ship status, delivery status.

- *SalesOrderLine cube*. This cube supplements the SalesOrder cube, mainly to address any aspects related to product, such as revenue by product
  - Measures: total count, total value, total units
  - Dimensions: product category (category -> product), product brand (brand->product), ship status, delivery status

- *SalesShipment cube*. This cube provides insights to sales order shipments.
- *PurchaseOrder cube*. The source for purchase or supply order information.
- *PurchaseOrderLine cube*. This cube supplements the PurchaseOrder cube, mainly to address any aspects related to product, such as cost or quantity by product.
- *SupplyShipment cube*. Cube for supply shipment information.
- *Inventory cube*. Cube for inventory related information.

8.3 Getting Started with the Cubes

Once you have loaded data into the supply chain tables, and the automated system tasks have built the analytics cubes, you can use the Analyzer to see the data. Consult Getting Familiar with a Subject Area to help you get acquainted with the cubes.

8.4 See Also

- Introduction to Business Intelligence (generic)
- Introduction to the Analyzer (generic)
- Getting Familiar with a Subject Area (generic)

9

BPL Tutorial for Supply Chain Orchestrator

In InterSystems Supply Chain Orchestrator™, issue analysis and resolution logic is provided by business processes within an interoperability production that you create and configure. The production needs to include business processes, each of which analyzes a specific kind of issue. You can create each business process in the Management Portal, using a graphical editor that creates a business process that contains Business Process Logic.

This tutorial covers the steps in the business process design/configuration that are required to connect the business process to the rest of the issue resolution logic provided by the product, apart from creating workflow scenarios, discussed separately.

Also see the online learning path Building Smart Real-Time Supply Chain Applications, which includes information on creating these BPL business processes. For more basic information, see this online training and product documentation.

9.1 Typical Structure

A typical business process for issue analysis has the following overall sequence:

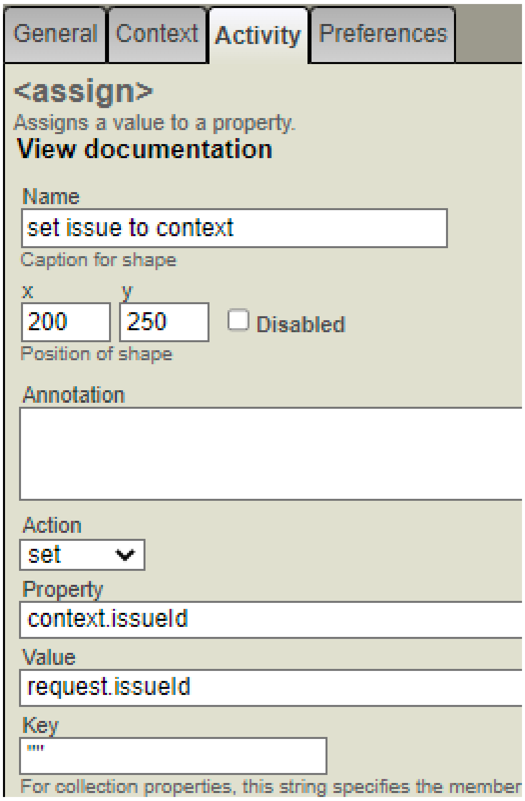
1. Initialize the context. This step keeps track of the issues and related business objects in the context object.
2. Perform the analysis. Within this analysis:
   - Specify analysis values, specifically the severity, urgency, root cause, and impact analysis of the issue.
   - If you are using workflow, use the <call> element to call the workflow operation at the appropriate points.
   - If you are using workflow, use the add-on tools to create recommendations for the workflow users. If the business process offers options for issue resolution, some extra steps are required to access the option details; see Adding Workflow Scenarios.

   Because the analysis is necessarily dependent on your use cases, the tutorial shows only parts of this.
3. Save the analysis results back to the issue, so the results can be accessed through APIs and UI.

9.2 Context Initialization

To initialize a business process for use in Supply Chain Orchestrator, do the following:

- On the **Context** tab, specify the following values:
  - **Request Class** — SC.Core.BP.Message.IssueAnalysisRequest
  - **Response Class** — Ens.Response
  - **Context Class** — SC.Core.BP.IssueContext (or a subclass of this class)
- As one of the initial steps in the BPL, use an <assign> element to set `context.issueId` as follows:



<assign>

Assigns a value to a property.

Name: set issue to context

Caption for shape

x: 200 y: 250 ☐ Disabled

Position of shape

Annotation:

Action: set

Property: context.issueId

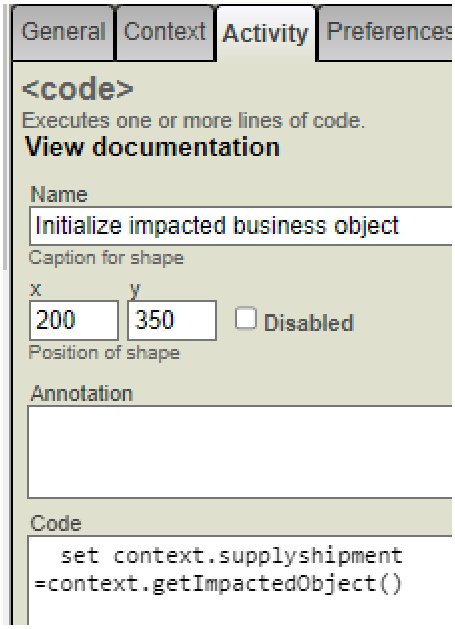
Value: request.issueId

Key: ""

For collection properties, this string specifies the member

- In many cases, an issue is tied to an instance of a business object or transaction, such as an order or a shipment. To make information about that object available to the BPL process, add a <code> element to retrieve the needed value and assign it to a property of the `context` object. Do this as one of the initial steps in the BPL.

  To retrieve the business object or transaction affected by the instance, use the syntax `context.getImpactedObject()` as shown here. The **getImpactedObject()** method is provided by chosen context class (SC.Core.BP.IssueContext). For example:



<code>

Executes one or more lines of code.

View documentation

Name: Initialize impacted business object

Caption for shape

x: 200
y: 350
☐ Disabled

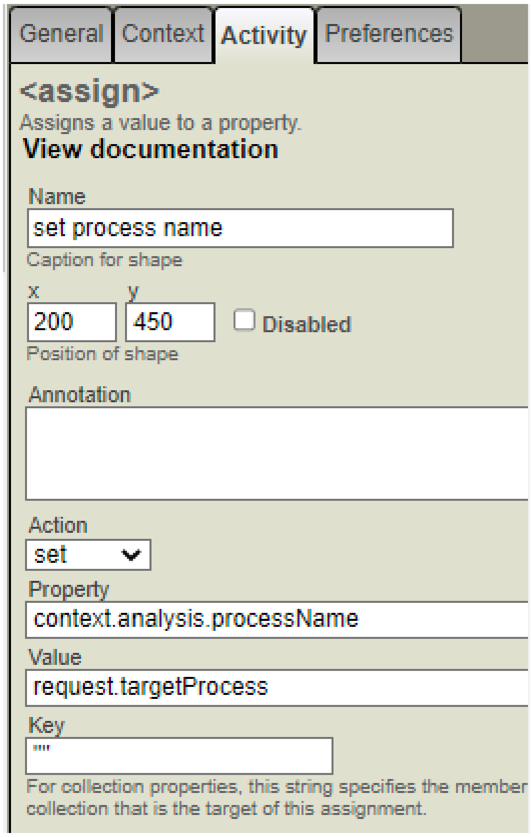
Position of shape

Annotation

Code

```
 set context.supplyshipment
=context.getImpactedObject()
```

- As one of the initial steps in the BPL, use an <assign> element to set `context.analysis.processName` equal to the `request.targetProcess`, as follows:



<assign>

Assigns a value to a property.

View documentation

Name: set process name

Caption for shape

x: 200
y: 450
☐ Disabled

Position of shape

Annotation

Action: set

Property: context.analysis.processName

Value: request.targetProcess

Key: ""

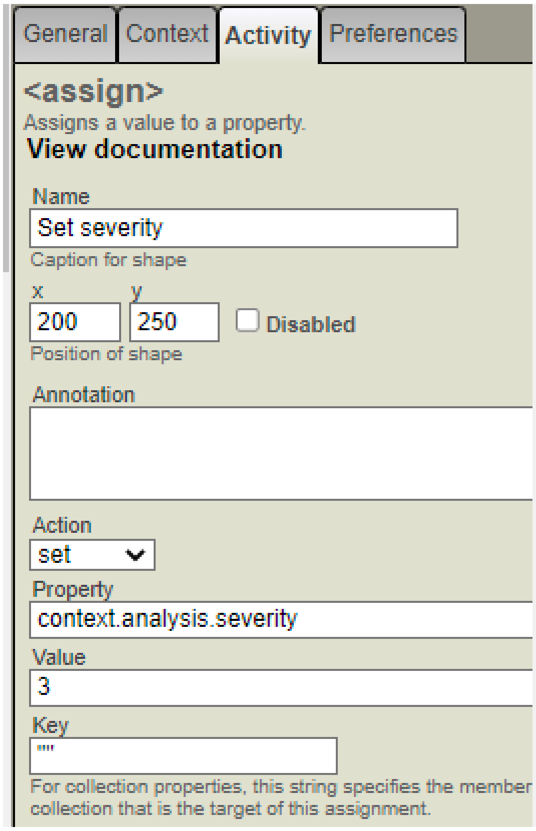
For collection properties, this string specifies the member collection that is the target of this assignment.

This step saves the business process name to the context, so it can be saved back to the issue with the analysis results. This is needed because different business process can be used to analyze the same issue.

9.3 Specifying Analysis Values

Once you have set up the context, add additional steps that contain the actual business logic to determine the severity, urgency, root cause, and impacts, and so on. This tutorial does not show those details. As you determine those values, save them to the `context.analysis` object; see Analysis Properties for names of the properties.

To specify the analysis values, use multiple <assign> elements, each of which sets one property. For example:



<assign>

Assigns a value to a property.

View documentation

Name: Set severity

Caption for shape

x: 200

y: 250

☐ Disabled

Position of shape

Annotation

Action: set

Property: context.analysis.severity

Value: 3

Key: ""

For collection properties, this string specifies the member collection that is the target of this assignment.

9.4 Analysis Properties

The analysis object is defined by the class SC.Core.Data.Internal.IssueAnalysis, which has the following properties for you to set as described here:

- severity — Specifies the severity of the issue, on a numeric scale.
- urgency — Specifies the urgency of the issue, on a numeric scale.
- rootCauseAnalysis — Describes the root cause of the issue, in user-friendly terms.
- impactAnalysis — Describes the results of the analysis, in user-friendly terms.

For data types and length limits, view SC.Core.Data.Internal.IssueAnalysis in the local class reference or in your choice of IDE.

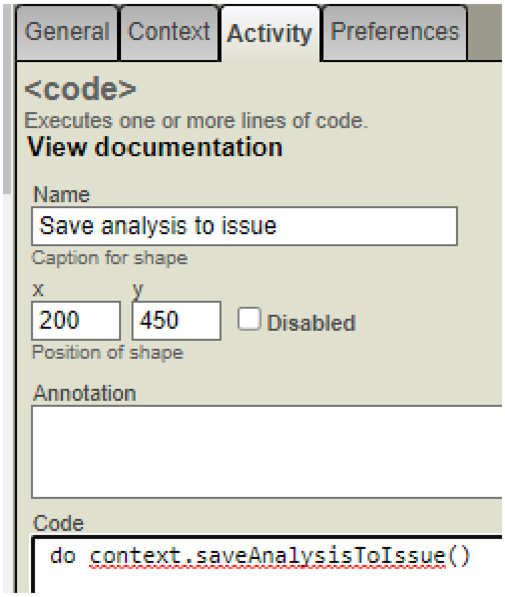
9.5 Saving the Analysis Results to the Issue

After the business logic is completed, the BPL must include a <code> element that saves the analysis details back to the issue:

- If the business process does not contain workflow, use the following syntax to save the analysis to the issue:

  ```
  do context.saveAnalysisToIssue()
  ```

  For example:

  Name: Save analysis to issue

  x: 200

  y: 450

  ☐ Disabled

  Annotation:

  Code:

  ```
  do context.saveAnalysisToIssue()
  ```

- If the business process does use workflow, the <code> element instead requires two lines of code:

  ```
  set context.analysis.workflowId = $listget(process.%MasterPendingResponses.GetAt(1),1)
  do context.saveWorkflowAnalysisToIssue()
  ```

  The extra step saves the workflow ID which can be used later if the workflow needs to be cleared up due to a new round of analysis being executed before the current workflow is closed.

9.6 Testing the Business Process

After configuring a business process for issue analysis, you can manually run the analysis on an issue through an API.

You can also use the analytics API, which includes a way to run an issue analysis using a specified business process and then retrieve the analysis results. To run an issue analysis, use the following API call:

```
POST    {{IRIS-SERVER}}/scbi/v1/runissueanalysis/ISSUE_ID
```

with the following JSON message in the body:

```
{ "processName": "Your business process name" }
```

If this call returns successfully, you can use the following API call to retrieve the analysis details:

```
GET: {{IRIS-SERVER}}/scbi/v1/issues/ISSUE_ID
```

The following is a sample response:

```
{
    "ID": "877",
    "recordCreatedTime": "2022-10-15T14:32:55.472Z",
    "lastUpdatedTime": "2022-10-17T09:50:35.180Z",
    "description": "Sales orders which were shipped late compared to the committed ship date",
    "triggerType": "KPI",
    "triggerObjectId": "SalesOrderLateShipVsCommitted",
    "impactedObjectType": "SalesOrder",
    "impactedObjectId": "d60924c1-ec18-4de8-963d-c569d75fb201",
    "severity": 2,
    "urgency": 3,
    "status": "workflow",
    "latestAnalysis": {
        "recordCreatedTime": "2022-10-17T09:50:35.174Z",
        "lastUpdatedTime": "2022-10-17T09:50:35.179Z",
        "issueId": 877,
        "runSequence": 3,
        "processName": "MfgOrderProcess",
        "severity": 2,
        "urgency": 3,
        "rootCauseAnalysis": "Order expected to be late due to a delayed maintenance
                caused by a late shipment of a part needed. ",
        "impactAnalysis": "Serious customer satisfaction concerns, and a penalty of $50K.",
        "workflowId": "54",
        "status": "workflow",
        "resolution": "none",
        "scenarios": [
            {
                "optionNumber": "A",
                "optionName": "Option A",
                "description": "Move the sales order to Hamburg plant",
                "costImpact": 1500,
                "timeImpact": 36,
                "supportingData": "Impact: The order will be fulfilled on time, with
                    extra $1500 shipping cost. Extra work load on Hamburg plant will
                    require an early maintenance window.",
                "feasibility": 1,
                "recommended": 1
            },
            {
                "optionNumber": "B",
                "optionName": "Option B",
                "description": "Wait for the part, complete the maintenance, and
                    use expedited shipping once done.",
                "costImpact": 500,
                "timeImpact": 0,
                "supportingData": "Impact: $2000 extra shipping cost. Order expected
                    to be 2 days late.",
                "feasibility": 1,
                "recommended": 0
            },
            {
                "optionNumber": "C",
                "optionName": "Option C",
                "description": "Leverage spare parts in Munich plant and redirect
                    current shipment to Munich.",
                "costImpact": 12000,
                "timeImpact": 0,
                "supportingData": "Impact: Current sales order will be done on time,
                    but this will put $250,000 revenue at risk at Munich plant due to
                    the delayed maintenance.",
                "feasibility": 1,
                "recommended": 0
            }
        ]
    }
}
```

9.7 See Also

- Supply Chain Solution Overview
- Building Smart Real-Time Supply Chain Applications
- Formal Overview of Productions (generic)
- Building BPL Business Processes (generic)
- Issue Management
- Adding Workflow Scenarios

10

Adding Workflow Scenarios

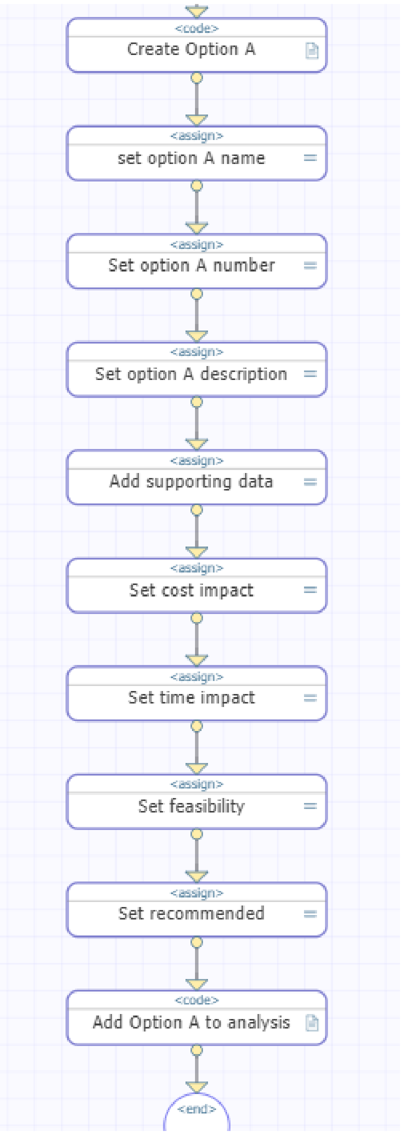
Within a BPL business process that examines a supply chain issue, you can include workflow, and you may need to present different scenarios to present to the users. In InterSystems Supply Chain Orchestrator™, this is easy to do.

This page describes how to create and populate a scenario object within a workflow.

Each scenario has a name, a description, and various properties such as the time and cost impact. Rather than having to define custom properties within the BPL, you can use the data object `SC.Core.Data.Internal.ActionScenario`, which already has the necessary properties.

10.1 Overview

As an overview, here is a screenshot of the relevant part of an example BPL diagram.



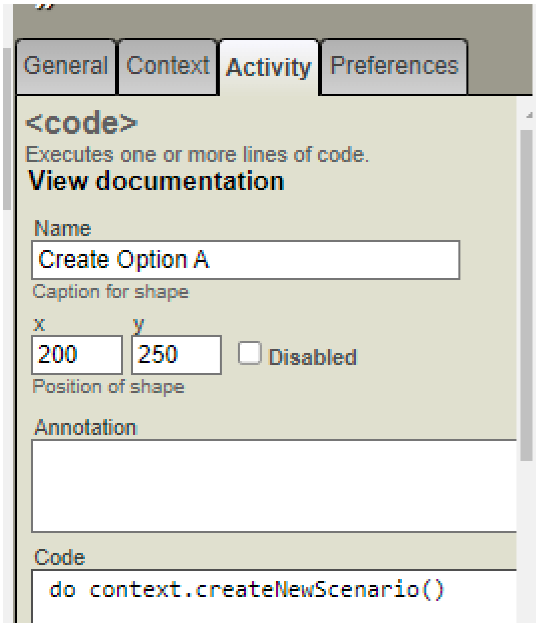
10.2 Steps to Create a Scenario

Within a BPL business process, use the following procedure to add each scenario to the business process:

1. Add a <code> element to create the scenario object. Specifically, for this element, specify **Code** as follows:

   ```
   do context.createNewScenario()
   ```

   For example:



<code>

Executes one or more lines of code.

Name: Create Option A

Caption for shape

x: 200 y: 250 ☐ Disabled

Position of shape

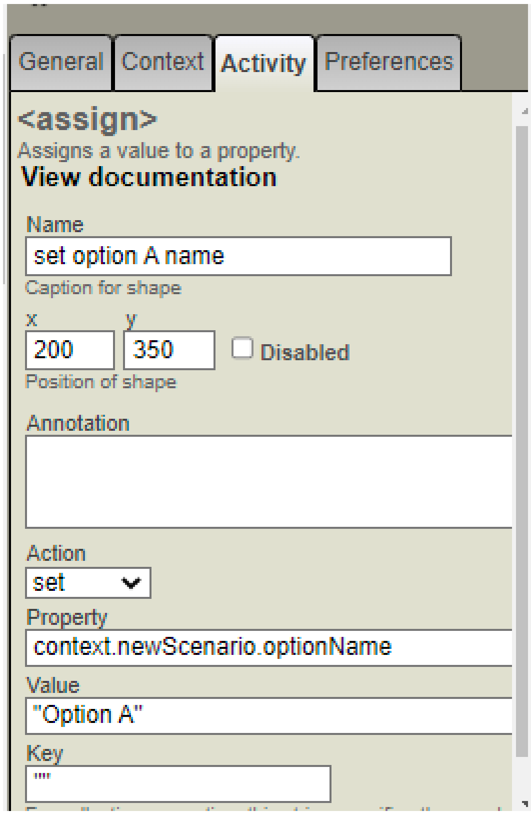
Annotation

Code

```
do context.createNewScenario()
```

This step adds a new property to the context object, so that now you can set properties of `context.newScenario`.

2. Include a set of <assign> elements, each of which specifies the value of a scenario property. For example, you might assign the `optionName` property like this:



<assign>

Assigns a value to a property.

Name: set option A name

Caption for shape

x: 200 y: 350 ☐ Disabled

Position of shape

Annotation

Action: set

Property: context.newScenario.optionName

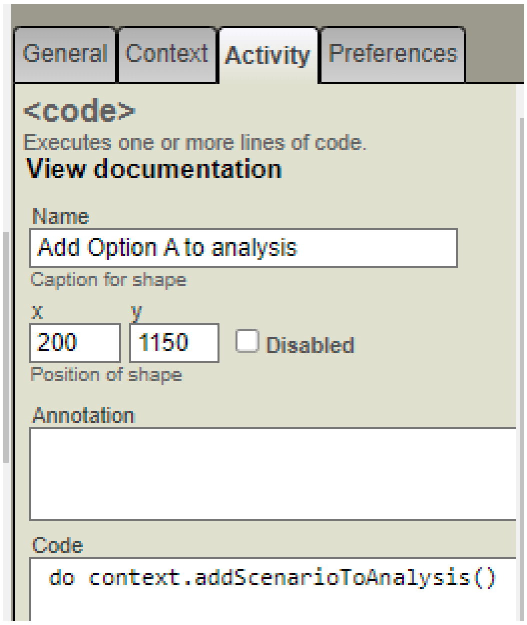
Value: "Option A"

Key: ""

3. Once you have specified all the properties, add a <code> element to create the scenario object. Specifically, for this element, specify **Code** as follows:

```
do context.addScenarioToAnalysis()
```

For example:



<code>

Executes one or more lines of code.

View documentation

Name: Add Option A to analysis

Caption for shape

x: 200

y: 1150

☐ Disabled

Position of shape

Annotation

Code

```
do context.addScenarioToAnalysis()
```

After the above steps are completed, the same set of steps can be repeated for another scenario.

Please keep in mind that these steps supplement the usual InterSystems IRIS® guidelines to implement the rest of the workflow.

10.3 Scenario Properties

The scenario object is defined by the class SC.Core.Data.Internal.ActionScenario, which has the following properties for you to set as described here:

- optionNumber — A very short identifier for the option such as a single letter or a single number, for use in presentation to the users. For an example, see Example Issue Analysis.
- optionName — A user-friendly short name of the option.
- description — A user-friendly description of the option.
- supportingData — Supporting data used in the analysis, such as inventory position, labor availability, etc.
- feasibility — Describes how feasible the scenario is, on a numeric scale.
- recommended — Indicates if this is the recommended scenario.
- costImpact — The financial impact of the option/scenario (numeric).
- timeImpact — Time impact of the scenario.

For data types and length limits, view SC.Core.Data.Internal.ActionScenario in the local class reference or in your choice of IDE.

10.4 See Also

- Supply Chain Solution Overview
- Formal Overview of Productions (generic)
- Building BPL Business Processes (generic)
- BPL Tutorial for Supply Chain Orchestrator
- Issue Management

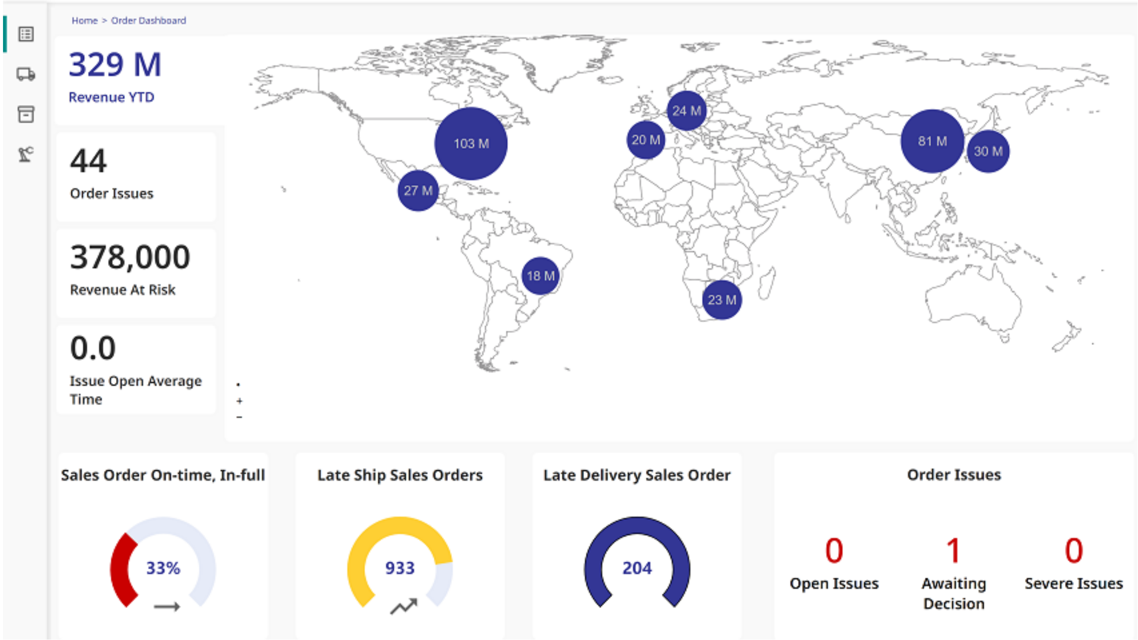
11

Configuring KPIs for InterSystems Supply Chain Orchestrator

This page describes how to configure definitions for *key performance indicators*, or *KPIs*.

11.1 Purposes of KPIs

KPIs are quantifiable measures that gauge a company's performance against a set of targets, objectives, or industry peers. KPIs are widely used in supply chain to help stakeholders understand how the business is doing, and what kind of risks exists across the supply chain. The following screenshot shows some supply chain KPIs: every number on the screen is a KPI.



In Supply Chain Orchestrator™, KPIs serve a second, important role: they can automatically generate issues. This is suitable when the KPI identifies a condition that needs correction.

11.2 Configuration Process

To define a KPI, use the following general process:

1. Define KPI logic conceptually, in business terms, with specific conditions. Usually, this is the information you collect from the business users on how they measure the business using this KPI.
2. Map KPI logic to the Supply Chain Orchestrator data model and analytics cube as shown in the KPI configuration example.
    - Does the data model cover the required business entity, or does the corresponding data model object have all the required attributes?

      If not, extend the data model as needed; see the data model documentation for more details.
    - Check to see if the associated cube meets the KPI requirements. For this process, you will use the Business Intelligence Analyzer. (If the Analyzer is not available, make sure that the namespace is analytics-enabled.)

      If a custom attribute is needed for the KPI, you may need to create your own copy of that cube to include that attribute as either a dimension or a measure. Also, if a new object is created for the KPI, a new cube is required for that object. For information on these topics, see Defining Models for Business Intelligence and related training material.
3. Map the KPI specification to KPI definition JSON structure. See the KPI configuration example.
4. Create the KPI definition using the API, and verify the created KPI definition by another API call.
5. Start using the newly created KPI definition in your application through KPI value API or KPI listing API.

11.3 KPI Business Logic

When collecting KPI business logic information, consider the following:

- What does the KPI measure? Is the goal to measure the number of transactions, or revenues impacted, or average time for specific tasks? Once the metric is defined, make sure it is represented as a cube measure. Once the cube measure is mapped, there are two possible variations for the measure:
    - Raw data. The KPI value is the exact value as what the cube measure defines, such as count, revenue, average time, and so on.
    - Percentage. The KPI value is a percentage, defined by the ratio of two numbers. In this case, business logic is required for both numerator and denominator. For example, the numerator might be the number of shipments expected to be late, and the denominator might be all shipments in transit.
- What is the primary business entity the KPI is measuring? A KPI can measure late shipped orders based on sales orders, or a KPI can measure late shipped shipment based on supply shipments. Such business entity needs to be mapped to an object in the Supply Chain Orchestrator data model.
- What are the KPI conditions? KPI conditions refer to the business logic which defines the conditions of the interests. For example, the KPI conditions for a late shipment KPI can be: shipments in transit with expected time of arrival later than the requested time of arrival. The tutorial shows how to capture such conditions in technical terms for the KPI definition. A couple of additional considerations:
    - The KPI condition can be empty. In this case, the KPI considers all records. For example, total order revenue may have an empty KPI condition, as it is about the sum of revenues from all orders.

  - In case the KPI measure is of type percentage, two set of conditions are required: one for the numerator, and the other for the denominator.

- How will the KPI data be further processed? You can slice and dice a KPI in different ways; specifically, you can configure the dimensions by which the KPI can be broken down. For example, you may want to see the breakdown of a late shipment KPI by supplier, carrier, and ship-to location. In this case, KPI dimension for supplier, carrier, and ship-to location can be defined. Each KPI dimension must have a corresponding cube dimension, but not all cube dimension need to map to a KPI dimension, i.e. KPI dimension is a subset of cube dimension.
- What are KPI threshold values? Thresholds can be defined to indicate how good or bad the measured aspect is for your business. Thresholds can be used to trigger notifications, or design visuals in the UI (such as green/yellow/red color scheme). Two threshold values are supported for each KPI: watching threshold (from green to yellow), and warning threshold (from yellow to red).
- Other descriptive information about the KPI. You can give each KPI a unique name (used as a key in APIs), a label (displayed name in UI), a description, and a status (active or not).

See the KPI configuration example.

For KPIs intended to capture risks or concerns in your supply chain, you can leverage the KPI definition to automatically generate issues which can then be tracked by the platform. For example, you can use the late delivery KPI to create issues automatically for shipments that meet the condition defined by the late delivery KPI. In such a case, you should consider an additional set of parameters for the KPI definition:

- Issue generating indicator
- Default issue severity level
- Business process name for issue analysis

11.4 See Also

- Supply Chain Solution Overview
- Issue Management
- KPI Configuration Example
- API Reference

12

KPI Configuration Example

This page shows an example of the process of defining and configuring a KPI for InterSystems Supply Chain Orchestrator™.

This example only covers KPI configurations using the provided analytics cubes. KPI implementations using other methods will be added in the future.

Also see the online learning path Building Smart Real-Time Supply Chain Applications, which includes information on creating KPIs.

12.1 Requirement Gathering

This example focuses on a KPI to capture shipments (from suppliers) that are expected to be delivered early. The following table captures more information about the logic needed by this KPI:

KPI feature	Requirement details
Name	ExpectedEarlyDeliverySupplyShipment (no space in name)
Label	Supply Shipment with Expected Early Delivery
Description	Supply shipments which are expected to be delivered early
Business entity	Shipments from suppliers
KPI measure	Shipment count
KPI measure type	Percentage
KPI conditions (numerator)	Shipments in transit, and ETA is more than 4 hours early than the requested delivery time
KPI conditions (denominator)	All shipments in transit
KPI dimensions	Users will want to break down the KPI by supplier, carrier, and ship-to location
Issue indicator	Issues should be generated for shipments meeting the KPI condition, with default severity level as 2

12.2 Requirement Mapping

With the business requirements collected in the table above, we can conduct the following mapping:

1. Specify the data model object to look at. The existing supply chain data model does support supply shipment entity, with object `SupplyShipment`. `SupplyShipment` object also has the required attributes for ETA (`estimatedTimeOfArrival`) and requested delivery time (`requestedTimeOfArrival`), thus no data model customization is needed. To learn more about data objects and attributes of each object in the supply chain data model, refer to data model API documentation. The result of this mapping gives us this part of the KPI definition:

   ```
   {
       "baseObject": "SupplyShipment"
   }
   ```

2. Specify the cube to use for this KPI. Most of the KPI logic is implemented in cube design and cube mapping. For this step, InterSystems IRIS® provides powerful tools that support the following techniques, accessible to non-technical users:

   - Understanding the analytics cubes. KPI logic is based on the predefined analytics cubes; specifically, you use cube queries to identify sets of records and measures of interest. The prebuilt cubes provide easy access for information on common supply chain entities, such as sales orders, supply shipments, and inventory.
   - Cube mapping. In general, the cube whose primary base object matches the data objects for the KPI base entity should be used. In our example, we use the out-of-box cube `SupplyShipmentCube`, because its base object `SupplyShipment` matches the business entity shipments from suppliers. The result of this mapping gives us this part of the KPI definition:

   ```
   {
       "deepseeKpiSpec":
       {
           "cube": "SupplyShipmentCube"
       }
   }
   ```

3. Specify the measure to use for this KPI. As mentioned above, a KPI measure must be a measure of the cube. The example KPI uses shipment count as the KPI measure. By default, all cubes have a measure for count. To find the exact name for the measure to use in a KPI definition, do the following:

   a. Open the cube in the Business Intelligence Analyzer.

   b. Drag and drop the measure from the left area (which shows the cube contents) to the **Measures** box on the right.

   c. Click the **Display Query** button on the toolbar. This displays the MDX Query window, which will then show a simple query like this example:

   ```
   SELECT [Measures].[%COUNT] ON 0 FROM [SUPPLYSHIPMENTCUBE]
   ```

   In this case, the name of the measure is `%COUNT`

   d. Use the measure name as the value of the `kpiMeasure` attribute within the KPI definition. The result is this part of the KPI definition:

   ```
   {
       "deepseeKpiSpec":
       {
           "kpiMeasure": "%COUNT"
       }
   }
   ```

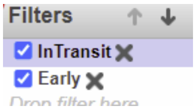
4. Describe the condition (possibly with multiple characteristics) that the KPI identifies. Ultimately the KPI condition is expressed as a set of MDX member expressions, which the system automatically combines via a logical AND. For common conditions, such as delivery status or order status and so on, the out-of-box cube design provides suitable dimensions. You can use the Analyzer to get these expressions:

   a. Clear any previous work in the Analyzer. (One option is to press the **New** button.)

   b. Expand the cube dimensions on the left, and drag and drop from there to the **Filters** box.

      For this example, the `SupplyShipmentCube` contains the dimension `Actual Time of Arrival`, which contains the members `Delivered` and `InTransit`. We drag `InTransit` and drop that into the **Filters** box. The same cube also contains the dimension `Estimated vs Actual Time of Arrival`, which has a `status` level that contains the members `Early`, `OnTime`, and `Late`. We drag `Early` and drop that into the **Filters** box.

      The **Filters** box then looks like this:



Filters

☑ InTransit

☑ Early

Drop filter here

   c. Click the **Display Query** button on the toolbar. This displays the MDX Query window. In this case, it shows the following query, shown with an artificial line break:

```
SELECT  FROM [SUPPLYSHIPMENTCUBE] %FILTER
NONEMPTYCROSSJOIN([estimatedVsRequestedDelivery].[H1].[status].&[Early],[actualTimeOfArrival].[H1].[value].&[<null>])
```

      In this case, the pivot table shows a crossjoin of the two MDX members we will use:

      - `[estimatedVsRequestedDelivery].[H1].[status].&[Early]`
      - `[actualTimeOfArrival].[H1].[value].&[<null>]`

      Notice that these are separated by a comma within the NONEMPTYCROSSJOIN function.

   d. Use these member expressions as items in the `kpiConditions` list, as in the following example:

```
{
    "deepseeKpiSpec": {
        "kpiConditions": [
            "[estimatedVsRequestedDelivery].[H1].[status].&[Early]",
            "[actualTimeOfArrival].[H1].[value].&[<null>]"
        ]
    }
}
```

5. If the KPI has the value type of `percentage`, also describe the *base condition* (for the denominator). The steps are the same as those in the previous step, except that now you specify the `baseConditions` property, as in the following example.

```
{
    "deepseeKpiSpec": {
        "baseConditions": [
            "[actualTimeOfArrival].[H1].[value].&[<null>]"
        ]
    }
 }
```

6. Identify the KPI dimensions, which are the possible breakdowns of the KPI values. Each KPI dimension is expressed as an MDX level expression. You can use the Analyzer to get these expressions:

   a. Clear any previous work in the Analyzer. (One option is to press the **New** button.)

b. Drag and drop the dimension or (level within a dimension) from the left area to the Rows box on the right. For example, drag and drop `carrier`.

c. Click the **Display Query** button on the toolbar. This displays the MDX Query window. In this case, it shows the following query, shown with an artificial line break:

```
SELECT NON EMPTY [carrier].[H1].[name].Members ON 1 FROM [SUPPLYSHIPMENTCUBE]
```

In this case, the level expression is `[carrier].[H1].[name]`

Members is the MDX function that displays the members, and you do not need that part of the syntax.

d. Use this level expression within an object in the `kpiDimensions` array in your KPI definition as follows:

```
{
    "deepseeKpiSpec": {
        "kpiDimensions": [
            {
                "name": "carrier",
                "label": "Carrier",
                "cubeDimension": "[carrier].[H1].[name]"
            }
        ]
    }
}
```

In this object, the `cubeDimension` attribute must be the level expression you found, the `name` attribute is the local name of this KPI dimension for use in APIs, and the `label` attribute is only for display purposes.

e. Repeat this step to get other KPI dimensions as needed.

7. Other KPI configuration data. This tutorial has covered the difficult part of the KPI configuration. The remaining should be fairly straightforward, as no additional mapping is required, and most information are collected in the business requirement section such as KPI name, description, and so on.

12.3 Final Result

The final KPI definition looks like the following:

```
{
    "name": "ExpectedEarlyDeliverySupplyShipment",
    "label": "Supply Shipment with Expected Early Delivery",
    "description": "Supply shipments which is expected to be delivered early",
    "baseObject": "SupplyShipment",
    "status": "Active",
    "watchingThreshold": 5,
    "warningThreshold": 10,
    "issueKpi": true,
    "defaultIssueSeverity": 2,
    "analysisService": "Early Arrival Process",
    "type": "DeepSee",
    "deepseeKpiSpec": {
        "namespace": "SC",
        "cube": "SupplyShipmentCube",
        "kpiMeasure": "%COUNT",
        "valueType": "percent",
        "kpiConditions": [
            "[actualTimeOfArrival].[H1].[value].&[<null>]",
            "[estimatedVsRequestedDelivery].[H1].[status].&[Early]"
        ],
        "baseConditions": [
            "[actualTimeOfArrival].[H1].[value].&[<null>]"
        ],
        "kpiDimensions": [
            {
                "name": "carrier",
                "label": "Carrier",
                "cubeDimension": "[carrier].[H1].[name]"
            },
            {
                "name": "supplier",
                "label": "Supplier",
                "cubeDimension": "[supplier].[H1].[name]"
            },
            {
                "name": "toCountry",
                "label": "Ship to country",
                "cubeDimension": "[shipToLocation].[H1].[country]"
            }
        ]
    }
}
```

You can use the following API to submit the JSON payload to create the KPI:

```
POST {{IRIS-SERVER-URL}}/api/scbi/v1/kpi/definitions
```

Once the KPI is defined, you can query the KPI value with API calls, such as:

```
GET {{IRIS-SERVER-URL}}/api/scbi/v1/kpi/values/ExpectedEarlyDeliverySupplyShipment
```

12.4 See Also

- Supply Chain Solution Overview
- Building Smart Real-Time Supply Chain Applications (which includes information on creating KPIs)
- Configuring KPIs for InterSystems Supply Chain Orchestrator
- API Reference

13

Introduction to the KPI API

InterSystems Supply Chain Orchestrator™ provides an analytics API for the supply chain KPIs, including the initial configuration step. This page introduces the API calls that apply to KPIs.

13.1 KPI Definitions

To find all defined KPIs, use the following API (no parameter required)

```
GET {{IRIS-SERVER-URL}}/api/scbi/v1/kpi/definitions
```

To define a new KPI, use the following API call:

```
POST {{IRIS-SERVER-URL}}/api/scbi/v1/kpi/definitions
```

With the following JSON in the message body:

```
{
    "name": "KPI name",
    "description": "KPI description",
    "type": "DeepSee",
    "baseObject": "SC base object for the KPI, such as SalesOrder",
    "status": "Status value, such as Active/Inactive",
    "watchingThreshold": xx,
    "warningThreshold": yy,
    "deepseeKpiSpec": {
    "namespace": "IRISnamespace",
    "cube": "KpiCubeName",
    "kpiMeasure": "cubeMeasure",
    "valueType": "raw/percentage",
    "kpiConditions": [
        "MDX condition 1",
        "MDX condition 2"
        ],
    "kpiDimensions": [
            {
                "name": "dim1",
                "cubeDimension": "MDX dimension expression"
            },
            {
                "name": "dim2",
                "cubeDimension": "MDX dimension expression"
            }
        ]
    }
}
```

Common KPI definition attributes are:

Attribute	Required	Description
name	Yes	KPI name, alphabetic string without any space or other characters. name must be unique within the same deployment.
description	No	KPI description
type	No	Type of KPI definition. Currently only one type is supported: DeepSee. More can be supported in the future.
baseObject	Yes	The supply chain object the KPI is based on, such as SalesOrder or SupplyShipment.
status	Yes	The status of the KPI definition, such as Active, Inactive, etc.
watchingThreshold	No	Threshold value for the watching level
warningThreshold	No	Threshold value for the warning level
resolutionService	No	Host name (within the supply chain production) of the business service that performs issue resolution for this KPI

Here are detailed descriptions of the KPI specification:

Attribute	Required	Description
cube	Yes	cube name
valueType	No	Value type, must be `raw` or `percentage`. Default value is `raw`, which means the same value as provided in the corresponding cube measure.
kpiMeasure	No	Cube measure in MDX format for the KPI. If not specified, the corresponding Count measure of the cube is used.
kpiConditions	No	List of MDX filter values, which is used to define KPI conditions. If no value is provided, the KPI gets the total measure, such as total number of orders, or total revenue.
baseConditions	No	This is used when the KPI value type is `percentage`. This set of MDX filter values determines the denominator of the KPI, while the KPI numerator is determined by combining this set of conditions together with the set of conditions defined in kpiConditions
kpiDimensions	No	This attribute is used to keep track of a set of dimensions that can be used in the scope of the KPI. The name of each dimension is used to do KPI filtering or breakdowns.

There are other API calls related to KPIs:

- Get the definition of a specific KPI:

  ```
  GET {{IRIS-SERVER-URL}}/api/scbi/v1/kpi/definitions/{KPI_NAME}
  ```

- Update a KPI definition. Use this API call with body with same format as used for creating a new KPI.

  ```
  PUT {{IRIS-SERVER-URL}}/api/scbi/v1/kpi/definitions/{KPI_NAME}
  ```

- Delete a KPI definition

  ```
  DELETE {{IRIS-SERVER-URL}}/api/scbi/v1/kpi/definitions/{KPI_NAME}
  ```

13.2 KPI Values

To get a KPI value without any dimension expansion, use the following API call:

```
GET  {{IRIS-SERVER-URL}}/api/scbi/v1/kpi/values/{KPI_NAME}
```

The response may look like:

```
{
    "kpiName": "kpiName",
    "values": [
        {
            "label": "kpi",
            "value": xxx
        }
    ]
}
```

To get KPI values with breakdown by a dimension:

```
GET  {{IRIS-SERVER-URL}}/api/scbi/v1/kpi/values/{KPI_NAME}?expandDimension={dimName}
```

and the response may look like:

```
{
    "kpiName": "kpiName",
    "expandDimension": "dimName",
    "values": [
        {
            "label": "dim-value-label1",
            "value": x1
        },
        {
            "label": "dim-value-label2",
            "value": x2
        },
    ...
    ]
}
```

This API call can be used for UI to show a chart to see the breakdown of the KPI, such as late shipment by region.

13.3 Detail Listings for KPIs

To get the source records related to a KPI, you can use the following API call:

```
GET {{IRIS-SERVER-URL}}/api/scbi/v1/kpi/listings/{KPI_NAME}
```

The response is a list of the source records. For example, a late ship order KPI listing request returns a list of orders which were shipped late. This API call accepts typical pagination parameters as well as sorting parameters.

13.4 Filtering KPIs

For any KPI defined, one can apply additional filter to a API call to get KPI values or listings. An KPI filter is a REST parameter `kpiFilter`, with value in the form of `(dimName1,value1),(dimName1,value2),...`. You can have any number of filter name-value pairs as needed, but for a given dimension, there can be at most one name-value pair in a filter. Any dimension used in `kpiFilter` must be defined in the KPI definition, under `kpiDimensions`. Another limitation: if a dimension is used in `expandDimension`, the same dimension should not be used in the filter. Other than that, one can use `expandDimension` together with `kpiFilter` in the same API call to get KPI values.

The following is an example of a KPI request with a filter defined in the request:

```
GET
{{IRIS-SERVER-URL}}/api/scbi/v1/kpi/values/OrderLateShip?kpiFilter=(productFamily,iPhone),(region,EMEA)
```

which retrieves Late ship order KPI value for product family `iPhone`, in the region `EMEA`.

13.5 See Also

- Supply Chain Solution Overview
- Issue Management
- Configuring KPIs for InterSystems Supply Chain Orchestrator
- API Reference

14

API Reference

This page provides links to APIs that are relevant when you are creating a solution with InterSystems Supply Chain Orchestrator™.

14.1 Available APIs

InterSystems Supply Chain Orchestrator includes three APIs:

- Data Model API — Use this to browse the supply chain data model; extend the data model.

  Also see Introduction to the Data Model API.

- Data API — Use this to create, update, and delete supply chain data; create issues.

  Also see Introduction to the Data API.

- Analytics API — Use this to configure KPIs; access KPI values and data; run issue analyses and close issues; close workflows.

  Also see Introduction to the KPI API and Introduction to the Issue Management API.

14.2 Related APIs

The underlying InterSystems IRIS® data platform provides additional, related Business Intelligence APIs:

- Business Intelligence REST API — Use this to access analytics cubes (or subject areas), pivot tables, and detail listings.
- DeepSee.js — Use this in conjunction with front-end libraries to create interactive displays of analytics data.

14.3 See Also

- Supply Chain Solution Overview
- Analytics Cubes for InterSystems Supply Chain Orchestrator