QAD Cloud ERP Channel Islands
2019

Developer Guide

QAD Enterprise Platform

70-3391-QEP2019

QAD Cloud ERP

Channel Islands 2019

March 2019

QAD Inc.
100 Innovation Place
Santa Barbara, California 93108
Phone (805) 566-6000
http://www.qad.com

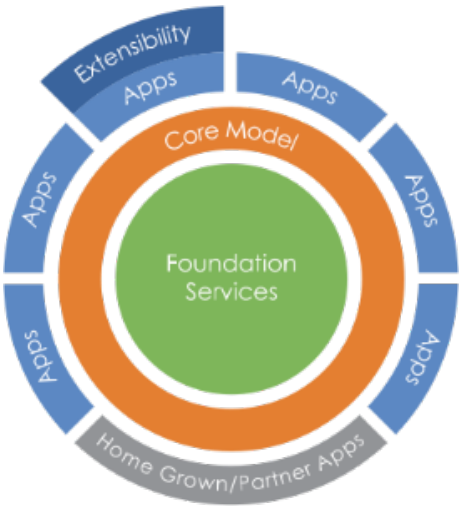
Table of Contents

QAD Enterprise Platform

The QAD Enterprise Platform enables you to extend QAD based on your unique business requirements.

The foundation services provide the core functionality, enabling the core model that consists of QAD apps and apps developed by QAD partners and customers.

In these pages, you will learn how to leverage the extensibility offered by the QAD Enterprise Platform and develop your own apps.



Overview

The QAD Enterprise Platform is a key aspect of QAD's Channel Islands program, a technology initiative that transforms the user experience, improving business outcomes through more effective users, processes, and decision-making. The Channel Islands program introduces a new user interface for accessing QAD Cloud ERP: the QAD Web UI. The Web UI provides an intuitive, engaging, and flexible user interface that optimizes and streamlines tasks while offering clear visual insights into business data. With the QAD Web UI, users interact with QAD apps to perform business tasks anywhere, anytime. For example, the QAD Sales app provides the resources for working with sales quotes, sales orders, and more. Underlying the Web UI is a new, agile architecture that can be extended to meet unique business requirements. As a QAD partner or customer, you can extend QAD using the capabilities of the QAD Enterprise Platform.

Build Future-Proof Apps

With the QAD Enterprise Platform, partners and customers can extend the QAD apps and innovate their own apps using QAD's low-code / no-code technology. As a partner or customer, you can extend the QAD apps to enable the unique processes that provide your competitive advantage. The QAD Enterprise Platform itself is available through the QAD Web UI, so you can extend QAD apps and build your own apps with the same convenience that all QAD Web UI users have, accessing the platform capabilities anywhere, anytime.

The extensions and apps that you build with the QAD Enterprise Platform are not only easy to create, they are also future-proof. The QAD Enterprise Platform uses a non-intrusive approach so that customers no longer need to worry about having intrusive customizations that could become obsolete or impact version upgrades.

Ready Now

The QAD Enterprise Platform is an extraordinary asset for QAD partners who can now develop add-on apps using this technology.

Next, learn more about:

- Recommended Prerequisites
- Architecture
- App Development

Recommended Prerequisites

Before you get started with the QAD Enterprise Platform, you should have a good understanding of **TypeScript**, **JavaScript**, **Chrome Developer Tools**, **OOABL**, **QRA Architecture**, **QRA Patterns** and other topics.

Free training about these topics is available on **YouTube**. Further, if you have access to **lynda.com** (lynda.com), the courses listed below are also recommended.

YouTube Training Courses

youtube.com						
Category	**Courses for beginners**		**Refresher courses**		**Advanced courses for more experienced developers**	
TypeScript	Introduction to TypeScript (Jess Chadwick) or TypeScript Tutorial (Derek Banas)	1h40m			Typescript tutorials - hackr.io	
Chrome Developer Tools	Chrome Dev Tools Tutorial (Anant Agarwal)	26m				
Optional - nice to know						
JavaScript	JavaScript Basics					
JavaScript / jQuery	JavaScript & jQuery Tutorials (LittleWebHut)	1h28m	w3schools on-line Tutorial and Reference		Advanced JavaScript Fundamentals	
AngularJS	AngularJS tutorial (Derek Banas)	40m				
Kendo UI	Kendo UI with AngularJS (tv.ssw.com) Getting Started With Kendo UI DataSource		Kendo UI Grid Tutorial and Reference			

Lynda.com Training Courses

lynda.com						
Category	**Courses for beginners**		**Refresher courses**		**Advanced courses for more experienced developers**	
JavaScript / jQuery	Introducing JavaScript Language	2h55m	w3schools on-line Tutorial and Reference			

Chrome Developer Tools	Chrome Dev Tools Tutorial (Anant Agarwal)	2h12m			Debugging the Web: JavaScript (lynda.com)	2h12m
Optional - nice to know						
TypeScript	Introduction to TypeScript (Jess Chadwick)	1h40m			Typescript tutorials - hackr.io	
AngularJS	Up and Running with AngularJS 1	1h13m				
Kendo UI	Kendo UI with AngularJS (tv.ssw.com) Getting Started With Kendo UI DataSource		Kendo UI Grid Tutorial and Reference			

OpenEdge Training Courses

QAD Enterprise Platform Extension Capabilities

The Platform can be used to extend and customize the standard Channel Islands product in the following ways within an application area:

- Creating own business components and screens
- Adding relations for business components
- Extending business component data
- Creating own browses
- Extending code for business components
- Extending code for the UI

Hands-On Activities

This guide includes a variety of hands-on activities, including:

- Developing a Custom App - Step by Step
- Examples - Step by Step

Architecture

The following topics are covered here:

Additionally, see also:

- Evolution of QAD Architecture

QRA Architecture

The Channel Islands program introduces a layered, services-oriented architecture, with separate layers responsible for presentation, business logic, data, and foundations services:

- **Presentation** — the presentation layer includes the components that implement the user interface.
- **Business Logic** — the business logic layer consists of the application business logic and exposes the functionality through business service APIs.
- **Data** — the data layer represents the data store of the application.
- **Foundation** — the foundation layer includes the functionality and services that are common to the various architectural layers (such as exception handling, logging, and so on).

The architecture is called the *QAD Reference Architecture (QRA)*.

The architecture is an **App**-based, modular architecture, and it is in the various apps that QAD's business-specific functionality is organized. Apps in turn are comprised of object-oriented **business components** (System >> Apps >> Business Components).

The Channel Islands program emphasizes user experience through the careful design of **metaphors** for interacting with the system. QAD carefully limits the number of metaphors in order to assure a cohesive, consistent, and easy-to-learn user experience.

Apps

Apps bring together related business activities. For example, the Sales app brings together Sales-related business activities with the Salespersons view, Sales Quotes view, Sales Orders view, and many more. The system is comprised of apps, where the QAD-provided apps include (not complete):

- **Foundation** — the Foundation apps provide the core technology used throughout the application, including the functionality of the QAD Enterprise Platform, Analytics / Action Centers, and more.
- **Base** — the Base app manages essential data needed in common throughout the application, such as Items, Suppliers, Customers, and much more.
- **Customer Management** — the Customer Management apps (e.g., Sales, Service) manage all phases of the customer lifecycle from acquiring customers, through managing orders, pricing, and fulfillment.
- **Manufacturing** — the Manufacturing apps (e.g, Product Structures, Production Lines, Inventory) provide tools for planning, scheduling, cost management, material control, shop floor control, and reporting in a variety of manufacturing environments.
- **Supply Chain** — the Supply Chain apps (e.g., Requisitions, Purchasing, Asset Management) support supply networks with the ability to drive margin and cost improvements, reduce lead times, increase inventory turns, and meet industry compliance.
- **Financials** — the Financials apps provides complete control and immediate information on any aspect of the organization's finances.

Business Components

A *business component* brings together the business logic and data necessary to represent a business activity within the application.

Users interact with business components through the user interface's menu items. These menu items are typically *hybrid views*, which combine a browse with a form where you create and interact with data based on the business logic. Typically, a hybrid view is based on one business component, but can use more than one business component. For example, the Address Types hybrid view is based on one business component, but the Sales Orders hybrid view is based on several related business components.

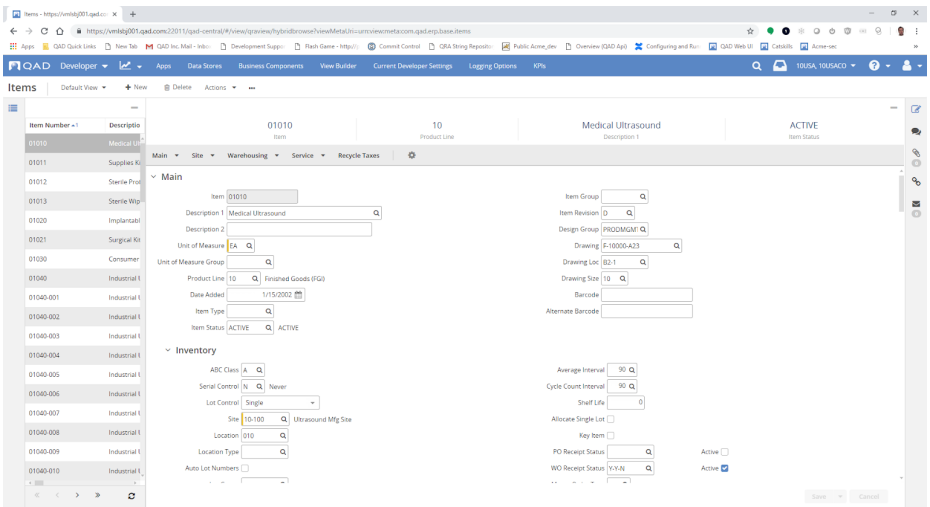
Business components provided by QAD (that is, business components in the "com.qad" environment namespace) cannot be modified.

Metaphors

A *metaphor* is a high-level user interface patten. As defined by the user experience guidelines, the main QAD Web UI metaphors currently include:

- **Browse** — the metaphor for displaying records in a grid
- **Form** — the metaphor that provides panels, sub-panels, and various types of fields for data entry
- **Hybrid** — this metaphor combines the Browse and Form metaphors
- **Report Viewer** — the metaphor for running reports
- **Action Center** — the metaphor for the new analytics dashboards
- **Dashboard** — the metaphor for the standard dashboards
- **Approvals** — the metaphor for managing an approval process

Hybrid View: an example



Views

A *view* is a user experience metaphor as rendered within the QAD Web UI. Views are user interface resources that you can launch from the menu bar or search. As a QAD Enterprise Platform developer, you will typically build one or more views using the hybrid metaphor; each *hybrid view* will be based on a business component you first create within the context of an app that you define.

Extensibility

As a developer, with the QAD Enterprise Platform you can extend the system by creating your own apps.

To learn about the evolution of the QAD architecture, see: Evolution of QAD Architecture.

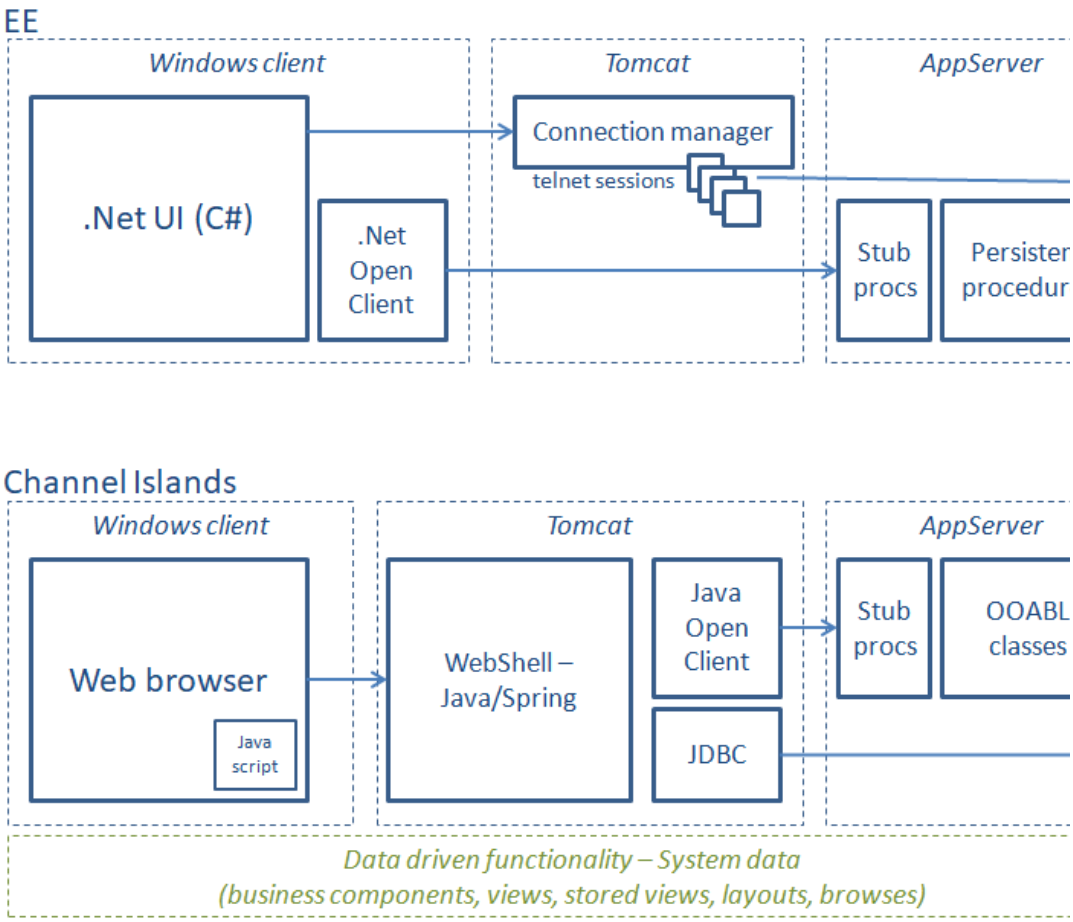
Next, get started with App Development.

Evolution of QAD Architecture

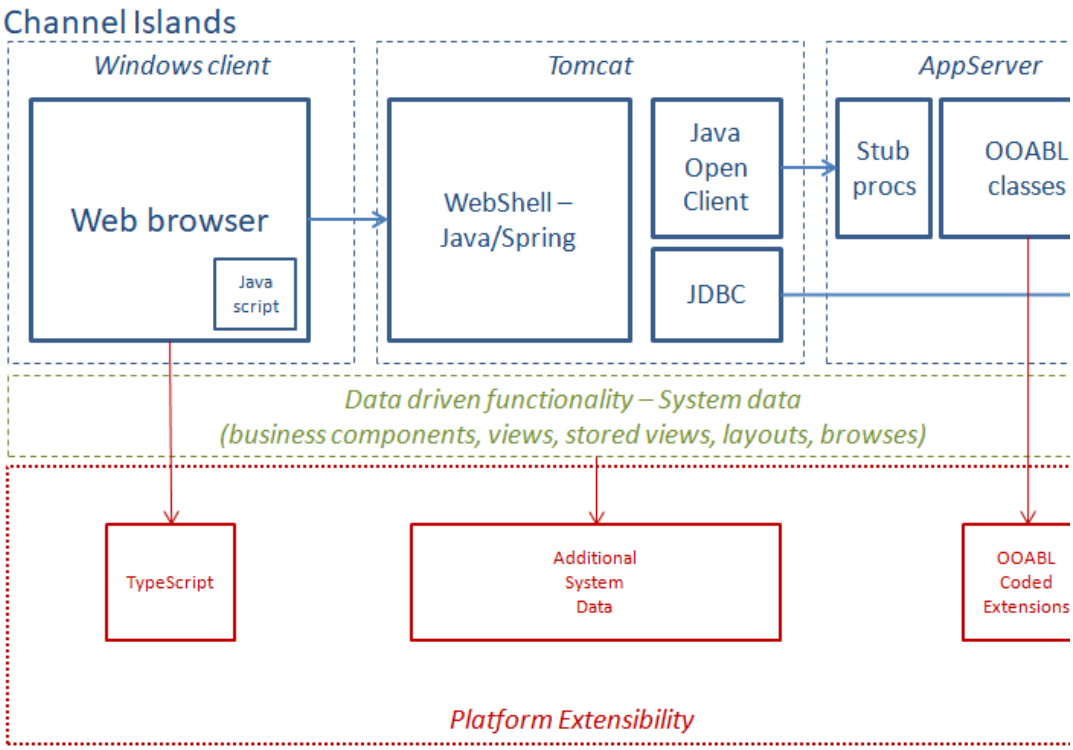
The following topics are covered here:

EE vs Channel Islands

The following diagram compares the architecture of QAD Enterprise Applications – Enterprise Edition (EE) with the architecture of Channel Islands.



Platform and Channel Islands


Channel Islands
Windows client
Web browser
Java script
Tomcat
WebShell – Java/Spring
Java Open Client
JDBC
AppServer
Stub procs
OOABL classes
Data driven functionality – System data
(business components, views, stored views, layouts, browses)
TypeScript
Additional System Data
OOABL Coded Extensions
Platform Extensibility

App Development Concepts

This section provides short explanations for important QAD Enterprise Platform concepts. These high level explanations are important for a good understanding of app development with the QAD Enterprise Platform.

Frequently Used Acronyms

These acronyms are frequently used:

- **UI** — User Interface
- **BL** — Business logic
- **BC** — Business Component
- **BCE** — Business Component Extension (Embedded Business Component)
- **OOABL** — Object Oriented Advanced Business Language (Progress development language)
- **TS** — TypeScript
- **URI** — Uniform Resource Identifier
- **URN** — Uniform Resource Name

List of child pages:

App

The App is the main building block for the QAD Enterprise Platform. The best way to understand an App is to look at it as a package that can be installed on the Platform, and that contains data and programs that make functionality available for a certain functional area. For example QAD offers apps for sales and financials functionality. The YAB tool is used to add new Apps or versions of Apps to an existing environment.

Functionality in a given App can be called through the exposed APIs for the App. Apps can call each others' APIs, can extend business components, or add relationships between its own business components and business components from other Apps. In these cases, the App has a dependency on the other App.

Essential things to understand about Apps:

- Every App is uniquely identified by its App URI (for example: `urn:app:com.qad.sales`)
- The ability to maintain an App is managed by the environment namespace, which is set once at the moment the environment is defined (for example, QAD uses `com.qad`.) The typical format is: `com.<partner-or-customer-name>`)
- Platform programs will always make sure that data gets defined in a given App. This way we can make sure there is a single owner of the data, which is important when updates need to be done. This way we also know what data to extract when a new App package needs to be created.
- For customization purposes specific Apps can be created to contain the customizations. In general we use the term "extensions" for customizations, so these apps are often referred to as "Extension Apps". So a customer environment can exist of the QAD owned apps coming with the standard product, I19 localization Apps, partner extension Apps and customer extension Apps.
- An App can be completely data driven, in which case the App package only contains the data, but it can also contain code that provides further functionality. Typically QAD owned Apps contain both data and code, as most of the business entities are coded in OOABL (Progress).
- All platform maintenance functions for the system data in the App are using the App URI of the active app every time a new piece of system data is created (examples are: a new business component, a new view, a new stored view, and so on). The active App is determined as follows: if the active App is set for the current developer, use that. If that is not the case, use the App that is marked as "default" in the system.

Business Component

A business component provides the maintenance (create, read, update, delete) functionality for a well-defined set of data. Typically, the data is stored in one or more tables in the database. The business component provides everything that is needed to allow creation, modification, fetching, and deletion of the data (traditionally called the CRUD methods). Things like data validation, calculation, concurrency checking, and so on, are included in the framework that exposes business components.

Business components are exposed through the views in the Web UI, and are also present in the REST API layer for back-end integration purposes.

The structure of the data in the business component is always represented by its metadata. This is data in the database that provides all information for the business component (top-level information like the label and description and the information about the fields of the business component (with all detail information, such as label, format, required, and so on.)

Business components are defined in the context of an App. We often make a distinction between *coded* business components and *virtual* business components:

> **Coded business components** are business components that are brought in via the traditional development process that does not rely on the platform business component builder, and rather retrieves the metadata from annotations in the code.
>
> **Virtual business components** are business components that get defined in the platform business component builder, and later deployed. These business components do not have any manual code behind this and are completely data driven.

Additionally, we often use the term *embedded* business component:

> **Embedded business components** (previously called extension business components) are defined in an App for the sole purpose of extending the data for another business component. Embedded business components do not have a form (or view) associated with them and cannot be run stand-alone.

⚠ Because the term "business component" is quite long, we sometimes use the acronym **BC** instead.

Business component field

Every business component uses business component fields to hold the data. These fields represent the *logical model* for the business component. Often the fields map directly to fields in the database behind it, but the system allows the field in the database to have a different name than the business component field. In some situations, the business component field is not mapped to a field in the database, in which case the business component business logic code needs to handle the values in the field properly in the code.

A recent addition in the system is the support for *formula* fields. These fields are defined with a formula expression (an Excel style formula) that gets calculated with each fetch or operation on the screen, and can combine values from other fields, or can include aggregations on child records. In some cases, these formula fields can be persisted in the database.

Business Component Relation

Very often business components are logically related. This is expressed by defining business component relations. In a business component relation fields from two related business components are associated with each other, and extra free-form conditions can be added.

Relations between BCs can be used to generically add behavior to the system. The following types of relations can be distinguished:

- A lookup relation: This relation should be seen as a simple reference. It is only used for automatic lookups on fields and it also feeds into the easy addition of "drill" capabilities between browses that are representing business components. Typically these are relationships with a "N-1" cardinality (for example a SalesOrderLine has a field "ItemCode", which drives the N-1 BC relation between SalesOrderLine and Item).
  This type of relationship is used by the system to provide automatic enhancement of data by enabling adding fields from the lookup related BC on a view.
- A parent-child relation: This relation indicates a closer connection between 2 BCs. When a BC is a child of another BC, a form can be built that automatically knows how to relate data between different pieces of the screen/form. To help determining how the data should be handled when data from parent-child related BCs are on one form, we also allow BCs to be marked as "embedded". If this happens, the system will automatically turn the child grid into an "internal" grid, which ensures the whole screen becomes one single transaction. In case of a non-embedded child the grid becomes an "external" grid and updates to the child records is done in separate transactions.
- An extension relation: The platform supports the extension of data for a given BC. This is done using extension relations. These can have a 1-1 or 1-N cardinality. The BC relation in this case is always between a business component and an embedded business component. For the 1-1 extension relations the system will automatically add these fields to the BC views and BC browses. For the 1-N extension relations these are added automatically as internal grids to the BC view (and as a consequence the data for it is always updated with the same transaction as the BC it extends).

Business Component Browse

The platform comes with a new implementation of browses, where the browse is based on a business component. We call these business component browses.

The main difference with the classical browses is that they are **completely data driven** and use all the metadata available for the business component it represents. By definition, a business component browse is always linked to one business component (BC).

Business component browses are using **direct SQL queries** to the database, so they do not have any progress code behind it.

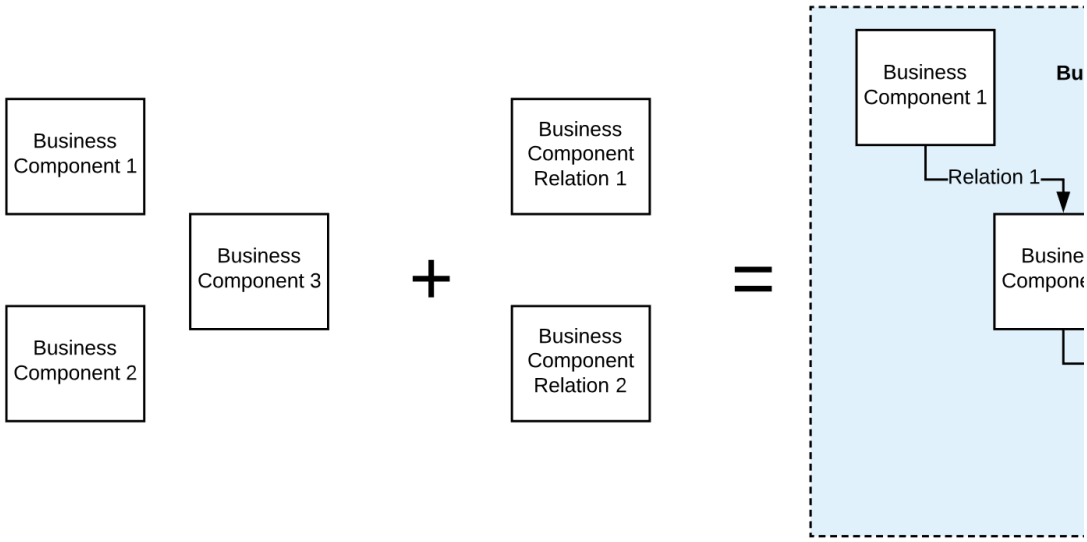
In general, a business component browse can have:

- Fields from the top-level table of the BC
- Fields from the top-level table of a BC that is related in a 1-1 fashion to the original BC
- Fields from an embedded BC for the BC represented in the browse
- Fields from the main table of a BC that has a lookup relation to the BC represented in the browse

Business Document

A business document is a collection of related business components, mainly used for integration with the application. The relationships involved are such that they make logical sense to be grouped together as a single business component, for example, the SalesOrderHeader and the SalesOrderLine. While the SalesOrderHeader may have a relationship with a Customer, they should not be considered to be in the same business document.

Business documents enable integration to work with the entire business document (a collection of related business components) rather than just one business component at a time. For example, when integrating with the Sales Order business document, the integration will work with all business components included in the Sales Order business document, such as SalesOrderHeaders, SalesOrderLines, and Notes all at once instead of three individual integration processes.



Hybrid Browse

The following topics are covered here:

Definition

A hybrid browse provides the most common way to interact with a business component (BC) from the Web UI.

The hybrid browse consists of:

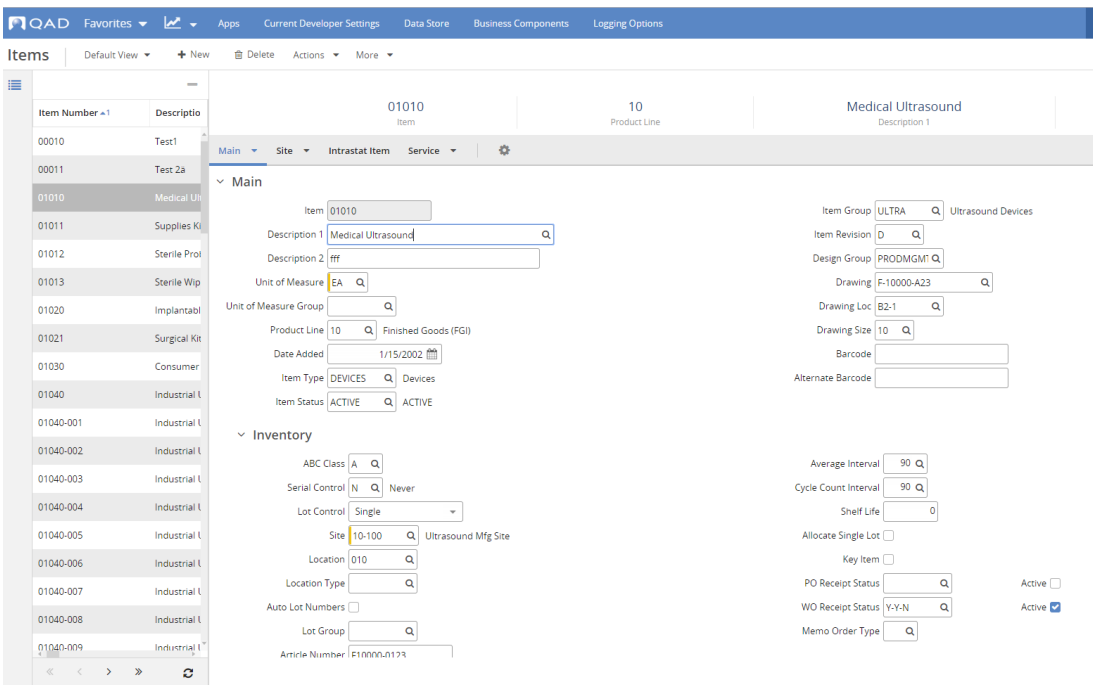
- a **browse** in which every line represents a BC instance (for example customer with code "10-100" is an instance of the Customer BC)
- a **form** that appears when a line in the browse gets selected by double-clicking, or the user selects "Edit" or "New". This form represents the BC and visualizes all information available for the BC, as defined in the BC definition itself.

Hybrid browses are represented in the system by their **metadata** and get defined in the **view builder tool**.

- The form view can be defined in the form view builder (which is part of the view builder), and the user can pull in data for the business component metadata, based on the business component's metadata.
- The browse for the hybrid browse can be specified in 2 different ways:
  - by defining a new BC browse
  - by selecting an existing browse (EE legacy browse).

Example

This is an example of a hybrid browse, where you can clearly see the browse on the left-hand side and the form on the right-hand side.



Logical and Physical Data Model

We use the term "physical data model" to refer to the tables in the physical database. Because many of the tables in the traditional mfgpro database can be cryptic, we use a "logical data model" that provides easier to understand names. The logical data model is exposed through the business components.

All functionality for the Platform apps is built against the logical model. The business component implementation is taking care of the mapping between the logical and the physical data model. This mapping is often referred to as **Entity Mapping**.

System Data

System data is all data that needs to be available in the environment to guarantee the functionality provided by the Apps work correctly in a Platform environment. A lot of generic capabilities like browses, forms, drills, lookups, and so on are dependent on system data.

The system data gets packaged with the App packages in the form of XML files.

The system data gets automatically loaded when an App package is brought into an environment through a YAB recipe, or through a `yab install` command.

Data Store

A data store is an abstraction of any data source that can be used by the business components to store or retrieve data. A defined data store needs to be able to provide an implementation for methods like "Fetch", "Create", "Update", and "Delete". Eventually data stores can have multiple possible types, but currently only the "Progress database" type is implemented.

Any Progress database can be defined as a data store in the platform. In general the mfgdb or any of the other standard QAD databases will only be available as "production" data stores. Production data stores are accessible for reading and writing, but they cannot be used to create new structures to support business components. For these at least one "development" data store needs to be available in the environment.

New types can only be introduced by Foundation development.

Environment Namespace

The following topics are covered here:

Definition

The Environment Namespace is a setting that needs to be specified in the `environment.xml` file (see below), and is used to indicate the area of **ownership** for developers in the installed QAD environment.

For example, by setting the environment namespace to `com.abcCompany`:

- Apps for which the URI matches the environment namespace, in our example the App `urn:app:com.abcCompany.Machines` is matching the environment namespace, and therefore developers in this QAD environment will be able to maintain system data in the context of this App.
- When the App does not match the environment namespace, for example for the Apps `urn:app:com.xyz.OtherApp`, or `urn:app:com.qad.base` the developers will not be able to maintain the system data.

Setup

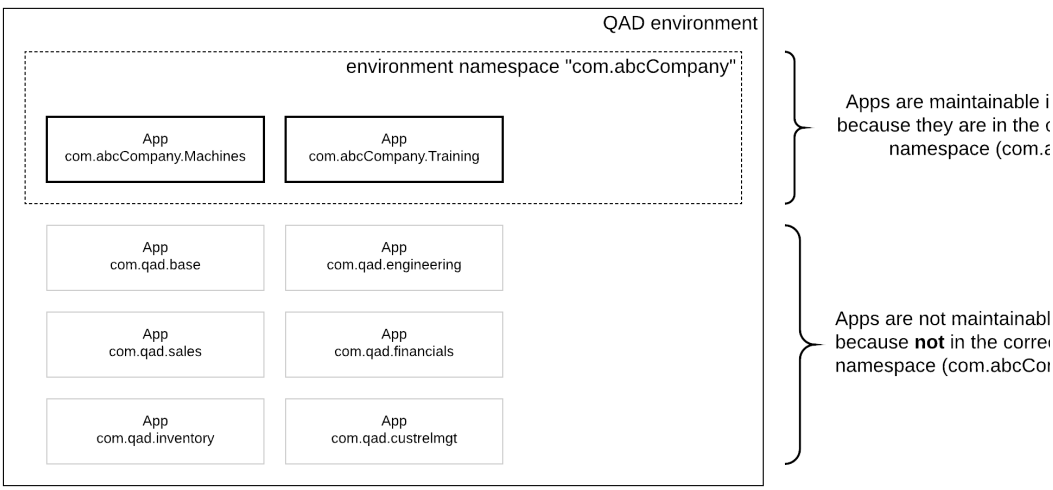
The official way to set the Environment Namespace value is using the yab tooling (`yab update`), providing following properties in the `build/config/`**`configuration.properties`** file. This step needs to be done upon the initial set up of the system.

```
qra.config.environment.environmentnamespace=com.abcCompany
qra.config.environment.defaultappname=Machines
```

The final result is in the `com.qad.qra/config/`**`environment.xml`** file, for example:

```
<Environment>
    ....
    <PlatformInfo>
        <EnvironmentNamespace Value="com.abc" />
        <DefaultAppName Value="Machines" />
    </PlatformInfo>
</Environment>
```

Environment view



Some related questions

1. *What guidelines do we use to choose an environment namespace for a given customer?*
   Typically customers have their own ".com" domain registered for their official communication through a website. Best would be to reverse the trailing entries, and use that as their environment namespace. For example for abcCompany, they use www.abcCompany.com for their website, so by default we would use "com.abcCompany" as environment namespace. This is something that might need to be discussed with the customer prior of setting up the environment or upgrading it.

2. *Do we need to guarantee that different customers or partners are not using the same namespace?*
   Yes, in order to ensure that customers, partners are not able to override the system data, they should not use the same environment namespace. It would be good if the cloud team manages a list of assigned environment namespaces.

3. *What if the customer has multiple EE instances should they all have the same namespace or different ones?*
   By default, they should all use the same environment namespace. Only in the situation where customers want to start creating apps that should be owned by different parts of the organization you could deviate from that, but that is only when CI is installed, for EE the environment namespace should always be the same.

4. *How do I set the default app after initial setup of the environment?*
   The `qra.config.environment.defaultappname` setting is taken into account when initially creating the environment. When you want to change the default app after this to something else, you need to use the "Apps" maintenance to set the default app to the new app.

Use of URIs in the Platform

To uniquely identify artifacts in the Platform, we make consistent use of URIs. This provides us with a consistent scheme for formatting identifier fields. For example business components, apps, business component relations, and so on are uniquely identified by their URI. Each maintenance function in the platform that allows users to create or edit artifacts will have the URI field.

The following table represents the typical formats of URIs for types of the most important artifacts in the platform.

Type of artifact	URI format	Example	Comments
App	`urn:app:<name>`	`urn:app:com.qad.base`	
Business Component	`urn:be:<name>`	`urn:be:com.qad.base.item.IItem`	The be in the URI format refers to the original name of business components ("business entity").

Browse	`urn:browse:<browse-type>:<name>`	`urn:browse:bebrowse:com.extensions.qadextensions.students`
		`urn:browse:mfg:mfg253`
		`urn:browse:fin:bgl.selectgl`
			"browse type" can be "fin", "mfg", "be" or "bebrowse", where only "be" and "bebrowse" can be created through the platform

Hybrid view ("Meta URI")	`urn:view:meta:<name>`	`urn:view:meta:com.qad.erp.base.items`	
Form view ("Maint URI")	`urn:view:maint:<name>`	`urn:view:maint:com.qad.erp.base.items`	This URI is used as the identifier for context-sensitive online help linking.
Hybrid browse	`urn:view:hybridbrowse:<name>`	`urn:view:hybridbrowse:com.qad.erp.base.items`	
Data store	`urn:datastore:<name>`	`urn:datastore:com.extensions.extension`	

YAB

Your Application Builder (YAB) is a console application that is used to create and administer QAD Enterprise Applications instances.

Instances are created from recipes (configuration documents) that specify the versions of application (and YAB) packages to include in the environment as well as settings describing how application resources should be setup and configured.

For more information on using YAB with the QAD Enterprise Platform, see Platform Development in YAB.

NOTES:

Also need to capture: App export/loading

Logstash and Kibana Configuration (YAB space link)

Datastores setup

Custom db

App Development

These pages give a complete overview of all possibilities of the QAD Enterprise Platform. The pages are developed to address extension developers in particular, but can also be used for people trying to get an understanding of the capabilities offered by the platform itself.

App Development Tools and Resources

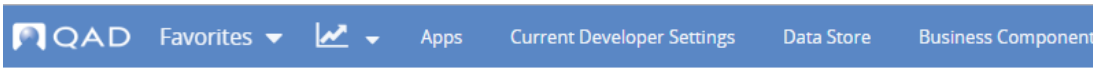
This section introduces all tools that are important for a developer when extending or adding new functionality to the system. The tools support all aspects of the development, packaging, deployment, and debugging of apps. These pages should be used as reference material for each of the tools. In subsequent sections, the examples and explanations will often refer to the pages in this section.

Platform Development in the Web UI

You can access the resources for platform development directly from the Web UI. Most of those resources are views, typically hybrid browses, which combine a browse display with a form that you can use to create and edit data.

When you set your role menu to QAD Admin, you can quickly access these views from the menu bar. You can also easily access them by entering their names in the menu search, located on the menu bar.

Or you could add them one by one to the Favorites:



The views include:

- Apps view
- Current Developer Settings view
- Data Store view
- Business Components view
- Logging Options view
- Lookup Definitions view

Menu items for managing roles, access, and security include:

- Roles view
- Role Permissions view
- User Access view
- Menus view

Additional menu of interest include:

- KPIs view
- Lookup Definitions view

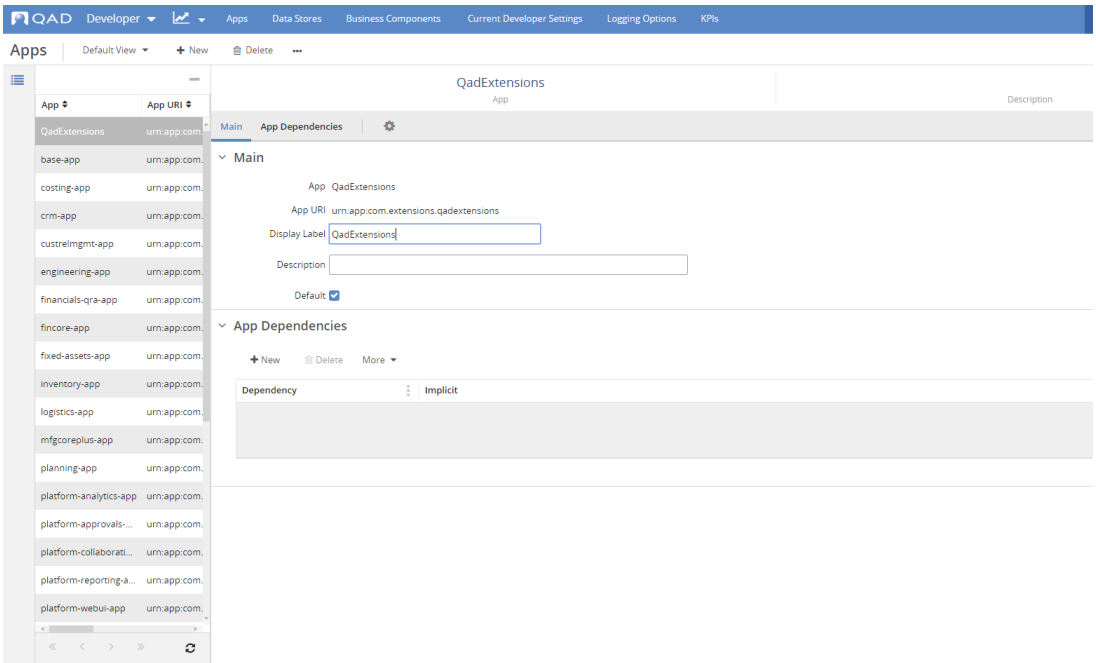
Apps view

IN PROGRESS

Introduction

With the Apps view, you can create, modify, and delete apps and app dependencies.

Link: App



What is an App?

An App is the container for a set of metadata that defines business components, extensions to business components, browses, form views for business components, and other artifacts needed for a functional extension. The app is also used to associate source code and compiled artifacts that provide customizations of the UI or BL flows.

Some of the metadata that can be defined in the context of an app is the following:

1. Business components metadata (created with Business Component Builder):
    a. Fields metadata
    b. Formula definitions
    c. Relations
2. Business component browse metadata
3. View data
    a. UI layout data (View metadata), this is the layout definition created with the view builder
    b. Browse data, this is all the metadata created when creating a BE browse and View
    c. Event handlers data: metadata and source code necessary to run event handlers

What are App Dependencies?

App dependencies are used to express any dependency from one app to another. If you for example create embedded bus iness components for Sales Order, then your app is dependent on the components in the Sales app.

Some dependencies are automatically recognized via business component relations, we call these "implicit" relationships. If you add a relation to a business component of an other app, your app will automatically (implicit) be dependent on the other app. Obviously the developer can also add dependencies explicitly.

The dependency on other Apps is required for extension coding in the App. Only if your App is dependent on another one, calls to the other App's services can be made. When exporting the App to start coding extensions, the yab tools provide a sandbox model that ensures all paths are including the necessary app APIs and all schema is available for building (compiling) code that accesses the database.

Creating, Editing, and Deleting Active App

You can create as many apps as you want, but make sure you set the correct one **active** when you want to start developing or customizing business components.

When deleting an app, all metadata belonging to that app will be deleted along, also customization data like event handlers will be deleted permanently.

UI Quick Reference

The Apps view includes the following panels and fields:

Main panel

App

Enter the app name. This represents the logical name of the app.

App URI

The app URI is read-only. The value gets auto-generated and is based on the environment namespace and the app name. This uniquely identifies the App in the environment.

The format of the URI is: urn.*app.environment_namespace.app_nam*e.
For example, if the environment namespace is "extensions" and the app name is "myapp", the app URI will be: urn.app.com.extensions.myapp.

Display Label

Enter the display label for the app. This is used anywhere in the system when the App is referenced on the screen.

Description

Enter a brief description of the app. This is used in the swagger site.

Default

Select to specify this app as the default app. The "default" app is by default the active app for any developer. The system can have only one active app for each developer at a time. New system data is stored in your active app.

In Current Developer Settings view, you can view and change your active app.

App Dependencies panel

An app can have implicit or explicit dependencies on other apps. Any implicit dependencies are listed automatically.

If you know that the new app must be dependent on another app, you can add that app here as an explicit dependency. For example, if the new app extends the capabilities of an existing app, then you can identify that existing app as an explicit app dependency by clicking **+New**, adding the app, and setting **Implicit** to No.

In case the developer wants to develop OOABL code for the extension app, there is a required dependency on the "qracore-app" app.

Current Developer Settings view

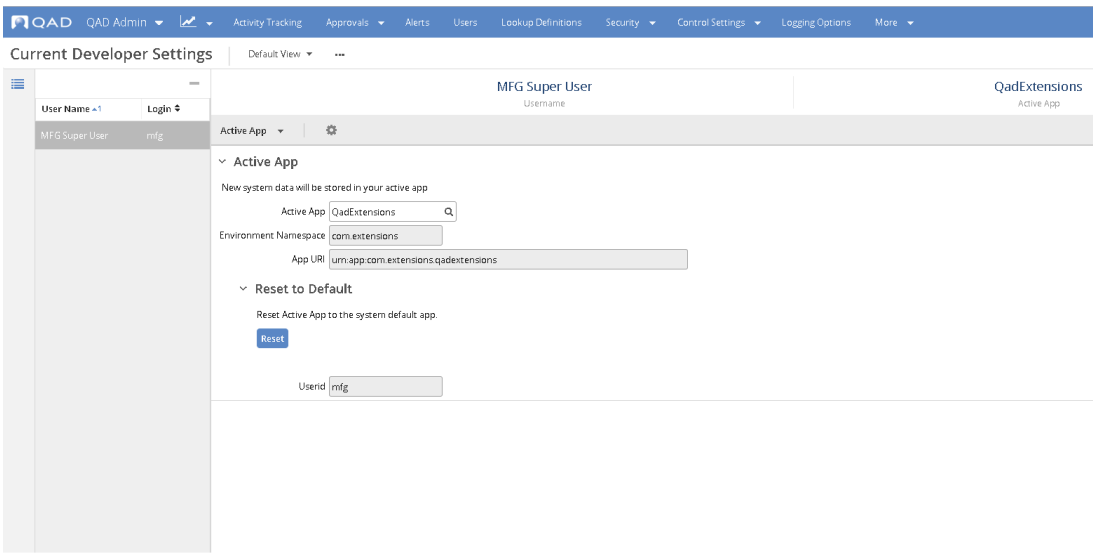
IN PROGRESS

Table of Contents:

Introduction

In Current Developer Settings, as a developer, you can select the active app or restore it to the default app for the current developer.

The UI looks as follows:



What is the Active App?

The active app contains all the business components and business component relations that the developer creates using the Business Components view. This also applies to any other platform function that is used to store System Data (such as the Form Builder, any stored views, or form layout changes using the Design Layout option).

Note that in the Apps view, the Default field indicates if the app you are viewing is the default, active app. You can change the default, active app here in the Current Developer Settings view.

UI Quick Reference

The Current Developer Settings view includes the following panels and fields:

Active App panel

Active App

The name of the currently active app for your development. You can select the available apps from the lookup. The lookup only shows the Apps that are defined within the environment namespace. Apps are defined using Apps view. New system data will be stored in your active app.

Environment Namespace

Displays the value of the environment namespace. This is always read-only, as it can only be specified using a YAB configuration (see environment namespace).

App URI

Displays the App URI of the currently active app.

Reset to Default panel

Click **Reset** to reset the currently active app to the system default app.

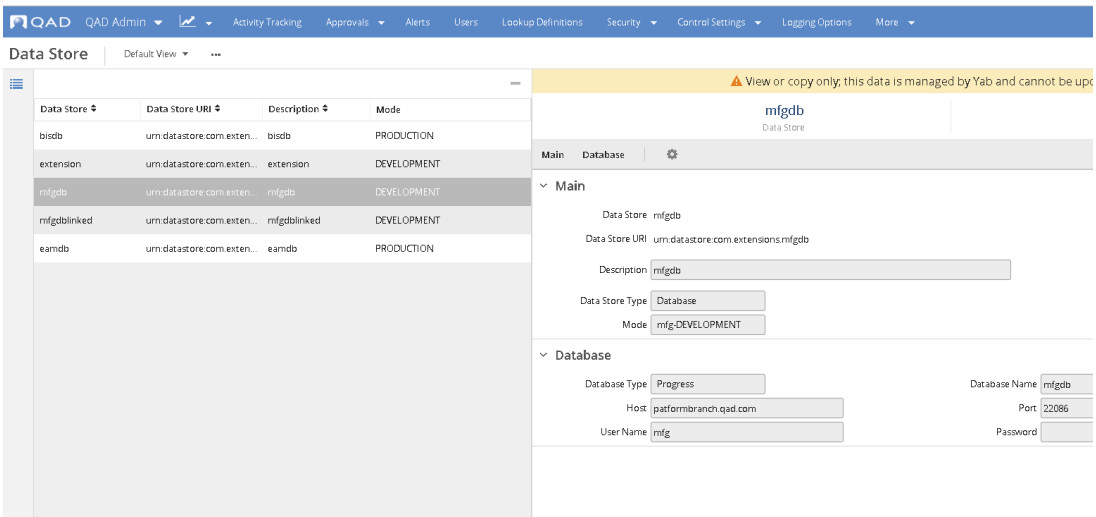
Data Store view

IN PROGRESS

Introduction

The Data Store view displays all the data stores available in the system and their settings. Data stores and their settings are read-only for the developer. They are maintained by the system admin using YAB.

Concept: Data Store



What is the Data Store?

Data Store is a place where the data of Business Components is stored. A data store defines the connection to (typically) a database, but it could also be a set of webservices or other ways to store or retrieve data. Currently, the platform only supports a Progress database as a valid data store. You can have several data stores in the system. Each data store has a Mode-property, which can have values DEVELOPMENT or PRODUCTION.

> You can deploy Business Components only to data stores with Mode=DEVELOPMENT.

UI Quick Reference

The Data Store view includes the following panels and fields:

Main panel

Data Store

The name of the data store.

Data Store URI

The Data Store URI, a unique identifier of the data store, will automatically be determined based on the entry in the Data Store field.

Description

A description of the data store.

Data Store Type

Indicates whether the data is stored as a Database or a File.

Mode

The mode of the data store: DEVELOPMENT or PRODUCTION. Business components can be deployed only to data stores in DEVELOPMENT mode, because only these data stores are available to inject new schema structures in.

Database panel

The Database panel includes the information needed for the JDBC driver to connect to the database data store.

Database Type

The type of database (currently only Progress is supported).

Host

The database host name.

User Name

The database user name.

Database Name

The database name.

Port

The database port number.

Password

The password to access the database.

Business Components view

IN PROGRESS

Table of Contents:

Child Pages:

Introduction

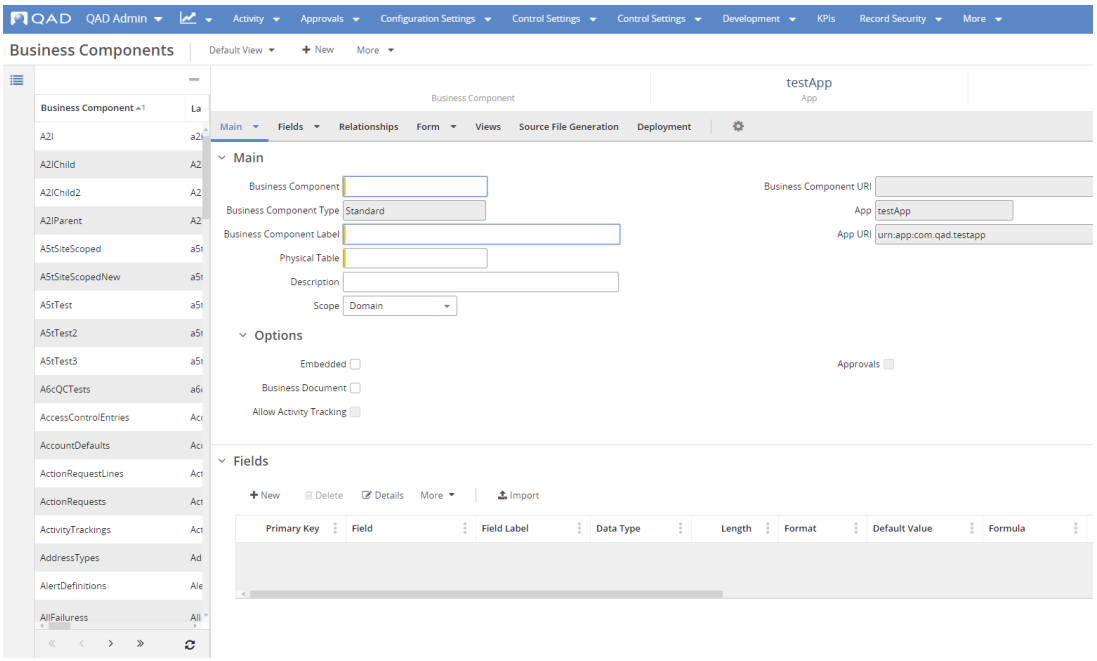
The Business Components view allows a developer to see the list of business components (BCs) in the system (including the ones coded using Progress), and to define new business components dynamically, from scratch. All BCs from all apps are visible, but only the BCs in apps that belong to the environment namespace can be modified; other BCs are read-only.

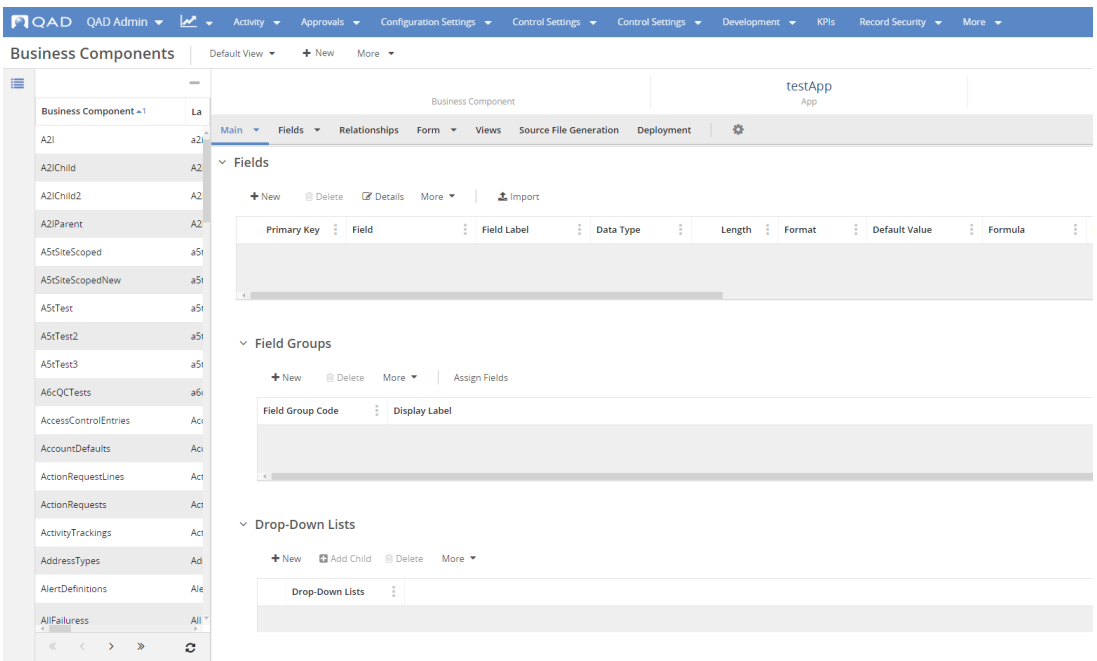
Link: Business Component

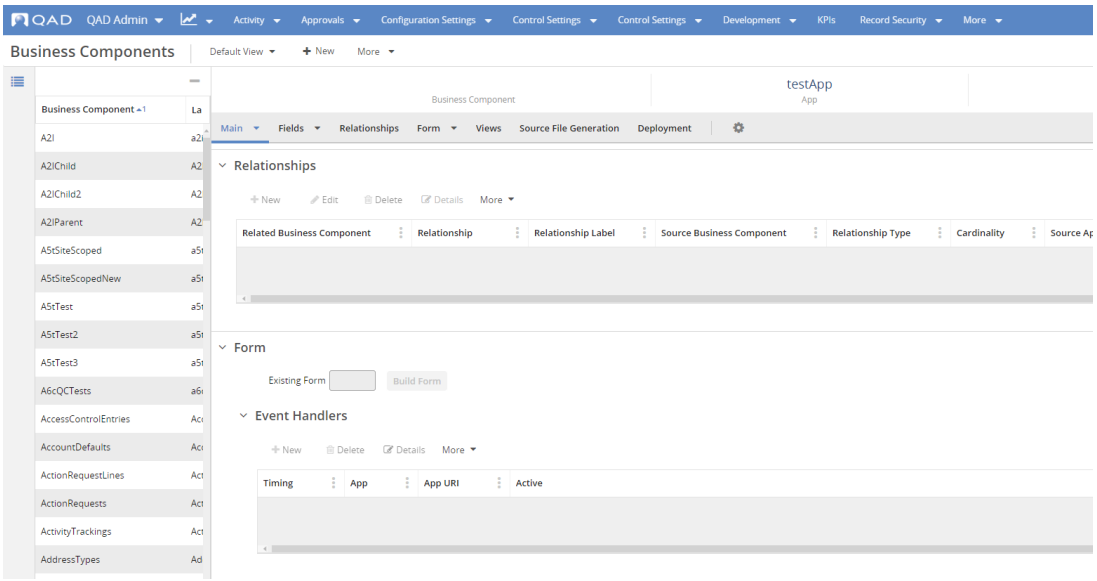
What is a Business Component?

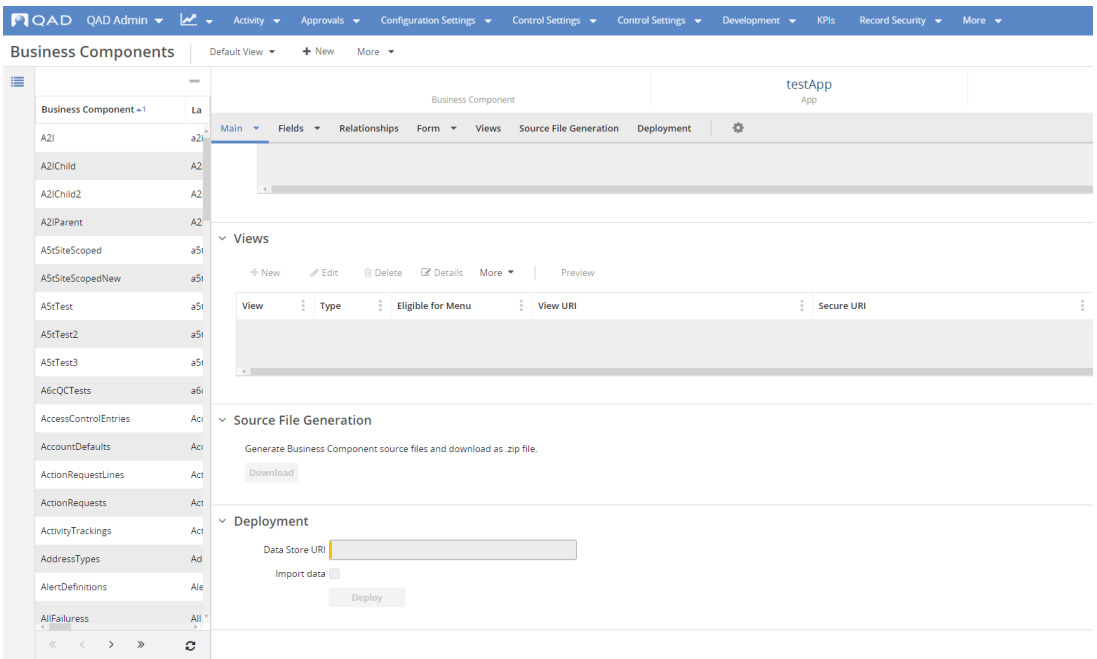
Business Components represent data that functionally belong together, which can be maintained in the system. The data related to BC is of two kinds: metadata describing the BC itself (tables, fields, fieldgroups etc.) and data of real instances of this entity. The Business Components form displays the metadata of a selected BC. For Progress-coded BCs, this metadata is always read-only information. Also, with the Business Components view, the developer has ability to create a new BC and then edit it as often as needed. Currently, BCs created using the Business Components form are single-table based. Upon deploying such BCs to some data store, the corresponding table is created in the database, and the component becomes available in menu search. If the user then goes to the View of newly created BC, the user can create and update some records of this BC there, and these records will be stored in the corresponding table in the database.

Example Screen Shots


QAD
QAD Admin
Activity
Approvals
Configuration Settings
Control Settings
Control Settings
Development
KPIs
Record Security
More
Business Components
Default View
New
More
Business Component
testApp
App
Main
Fields
Relationships
Form
Views
Source File Generation
Deployment
Main
Business Component
Business Component Type
Standard
Business Component Label
Physical Table
Description
Scope
Domain
Business Component URI
App
testApp
App URI
urn:app:com.qad.testapp
Options
Embedded
Business Document
Allow Activity Tracking
Approvals
Fields
New
Delete
Details
More
Import
Primary Key
Field
Field Label
Data Type
Length
Format
Default Value
Formula
Business Component
A2I
A2IChild
A2IChild2
A2IParent
A5tSiteScoped
A5tSiteScopedNew
A5tTest
A5tTest2
A5tTest3
A6cQCTests
AccessControlEntries
AccountDefaults
ActionRequestLines
ActionRequests
ActivityTrackings
AddressTypes
AlertDefinitions
AllFailuress


QAD
QAD Admin
Activity
Approvals
Configuration Settings
Control Settings
Control Settings
Development
KPIs
Record Security
More
Business Components
Default View
New
More
Business Component
testApp
App
Main
Fields
Relationships
Form
Views
Source File Generation
Deployment
Fields
New
Delete
Details
More
Import
Primary Key
Field
Field Label
Data Type
Length
Format
Default Value
Formula
Field Groups
New
Delete
More
Assign Fields
Field Group Code
Display Label
Drop-Down Lists
New
Add Child
Delete
More
Drop-Down Lists
Business Component
A2I
A2IChild
A2IChild2
A2IParent
A5tSiteScoped
A5tSiteScopedNew
A5tTest
A5tTest2
A5tTest3
A6cQCTests
AccessControlEntries
AccountDefaults
ActionRequestLines
ActionRequests
ActivityTrackings
AddressTypes
AlertDefinitions
AllFailuress





What is a Form?

A form organizes everything a user needs for some business task on a single page. In a form, a user can create, view, edit, and delete data. In a hybrid browse, which initially displays the full browse, the form opens when the user double-clicks a row (that is, a record) in the browse.

A form includes a summary panel, navigation bar, main panel, and then various detail panels and sub-panels.

What is a View?

The QAD Web UI menu offers *views* in which you can interact with the system. You can think of views as the QAD Web UI's web pages or like the screens in the QAD .NET UI.

Note that users can modify a view and then save it for later use as a *stored view*. Aspects of a view you can modify include the columns, search conditions, and form panels and fields.

Creating, Editing, and Deleting

You can create a business component by defining fields manually in the Business Components' Fields panel, or by first defining the fields in an Excel file that you then upload.

You can also create a business component based on an existing table in the database, and then optionally generate the Progress source code for the business component.

When creating a business component from an Excel file, the potential fields and their datatypes are inferred by the system based on column names and cell values in Excel file. You should then check and edit the proposed field set.

In order to save the business component, you must identify at least one field as the Primary Key. Primary key fields must be specified in their proper numerical order, where the order is specified as integers starting from 1. The numerical order identifies the order of fields in the complex primary key.

After the business component is saved, it can be edited as many times as you would like, until it is deployed. When a business component is being deployed, the corresponding table for it is created in the database, specified by the data store. Once the table has been created, you cannot delete fields from the business component definition or change the Primary Keys.

However, it is possible to add new fields to a business component after it has been deployed:

- if a Formula field is added, it can be used immediately;
- if an ordinary field is added, the business component becomes un-deployed and a user cannot run this business component from the menu until it is re-deployed again.

UI Quick Reference

The Business Components view includes the following panels and fields:

Main panel

Business Component

Enter the identifier of the business component. This identifier is used to construct the Business Component URI. This field becomes read-only after you first save it. This value should be unique within the current, active app.

Business Component Type

Indicates the business component type: Standard or Action. If you create a new virtual business component, Business Component Type is automatically set to Standard and disabled. The creation of a virtual Action business component is not supported at this time, but you can build part of a hand-coded Action business component using the Business Components > Form Builder screens.

- Standard. The Standard business component provides everything that is needed to allow creation, modification, fetching, and deletion of the data (traditionally called the CRUD methods). The Standard business component URI starts with urn:be.
- Action (Service). The Action business component provides methods that cover functionality outside the normal CRUD operations. The Action business component URI starts with urn:service. Note that the **@is be** annotation can be added to the interface definition to indicate that this is a Standard business component, not an Action business component.

Think of a Standard business component as an object to store state and business logic, and an Action business component as an API or set of functions/methods.

The Action business component has the following limitations:

- In the Views pop-up window, the only available view type is Form Only.
- The preview is not available from the Views grid.

If you build a form for a Bulk action:

- Add the Search Criteria and Required Criteria fields as labels.
- Add the Search button and the Reset Criteria button by using a two-column field group in the Form Builder. Note: The Reset Criteria button is optional.
- Custom buttons on Form Grids are not currently supported in the Form Builder. If you want to add the Clear Unselected Lines button to the grid, you can do it through the code.
- To include an option to simulate the behavior, you can add the Simulate button through the code.

Business Component Label

Enter the label name for this business component.

Physical Table

Enter the database table name where the records for this business component will be saved.

Description

Enter a description of the business component for informational purposes. For example, it is included in the Swagger-based documentation.

Scope

Select the scope of the entity: Domain, Entity, Site, or System. By selecting the scope, the system will automatically validate the existence of the appropriate scope field (respectively "DomainCode", "EntityCode", or "SiteCode") in the business component when it is saved.

Business Component URI

The business component URI is generated automatically based on the App URI and Business Component identifier. This is the identifier that uniquely identifies the BC overall apps.

App

The name of the currently active app for your development.

App URI

The App URI is set to the developer's active app as specified in Current Developer Settings view.

Options panel

Embedded

This checkbox indicates whether the business component is either standalone or can only be used as a data extension of some other business component, either as a one-to-one (1-1) extension or many-to-one (N-1) extension.

Business Document

Indicates whether a business document is enabled for this business component. A business document is a collection of related business components, mainly used for integration with the application. When this option is activated, the Business Document panel appears on the Business Components page.

Allow Activity Tracking

Indicates whether this business component is enabled for activity tracking. This field is read-only. Activity tracking can be enabled for any business component using the "Activity Tracking" menu option.

Approvals

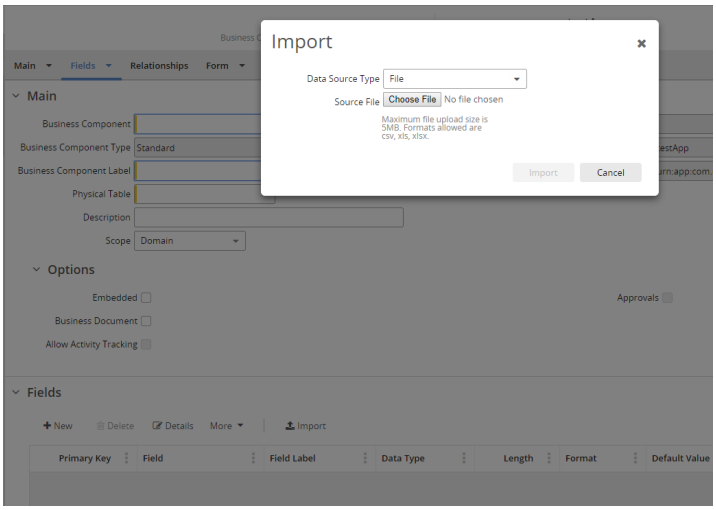
Indicates whether this business component is enabled for approvals. When this option is activated, the Approvals panel appears on the Business Components page, and the business component can then be used in an approval process (using generic approvals). You can configure the business component approval settings by using the Configure button on the Approvals panel. For more information about creating an approval for a BC, see Creating Approvals - Step by Step.

Fields panel

The Fields grid lists all the fields of the business component. (Note that this grid is an "internal" grid, so it is saved only when the entire business component definition form is saved.)

On this Fields grid, fields can be created, edited, or deleted. *Important:* To save the business component, at least one field has to be marked as the Primary Key.

The **Import** option () supports loading from a file or a database. You can import fields from an Excel (.xls or .xlsx) or comma-separated value (.csv) file. The maximum file size is 100 MB. The first row in the sheet gets interpreted to determine the field names, while the cells in the next rows are used to determine the data type and the format.



The **Formula** option allows you to include a formula for the currently selected field. Fields with Formula set to Yes can be edited by clicking the Details button, which opens the Fields pop-up window. Scroll down to the Formula panel. Here, you can specify an Excel-style formula for calculating the value of the field based on other fields (for more details, see detail page on formulas):

- Click Include Field to select fields. The fields that can be selected include the other fields of the current business component, fields from lookup related business components and typically for use with aggregation functions, fields from business components that are related with a parent-child BC relation.
- Click Include Operator to select the Excel-style mathematical operations to apply to the fields.
- Click Check Syntax to validate the syntax of your formula. Note that the field's Formula column must be selected to indicate the field's value is calculated based on a formula.

Primary Key

Identifies whether a field is a part of primary key. Primary key fields must be specified in the proper numerical order. The values put here should be integers starting from "1" and they are identifying the order of fields in case of a composite primary key.

Field

The code that identifies the field (the entity field code).

Field Label

Enter a display label for the field. This label will be used on the user interface, such as on form fields and browse columns.

Lookup

Indicates whether a lookup based on the lookup relation is set up for the field. You can set up a lookup relation for this field in the Field Details screen. When set to Yes, the field will have the magnifying glass icon to open a pop-up window and select different options for the field value. (This field is read-only.)

Data Type

Select the data type: Character, Date, Datetime, Datetime-tz, Decimal, Drop Down, Integer, Integer (64-bit), Logical, Percent, and URL.

Length

For character fields only, specify the maximum character length.

Format

Indicate the data format based on the data type. The format is aligned with the OOABL field format. The system provides the default format, but you can change it to some other valid value.

Drop-Down List

The drop-down list that you can assign to the field. Then the field will have a drop-down list where you can select the possible values for the field values.

Default Value

Enter the default value for the field.

Formula

Indicate whether this field's value should be calculated based on a formula, rather than entered by the user and stored in the database. When this field is put on a view, this field will always be read-only at run-time.

Physical Field

The name of the column in the corresponding database table for business components created by importing from the database. (This field is read-only.)

Field Group

Displays the name of the field group to which this field is assigned. You can assign a field to a field group using the Field Groups grid. (This field is read-only, and is specified by using the field group grid panel - see below)

Minimum Value

Specifies the minimum value allowed for the field. This setting can be applied to fields of all data types except the Character and Logical data types.

Maximum Value

Specifies the maximum value allowed for the field. This setting can be applied to fields of all data types except the Character and Logical data types.

Required

Indicate whether this field is required for a valid set of data for the business component. On a form, required fields are indicated with the yellow bar along the left side of the field box.

Read Only

Specifies whether the field value is read-only or can be changed by the user.

Internal

Specifies whether the field will be displayed on the form.

User Defined

Indicates whether the field is user-defined.

This field is only used for the legacy implementations in which one or more fields in the physical table can be configured to hold data specifically for the customer's implementation. For these fields specific code is required on the back-end business logic implementation to fetch and update the fields properly. For platform data driven components this indication should always be "false", as the guideline is to use the embedded business components together with a 1-1 BC relation to bind it to the business component it extends.

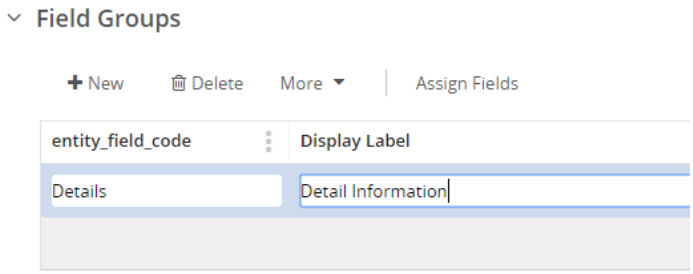
Deployed

Indicate whether this field is currently deployed. This is always read-only.

Discriminator

Indicates whether the field will be used as a "discriminator" field. A "discriminator" field is a sort of helper field used only for embedded BCs to provide the facility of DEO BCs extending several parent base BCs via 1-1 relation. This field must be a part of primary key and will not be visible on maintenance view. Under the hood the field will hold the value of entityUri of parent BC and that is why it should always have the datatype=Character.

Field Groups panel



New

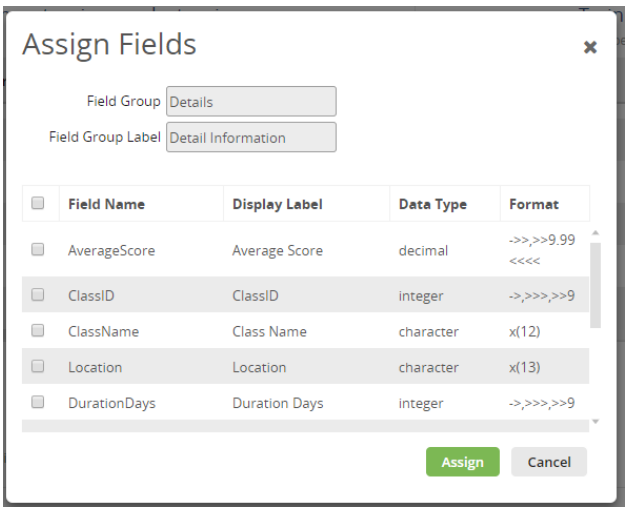
Create a new field group.

Delete

Delete a field group.

Assign Fields

Assign fields to the current field group.



Field Group Code

Enter the name for the field group.

Display Label

Enter a label for the field group.

Drop-Down Lists panel

New

Enter a name for a new drop-down list. You can then assign this drop-down list to the field.

Add Child

Add a child to a drop-down list and enter a value (database value) and label (name that will be displayed in drop-down options) for it. To add additional value and label to a drop-down list, click +New.

Delete

Delete an existing drop-down list.

More

Show Group By – Select a column header and group by that column.

Collapse All Rows – Collapse rows to see a more detailed view of the table.

Drop-Down Lists

Shows a list of drop-down lists available, which you can sort ascending, descending, or filter as needed.

Relationships panel

Use this grid to view and maintain the relationships of the business component. You can add new relationships only after the first save of the business component. (This grid is an "external" grid.) For more information about relationships, see Relationships.

New

Define a new BC relation. This opens the detail form view for entering further details.

Edit

Modify an existing BC relation. This opens the detail form view for entering further details.

Delete

Delete an existing BC relation.

Details

This opens the detail form view for entering further details.

Relationship

The relationship of this business component to the current business component.

Relationship Label

The label for the business component relationship.

Source Business Component

Specifies the business component on which a relation was created.

Related Business Component

The name of the business component the current business component is related to.

Relationship Type

Specifies the Parent or Child relationship type. For more information about relationship types, see Business Component Relation.

Cardinality

Specifies the cardinality of the relationship: 1-1, N-1, or 1-N.

This field is automatically populated based on the selection from the **Related Business Component** lookup, which is based on comparing primary keys of the Source and Related Business Components, which take part in this relationship. For example, if the amount and datatypes of primary key fields coincide, a system will propose 1-1 cardinality and fill in the Field Mapping grid with corresponding primary keys. Still, the developer can then modify the cardinality. If the developer modifies the cardinality, the Field Mapping grid is filled in automatically only on the 1-side of relationships.

Source App

Specifies the application to which the source business component belongs.

Source App URI

Specifies the unique identifier of the app where the source business component resides.

Related App

Specifies the app of the related business component.

Related App URI

Specifies the unique identifier of the related app.

Business Document panel

The Business Document panel appears only if it is activated on the Options panel under the Main panel of a business component. The business document structure is automatically generated by looking at the existing relationships for the current business component, and it can be viewed via a hierarchical grid.

To create a business document:

- Create a relationship under Relationships. The business document only considers a parent-child relationship defined on a parent side. If a relationship is defined on a child side, it is not considered in a business document.
- Enable a business document under Main > Options panel.

Note: Excel integration (import and export) is available for all business components (except extended business components) and business documents.

Business Document

Enter the name for this business document. By default, this field displays the value from the Business Component field. The field is enabled until the business document is released as part of an app package.

Business Document Label

Enter the label for this business document. By default, this field displays the value from the Business Component Label field.

Business Document URI

The URI of the business document associated with this business component. It reflects changes to the Business Document field when created.

Form panel

When you create a form, it is always saved with the same app as the business component app.

Existing Form

Indicates whether a form already exists for this view.

Build Form (or Edit Form)

Click to start building a new form or to edit an existing form. Click **Build Form** or **Edit Form** to open the Build Form pop-up window. Note: The Build Form button is disabled until the business component is saved for the first time.

Event Handlers panel

Edit event handlers for the form. You can specify whether the event handlers are active and the timing as Pre (run before any other event handlers) or Post (run after any other event handlers).

Views panel

The Views panel lists all the views created for the BC. Click New to define a new view or click Details to view and edit the details of an existing view. When you click New or Details, the Form Builder - Views pop-up window opens.

View

The display label for the view.

Type

Indicates the view type: Hybrid Browse (includes a browse and a form) or Form Only (only includes a form).

Eligible for Menu

Indicates whether this view will be available on the menu (and menu search).

App

The name of the app associated with the view.

App URI

The URI of the app associated with the view.

View URI

The URI of the view.

Secure URI

The URI for security and menu linking.

Source File Generation panel

Click **Download** to generate the business component source code files and download them in a .zip file.

Deployment panel

Deployment of a business component is a process of verifying if everything is available and correctly defined for the business component, this includes having available view and view resources available.
This will fail if the view is not yet defined. Deployment of a business component includes creation of the right table in the database associated with the selected data store, or the update of the table in case new fields have been added since the initial creation of the table.

Data Store URI

The URI of the data store that stores the schema and the data of the business component. This is a lookup to select a data store to which the current is going to be deployed. *Important:* After the business component is deployed for the first time, this field becomes read-only: the Data Store URI cannot be changed.

Import Data

Select to import data from the data source from which the business component definition has been loaded. This field becomes enabled only if business component fields were created using a file upload. If this checkbox is selected, then during the deploying of the business component, the data from the Excel file is copied to the corresponding database table.

Deploy

Click **Deploy** to deploy the business component. Once a business component has been deployed, a user interface view can be built for it using the Form Builder.

Relationships (Business Components)

Introduction

In the Business Components view's Relationships panel, a developer can view and manage (create, update, and delete) the relationships of the current business component to other business components.

What is a Relationship between Business Components?

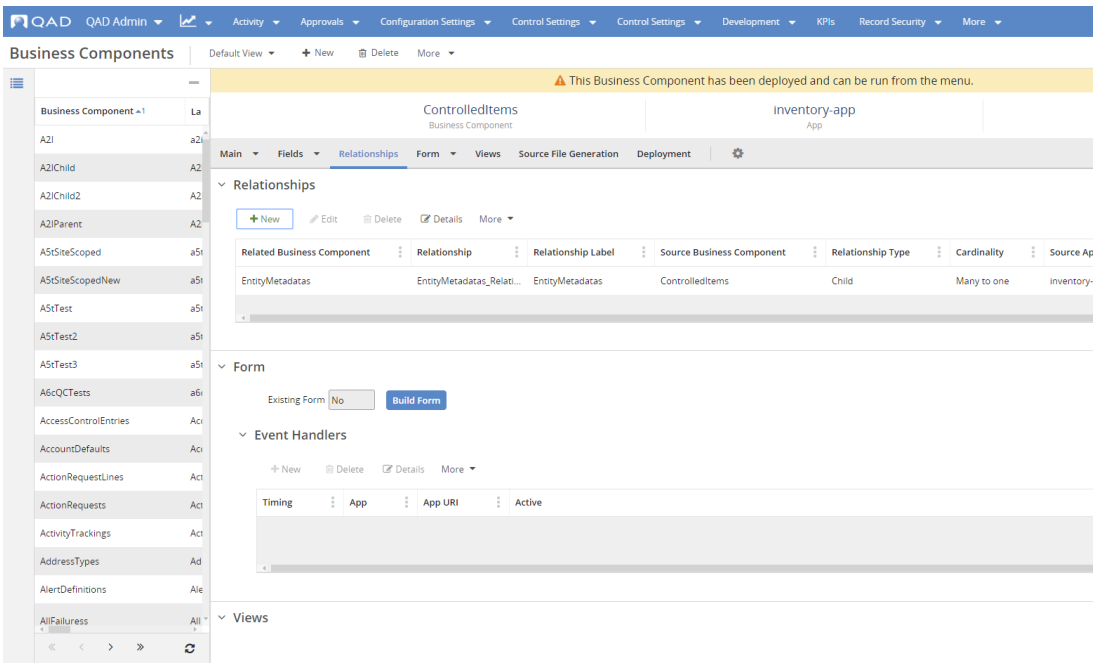
A relationship between business components (BCs) is like the relationship between tables in relational databases.

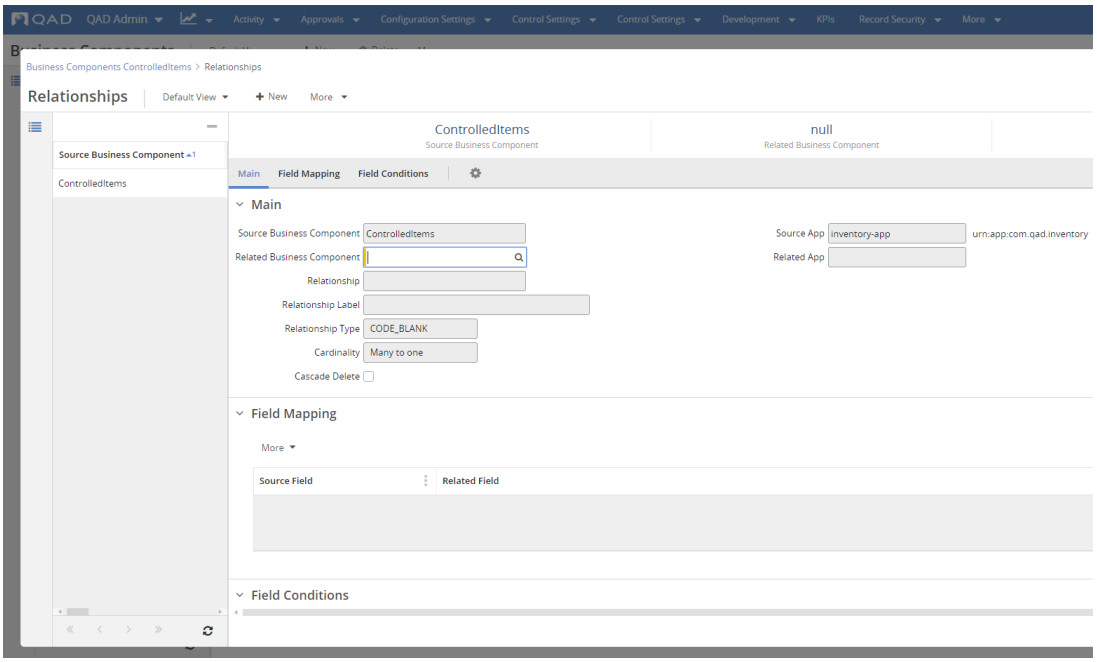
The relationship between BCs can have cardinality: 1-1 (one-to-one) , N-1 (many-to-one), and 1-N (one-to-many). However, many-to-many relationships between BCs are not supported.

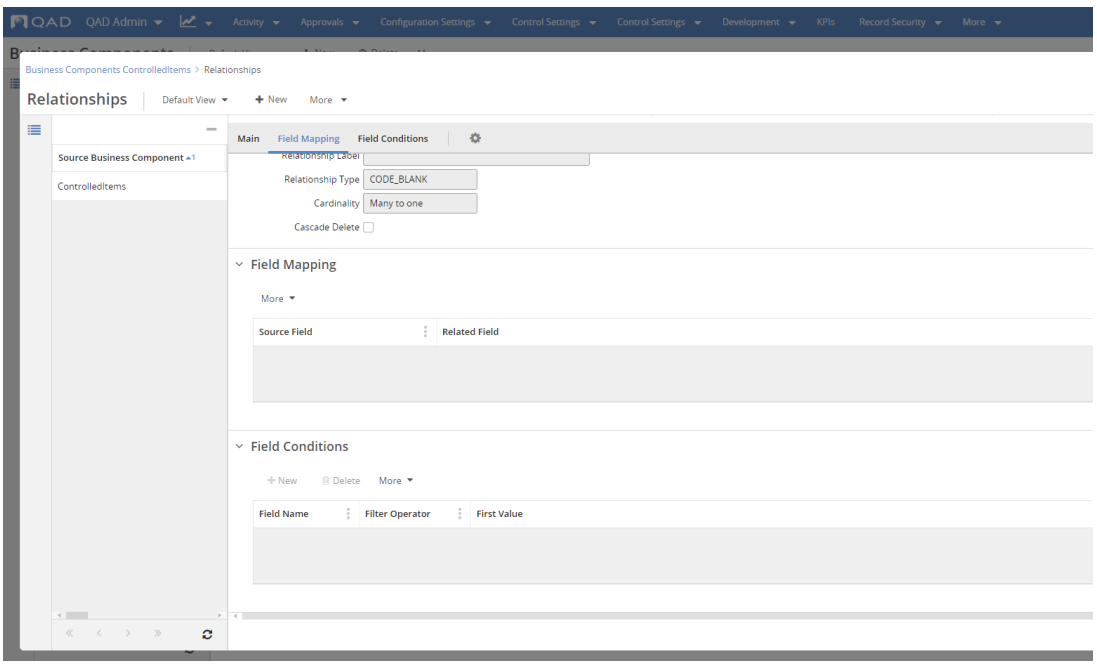
As with relationships between tables in relational databases, business component relationships are based on primary keys. To be able to create 1-1 relationships between two BCs, they must have the same number and type of fields for their primary keys. For 1-N or N-1 relationships, the "N"-side should have the fields which can be selected as foreign keys and which will have in turn the same type as the primary keys of the BC on the "1"-side.

Screen Shots

Can be opened using the New button on the Relationships panel in Business Components.







Creating, Editing, and Deleting

You can create relationships between any pair of business components (coded BC, BC created by Business Component Builder and deployed, BC created by Component Builder and not deployed, embedded BC) as long as their primary keys (or foreign keys) have the same number of fields and the same field data types. When you create some relationship using one source business component, this relationship will also appear in the Relationships panel of the related business component. Extension relationships cannot be edited; they can only be deleted and then recreated again. If the last Extension relationship for some deployed embedded business component is deleted, this embedded BC is then un-deployed.

UI Quick Reference

Main panel

Source Business Component

Specifies the business component on which a relation was created.

Source App

Specifies the application to which the source business component belongs.

Related Business Component

In the lookup, select the business component the current business component is related to, and then click OK.

Related App

Specifies the app of the related business component.

Relationship

Displays the code of relationship. This field is automatically populated based on the selection from the **Related Business Component** lookup, but can be changed.

Relationship Label

Displays the relationship label. This field is automatically populated based on the selection from the **Related Business Component** lookup, but can be changed.

Relationship Type

Select the Parent or Child relationship type. For more information about relationship types, see Business Component Relation.

Cardinality

Specifies the cardinality of the relationship: 1-1, N-1, or 1-N.

This field is automatically populated based on the selection from the **Related Business Component** lookup, which is based on comparing primary keys of the Source and Related Business Components, which take part in this relationship. For example, if the amount and datatypes of primary key fields coincide, a system will propose 1-1 cardinality and fill in the Field Mapping grid with corresponding primary keys. Still, the developer can then modify the cardinality. If the developer modifies the cardinality, the Field Mapping grid is filled in automatically only on the 1-side of relationships.

Cascade Delete

When this checkbox is selected, it means that if you delete the parent business component, its child components will also be deleted.

Note: If the relationship is a Lookup Relationship, the Cascade Delete checkbox is hidden.

Field Mapping panel

The grid where the mapping between fields of two BCs participating in a relationship is displayed. The system automatically fills in the fields for 1-side of the relationship and the developer cannot edit this column in the grid. The fields from the N-side must be specified by the developer. The system provides lookups to select fields of the same type as the one of the corresponding primary key field.

Field Conditions panel

This grid allows the developer to include additional conditions on fields of a related BC. These conditions can then be seen on Drill Down browses based on 1-N relationships as filter conditions.

Formula (Business Components - Fields panel)

Introduction

What is a Formula Field of a Business Component

Formula Fields are a separate type of Business Component (BC) fields whose values are calculated according to formula definition. Formula fields do not have an appropriate physical column in a database. These fields may be added to the Form and to the BC Browse using the View Builder tool as usual fields and will always show up as read-only at runtime.

Formula fields are only available for business components created and maintained in the business component builder (so, not supported for traditional coded business components).

Which Fields May Be Used in a Formula Definition

For the definition of a formula field in a BC, which is not an embedded BC, only the following fields can be used:

- fields from the BC table itself : tableName.fieldName
- fields from related DEO BC: relationName.fieldName

For the definition of a formula field in an embedded BC, only the following fields can be used:

- fields from the BC table itself : tableName.fieldName

Which Operators May Be Used in a Formula Definition

Following operators may be used in a formula definition:

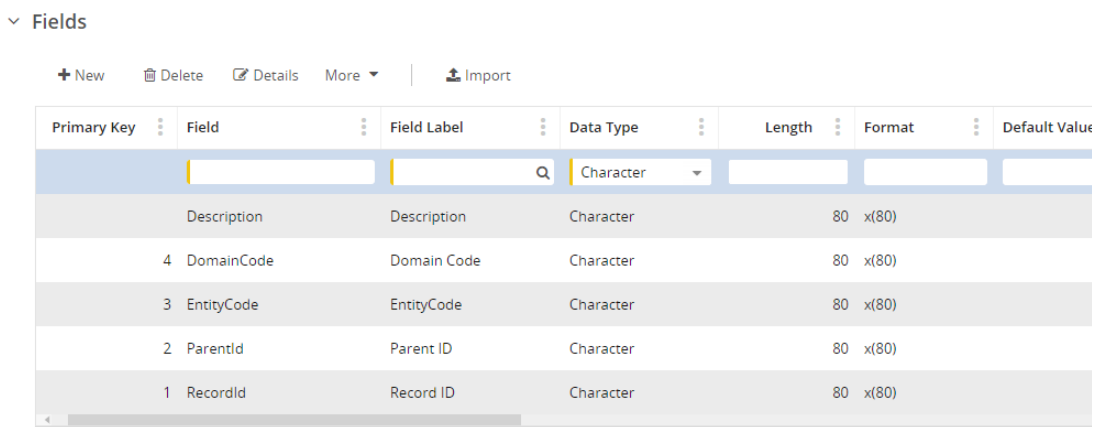
'ABS', 'ACCRINT', 'ACOS', 'ACOSH', 'ACOT', 'ACOTH', 'ADD', 'AGGREGATE', 'AND', 'ARABIC', 'ARGS2ARRAY', 'ASIN', 'ASINH', 'ATAN', 'ATAN2', 'ATANH', 'AVEDEV', 'AVERAGE', 'AVERAGEA', 'AVERAGEIF', 'AVERAGEIFS', 'BASE', 'BESSELI', 'BESSELJ', 'BESSELK', 'BESSELY', 'BETA.DIST', 'BETA.INV', 'BETADIST', 'BETAINV', 'BIN2DEC', 'BIN2HEX', 'BIN2OCT', 'BINOM.DIST', 'BINOM.DIST.RANGE', 'BINOM.INV', 'BINOMDIST', 'BITAND', 'BITLSHIFT', 'BITOR', 'BITRSHIFT', 'BITXOR', 'CEILING', 'CEILINGMATH', 'CEILINGPRECISE', 'CHAR', 'CHISQ.DIST', 'CHISQ.DIST.RT', 'CHISQ.INV', 'CHISQ.INV.RT', 'CHOOSE', 'CHOOSE', 'CLEAN', 'CODE', 'COLUMN', 'COLUMNS', 'COMBIN', 'COMBINA', 'COMPLEX', 'CONCATENATE', 'CONFIDENCE', 'CONFIDENCE.NORM', 'CONFIDENCE.T', 'CONVERT', 'CORREL', 'COS', 'COSH', 'COT', 'COTH', 'COUNT', 'COUNTA', 'COUNTBLANK', 'COUNTIF', 'COUNTIFS', 'COUNTIN', 'COUNTUNIQUE', 'COVARIANCE.P', 'COVARIANCE.S', 'CSC', 'CSCH', 'CUMIPMT', 'CUMPRINC', 'DATE', 'DATEVALUE', 'DAY', 'DAYS', 'DAYS360', 'DB', 'DDB', 'DEC2BIN', 'DEC2HEX', 'DEC2OCT', 'DECIMAL', 'DEGREES', 'DELTA', 'DEVSQ', 'DIVIDE', 'DOLLARDE', 'DOLLARFR', 'E', 'EDATE', 'EFFECT', 'EOMONTH', 'EQ', 'ERF', 'ERFC', 'EVEN', 'EXACT', 'EXP', 'EXPON.DIST', 'EXPONDIST', 'F.DIST', 'F.DIST.RT', 'F.INV', 'F.INV.RT', 'FACT', 'FACTDOUBLE', 'FALSE', 'FDIST', 'FDISTRT', 'FIND', 'FINV', 'FINVRT', 'FISHER', 'FISHERINV', 'FLATTEN', 'FLOOR', 'FORECAST', 'FREQUENCY', 'FV', 'FVSCHEDULE', 'GAMMA', 'GAMMA.DIST', 'GAMMA.INV', 'GAMMADIST', 'GAMMAINV', 'GAMMALN', 'GAMMALN.PRECISE', 'GAUSS', 'GCD', 'GEOMEAN', 'GESTEP', 'GROWTH', 'GTE', 'HARMEAN', 'HEX2BIN', 'HEX2DEC', 'HEX2OCT', 'HOUR', 'HTML2TEXT', 'HYPGEOM.DIST', 'HYPGEOMDIST', 'IF', 'IMABS', 'IMAGINARY', 'IMARGUMENT', 'IMCONJUGATE', 'IMCOS', 'IMCOSH', 'IMCOT', 'IMCSC', 'IMCSCH', 'IMDIV', 'IMEXP', 'IMLN', 'IMLOG10', 'IMLOG2', 'IMPOWER', 'IMPRODUCT', 'IMREAL', 'IMSEC', 'IMSECH', 'IMSIN', 'IMSINH', 'IMSQRT', 'IMSUB', 'IMSUM', 'IMTAN', 'INT', 'INTERCEPT', 'INTERVAL', 'IPMT', 'IRR', 'ISBINARY', 'ISBLANK', 'ISEVEN', 'ISLOGICAL', 'ISNONTEXT', 'ISNUMBER', 'ISODD', 'ISODD', 'ISOWEEKNUM', 'ISPMT', 'ISTEXT', 'JOIN', 'KURT', 'LARGE', 'LCM', 'LEFT', 'LEN', 'LINEST', 'LN', 'LOG', 'LOG10', 'LOGEST', 'LOGNORM.DIST', 'LOGNORM.INV', 'LOGNORMDIST', 'LOGNORMINV', 'LOWER', 'LT', 'LTE', 'MATCH', 'MAX', 'MAXA', 'MEDIAN', 'MID', 'MIN', 'MINA', 'MINUS', 'MINUTE', 'MIRR', 'MOD', 'MODE.MULT', 'MODE.SNGL', 'MODEMULT', 'MODESNGL', 'MONTH', 'MROUND', 'MULTINOMIAL', 'MULTIPLY', 'NE', 'NEGBINOM.DIST', 'NEGBINOMDIST', 'NETWORKDAYS', 'NOMINAL', 'NORM.DIST', 'NORM.INV', 'NORM.S.DIST', 'NORM.S.INV', 'NORMDIST', 'NORMINV', 'NORMSDIST', 'NORMSINV', 'NOT', 'NOW', 'NPER', 'NPV', 'NUMBERS', 'OCT2BIN', 'OCT2DEC', 'OCT2HEX', 'ODD', 'OR', 'PDURATION', 'PEARSON', 'PERCENTILEEXC', 'PERCENTILEINC', 'PERCENTRANKEXC', 'PERCENTRANKINC', 'PERMUT', 'PERMUTATIONA', 'PHI', 'PI', 'PMT', 'POISSON.DIST', 'POISSONDIST', 'POW', 'POWER', 'PPMT', 'PROB', 'PRODUCT', 'PROPER', 'PV', 'QUARTILE.EXC', 'QUARTILE.INC', 'QUARTILEEXC', 'QUARTILEINC', 'QUOTIENT', 'RADIANS', 'RAND', 'RANDBETWEEN', 'RANK.AVG', 'RANK.EQ', 'RANKAVG', 'RANKEQ', 'RATE', 'REFERENCE', 'REGEXEXTRACT', 'REGEXMATCH', 'REGEXREPLACE', 'REPLACE', 'REPT', 'RIGHT', 'ROMAN', 'ROUND', 'ROUNDDOWN', 'ROUNDUP', 'ROW', 'ROWS', 'RRI', 'RSQ', 'SEARCH', 'SEC', 'SECH', 'SECOND', 'SERIESSUM', 'SIGN', 'SIN', 'SINH', 'SKEW', 'SKEW.P', 'SKEWP', 'SLN', 'SLOPE', 'SMALL', 'SPLIT', 'SPLIT', 'SQRT', 'SQRTPI', 'STANDARDIZE', 'STDEV.P', 'STDEV.S', 'STDEVA', 'STDEVP', 'STDEVPA', 'STDEVS', 'STEYX', 'SUBSTITUTE', 'SUBTOTAL', 'SUM', 'SUMIF', 'SUMIFS', 'SUMPRODUCT', 'SUMSQ', 'SUMX2MY2', 'SUMX2PY2', 'SUMXMY2', 'SWITCH', 'SYD', 'T', 'T.DIST', 'T.DIST.2T', 'T.DIST.RT', 'T.INV', 'T.INV.2T', 'TAN', 'TANH', 'TBILLEQ', 'TBILLPRICE', 'TBILLYIELD', 'TDIST', 'TDIST2T', 'TDISTRT', 'TIME', 'TIMEVALUE', 'TINV', 'TINV2T', 'TODAY', 'TRANSPOSE', 'TREND', 'TRIM', 'TRIMMEAN', 'TRUE', 'TRUNC', 'UNICHAR', 'UNICODE', 'UNIQUE', 'UPPER', 'VAR.P', 'VAR.S', 'VARA', 'VARP', 'VARPA', 'VARS', 'WEEKDAY', 'WEEKNUM', 'WEIBULL.DIST', 'WEIBULLDIST', 'WORKDAY', 'XIRR', 'XNPV', 'XOR', 'YEAR', 'YEARFRAC'

It is possible to reference fields from tables in a child business component (this is a business component that has a 1-N parent-child BC relationship defined), but for these fields only the following operators are applicable:

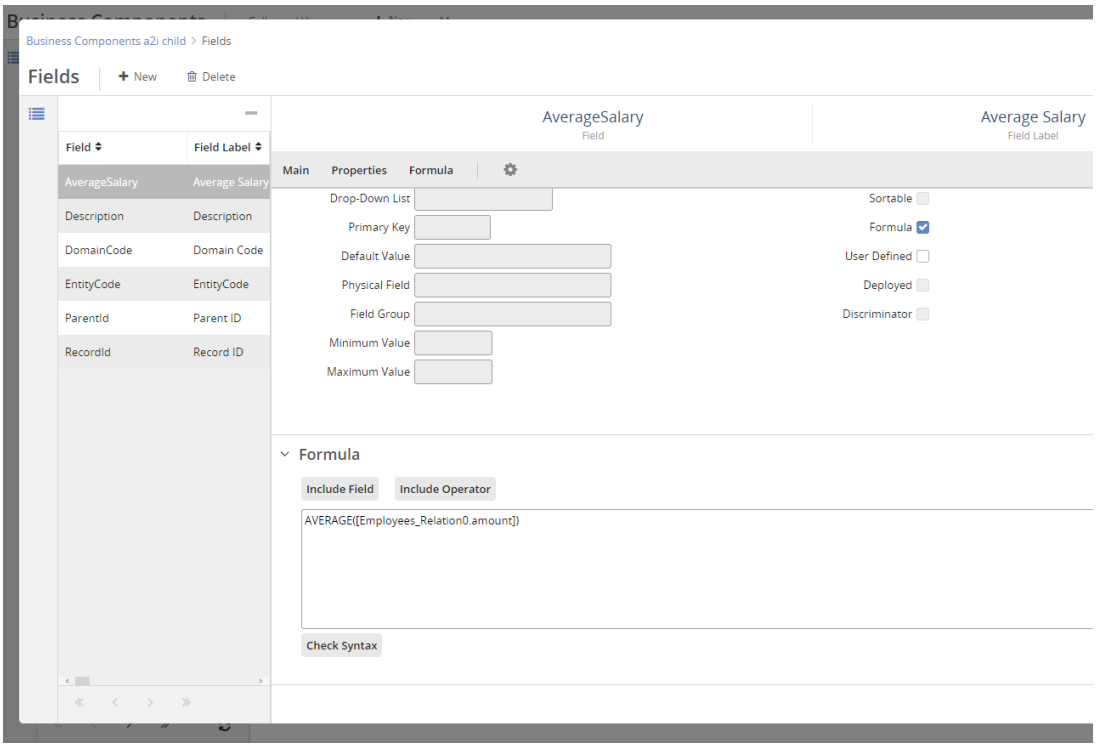
- SUM
- MIN
- MAX
- COUNT
- AVERAGE

Example Screenshots

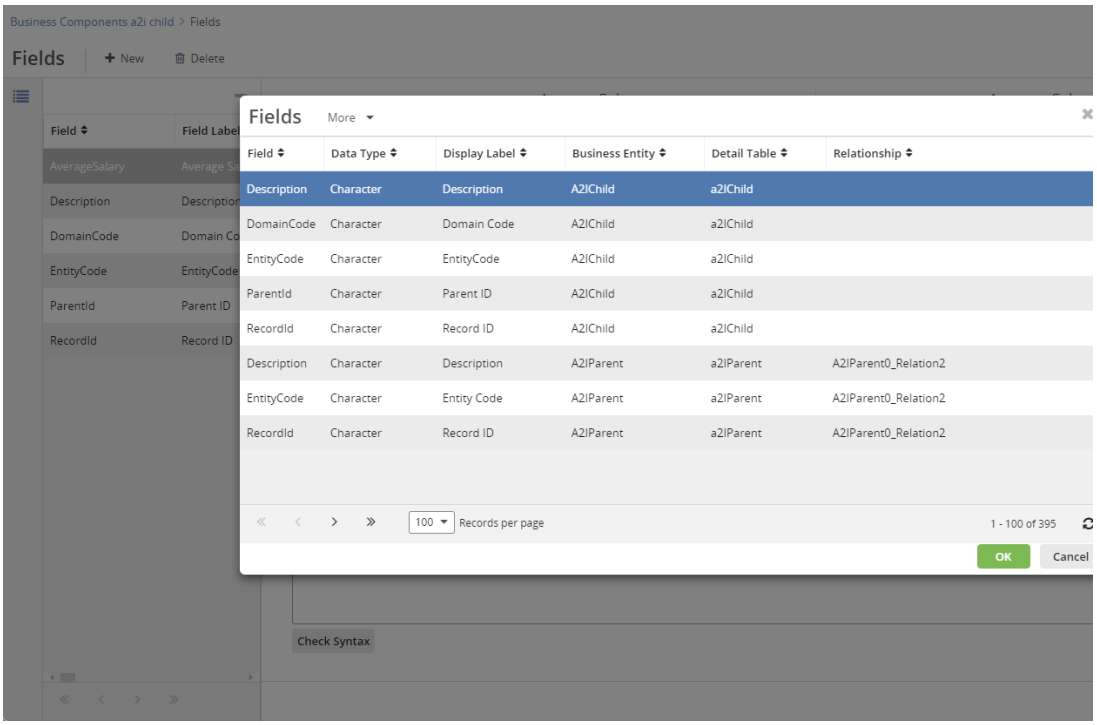
Business Components view > **Fields panel**



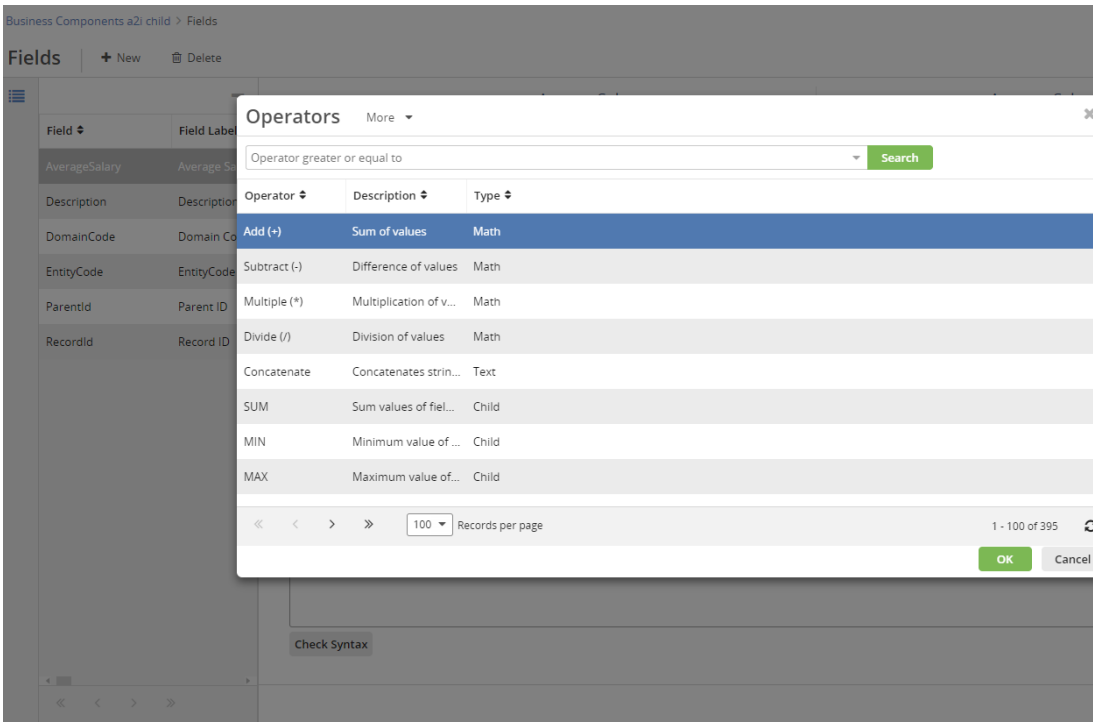
Business Components view > Fields panel > Details button > **Formula panel**



Business Components view > Fields panel > Details button > Formula panel > **Include Field**



Business Components view > Fields panel > Details button > Formula panel > **Include Operator**



UI Quick Reference

In the Business Components view, the grid in the Fields panel includes a Formula column that indicates whether the field is based on a formula.

Fields with Formula set to Yes can be edited by clicking the Details button, which opens the Fields pop-up window. Scroll down to the Formula panel. Note: If you cannot see the Formula panel, check if the Formula checkbox is selected on the P roperties panel.

Formula panel

Include Field

Click to open the Fields pop-up window, where you can choose the fields that will be available for inclusion. You can then include these fields directly in the formula definition area.

Include Operator

Click to open the Operators pop-up window, where you can choose the operators available for use in the formula definition. You can place the operators directly at the current cursor location in the formula definition area.

Check Syntax

Click to validate the syntax of the formula.

Fields pop-up (Include Field)

Field

The name of the formula field.

Display Label

The display label of the formula field.

Additionally, the Data Type, Business Component, Detail Table, and Relationship columns are displayed for unique identification of the field.

Operators pop-up (Include Operator)

Operator

The name of the operator.

Description

A short description of the operator.

Type

The type of the operator, such as Text, Math, Child, or Excel.

Creating Approvals - Step by Step

Table of Contents

Steps

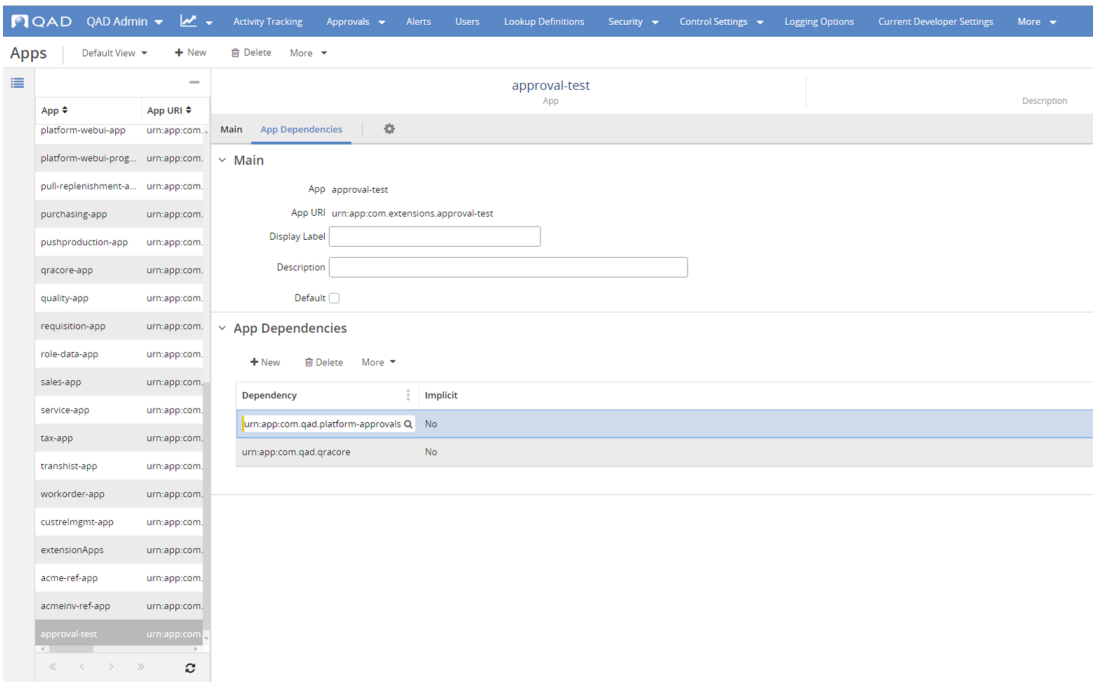
The steps for creating an approval flow for a virtual business component are as follows.

Step 1: Create a new app

Since we will be customizing **platform-approvals-app**, let's create a new app named "approval-test". If you already have an app you can use, skip this step.

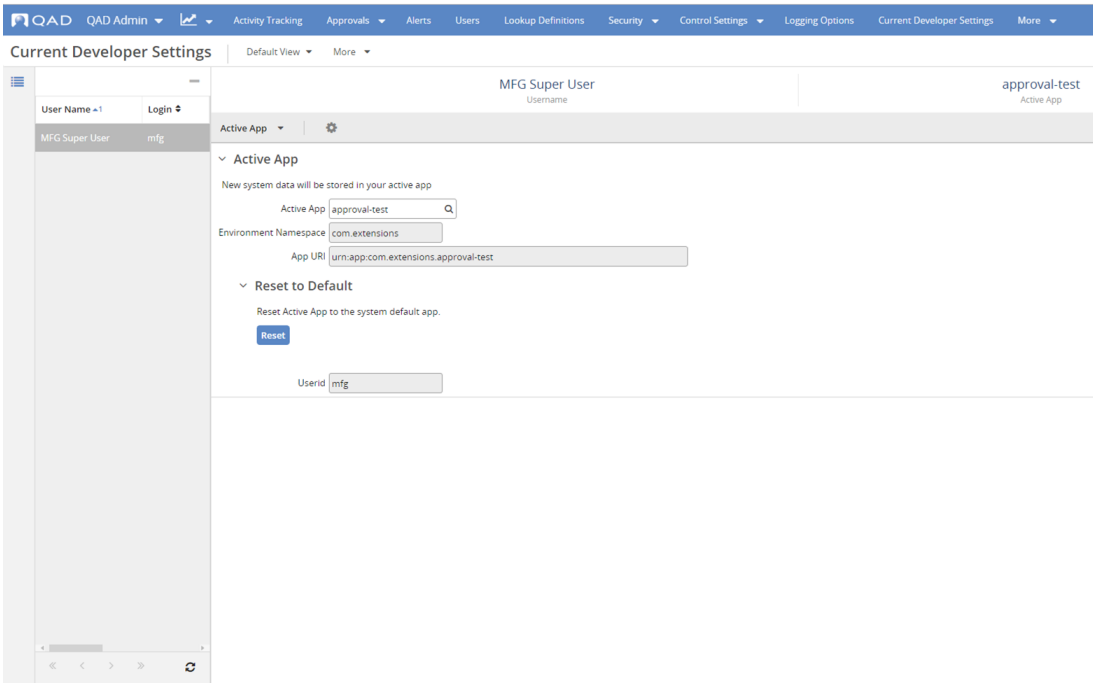
Our new app should have a dependency on the standard app **platform-approvals-app** and on qracore. Note that the dependency on qracore will automatically be added when you save the app with the other dependencies in place.

1. Launch Apps from the menu, and then click the New toolbar button.
2. Fill in the necessary fields on the Main panel.
3. On the App Dependencies panel, manually add the app dependency "urn:app:com.qad.platform-approvals". Ensure that the app dependency "urn:app:com.qad.qracore" is added automatically.
4. Save.



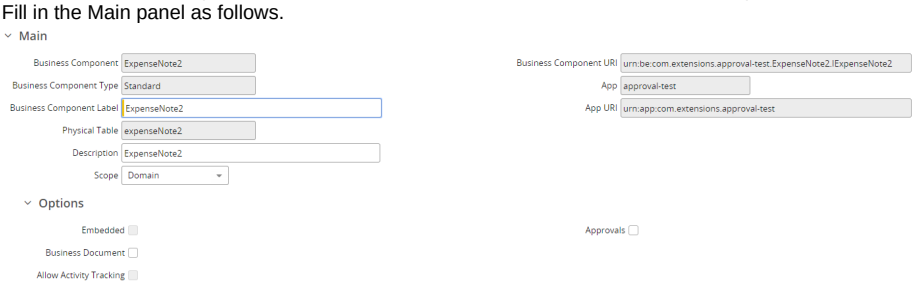
Step 2: Set the new app as Active for yourself

1. Launch Current Developer Settings from the menu, and then select the app created in Step 1.
2. Save.

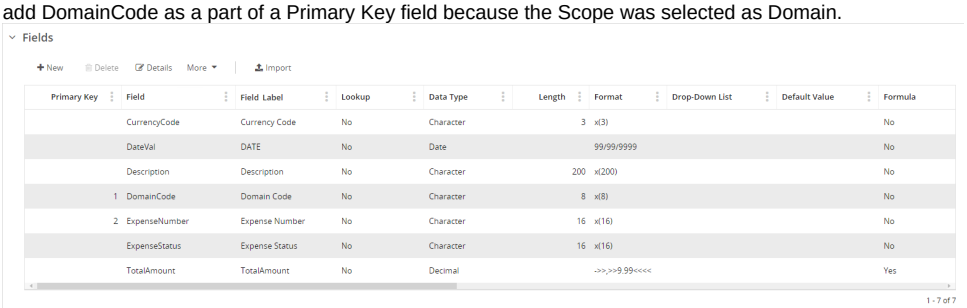


Step 3: Create and deploy a virtual business component

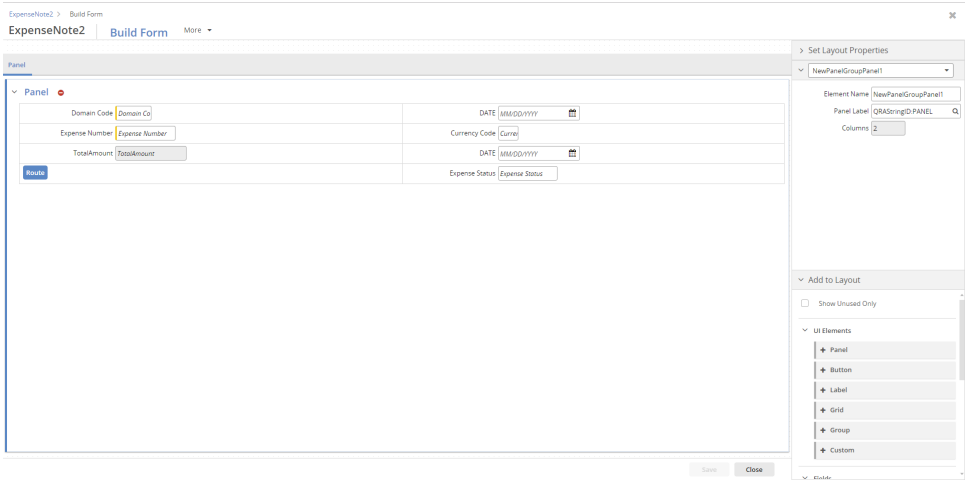
1. Launch Business Components from the menu, and then create a new business component.
2. Fill in the Main panel as follows.



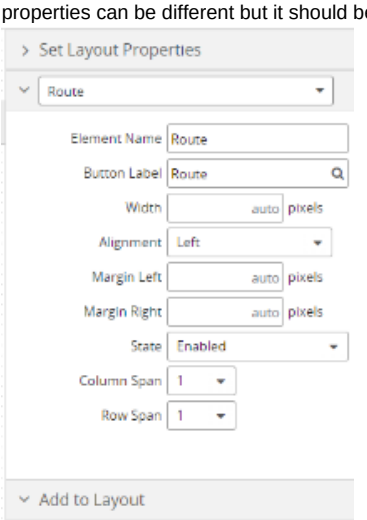
3. On the Fields panel, fill in the Fields as listed on the screen shot below. There should be a field in the BC to represent the Status, like in this example the ExpenseStatus field. Note that you also need to add DomainCode as a part of a Primary Key field because the Scope was selected as Domain.



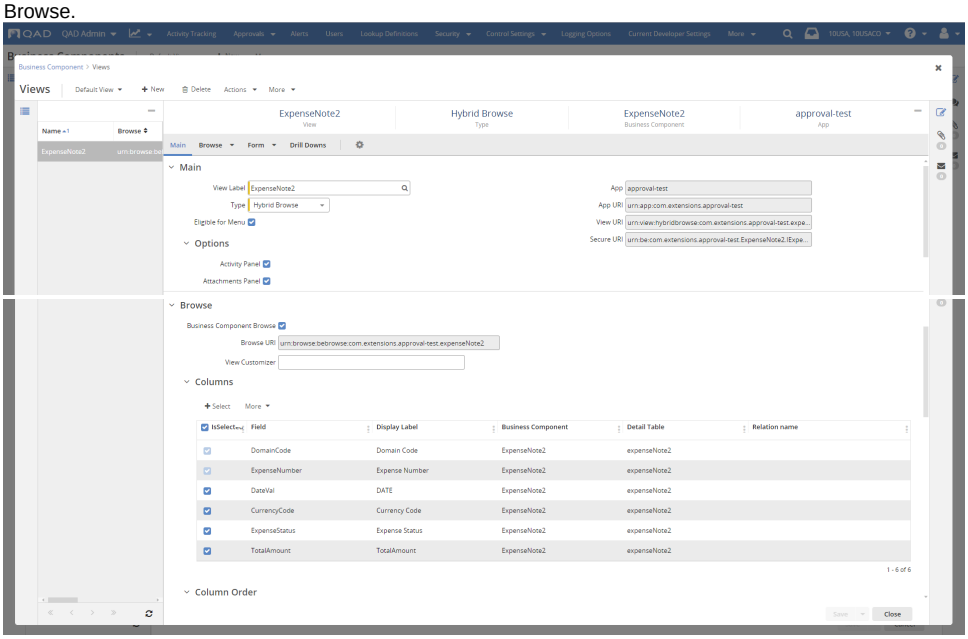
4. Save.
5. Navigate to the Form panel, and then click the Build Form button. In the Build Form pop-up window, compose a Form: drag-and-drop a Panel, all fields, and a button named "Route".



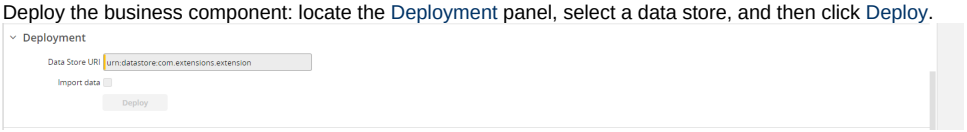
Note: The Route button should have the following properties: Element Name and Button Label as "Route". The properties can be different but it should be considered in the Event Handler that handles this button.



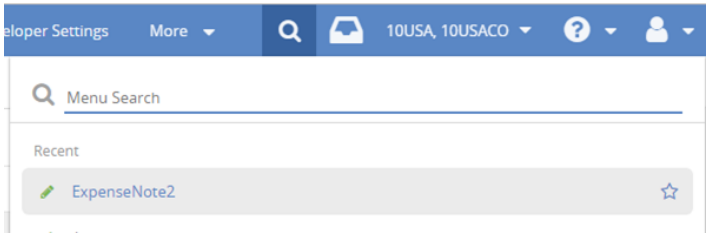
6. Navigate to the Views panel, and then click the New grid button. In the Views pop-up window, compose a Hybrid Browse.



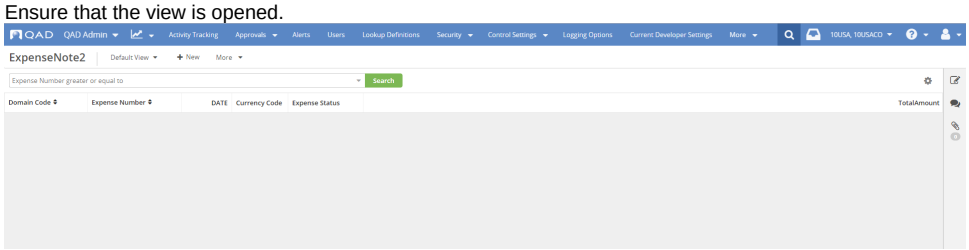
7. Deploy the business component: locate the Deployment panel, select a data store, and then click Deploy.



8. Open the business component in the runtime from the menu search.



9. Ensure that the view is opened.



Step 4: Create an event handler for the Route button

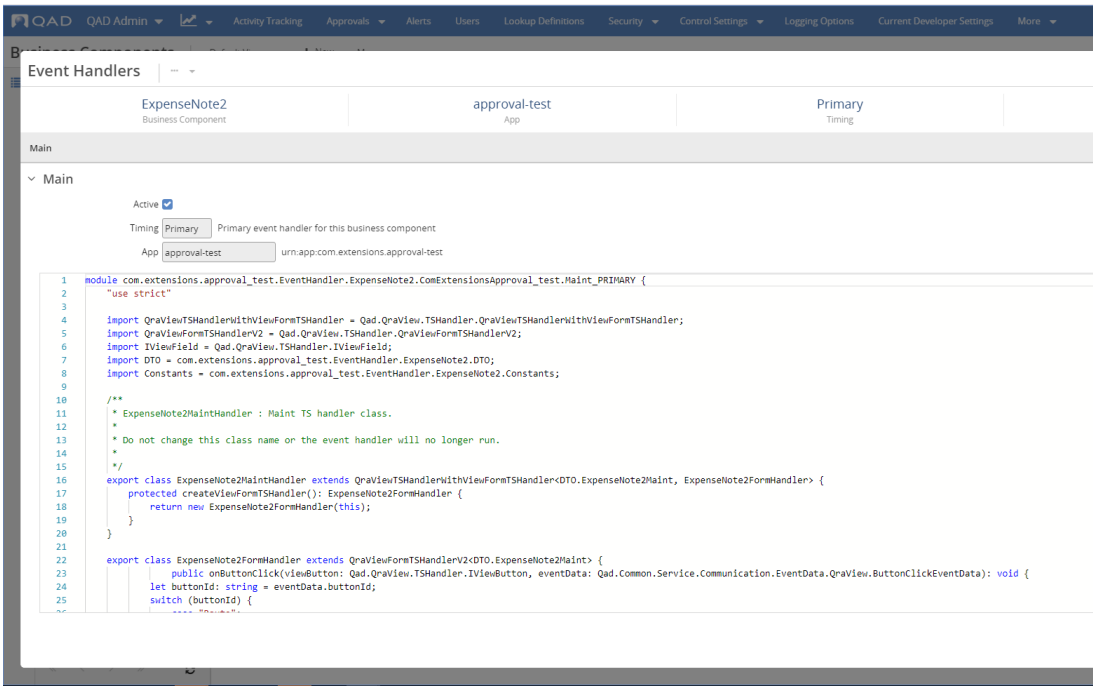
For managing the Route button, create a TypeScript handler:

1. Navigate to Business Components > Form panel > Event Handlers panel, and then click the New grid button.
2. In the Event Handlers pop-up window, set the handler as "Active", "Primary".
3. Add a TypeScript code, click the Compile button, and then Save.

Typescript Code

```
module com.extensions.approval_test.EventHandler.ExpenseNote2.ComExtensionsApproval_test.
Maint_PRIMARY {
    "use strict"

    import QraViewTSHandlerWithViewFormTSHandler = Qad.QraView.TSHandler.
QraViewTSHandlerWithViewFormTSHandler;
    import QraViewFormTSHandlerV2 = Qad.QraView.TSHandler.QraViewFormTSHandlerV2;
    import IViewField = Qad.QraView.TSHandler.IViewField;
    import DTO = com.extensions.approval_test.EventHandler.ExpenseNote2.DTO;
    import Constants = com.extensions.approval_test.EventHandler.ExpenseNote2.Constants;

    /**
     * ExpenseNote2MaintHandler : Maint TS handler class.
     *
     * Do not change this class name or the event handler will no longer run.
     *
     */
    export class ExpenseNote2MaintHandler extends QraViewTSHandlerWithViewFormTSHandler<DTO.
ExpenseNote2Maint, ExpenseNote2FormHandler> {
        protected createViewFormTSHandler(): ExpenseNote2FormHandler {
            return new ExpenseNote2FormHandler(this);
        }
    }

    export class ExpenseNote2FormHandler extends QraViewFormTSHandlerV2<DTO.ExpenseNote2Maint> {
                public onButtonClick(viewButton: Qad.QraView.TSHandler.IViewButton, eventData:
Qad.Common.Service.Communication.EventData.QraView.ButtonClickEventData): void {
            let buttonId: string = eventData.buttonId;
            switch (buttonId) {
                case "Route":
                    this.allowApproval(this.$scope.primaryEntityUri);
                    break;
                default:
                    break;
            }
        }
```

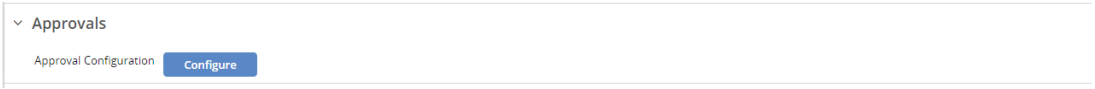
```
        allowApproval(beURI:string){
            if(beURI){
                        let httpData: string = JSON.stringify(this.NgData);
             let httpConfig: ng.IRequestShortcutConfig = {
                headers: { "Content-Type": "application/json; charset=UTF-8" },
                responseType: "application/json; charset=UTF-8",
            };
            let targetUrl = "api/approvals/startRouting/{beURI}";
            this.ViewController.doHttpPost(
                bindTemplateData(targetUrl, {beURI: beURI}),
                (data: Qad.Common.DTO.DataResult, status, headers, config) => {

                                },
                (data, status, headers, config) => {

                },
                null,
                httpData,
                httpConfig)
            }
        }

    }
}
```



Step 5: Create an approval for a business component

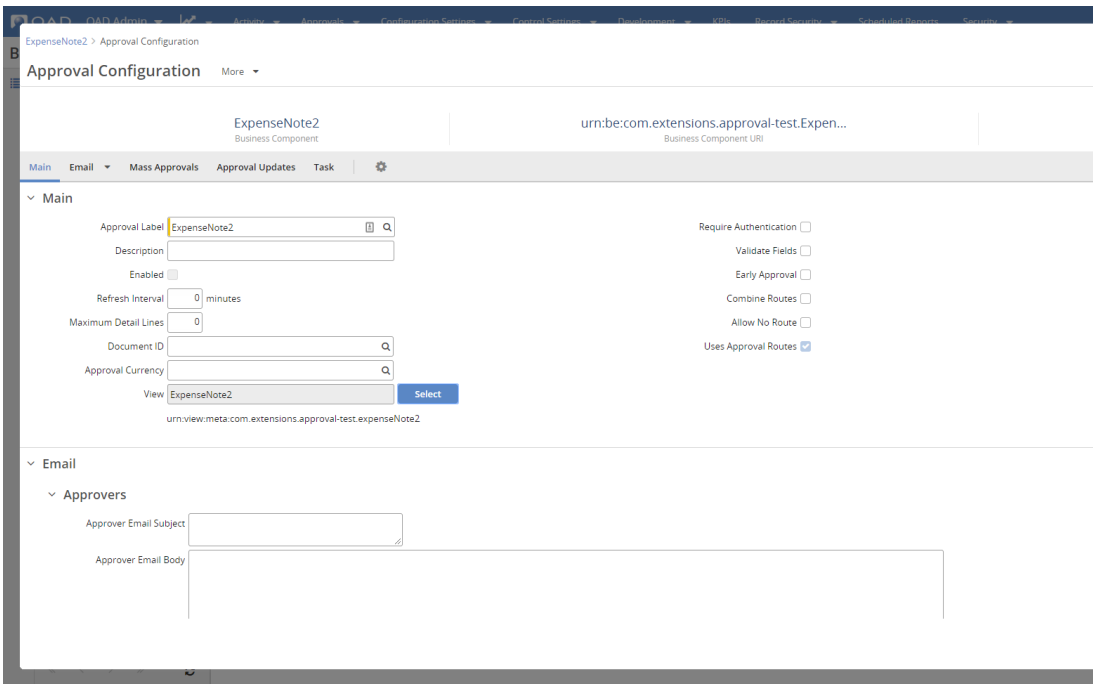
On the business component's Main panel, select the Approvals checkbox.

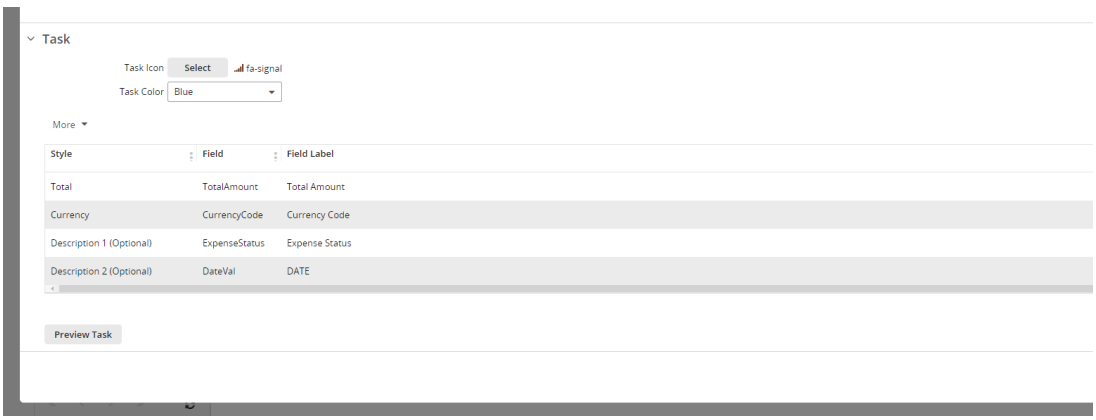
Now the additional Approvals panel appears at the bottom of the page. The Configure button allows you to configure approval settings for the current business entity.



Step 6: Approval configuration

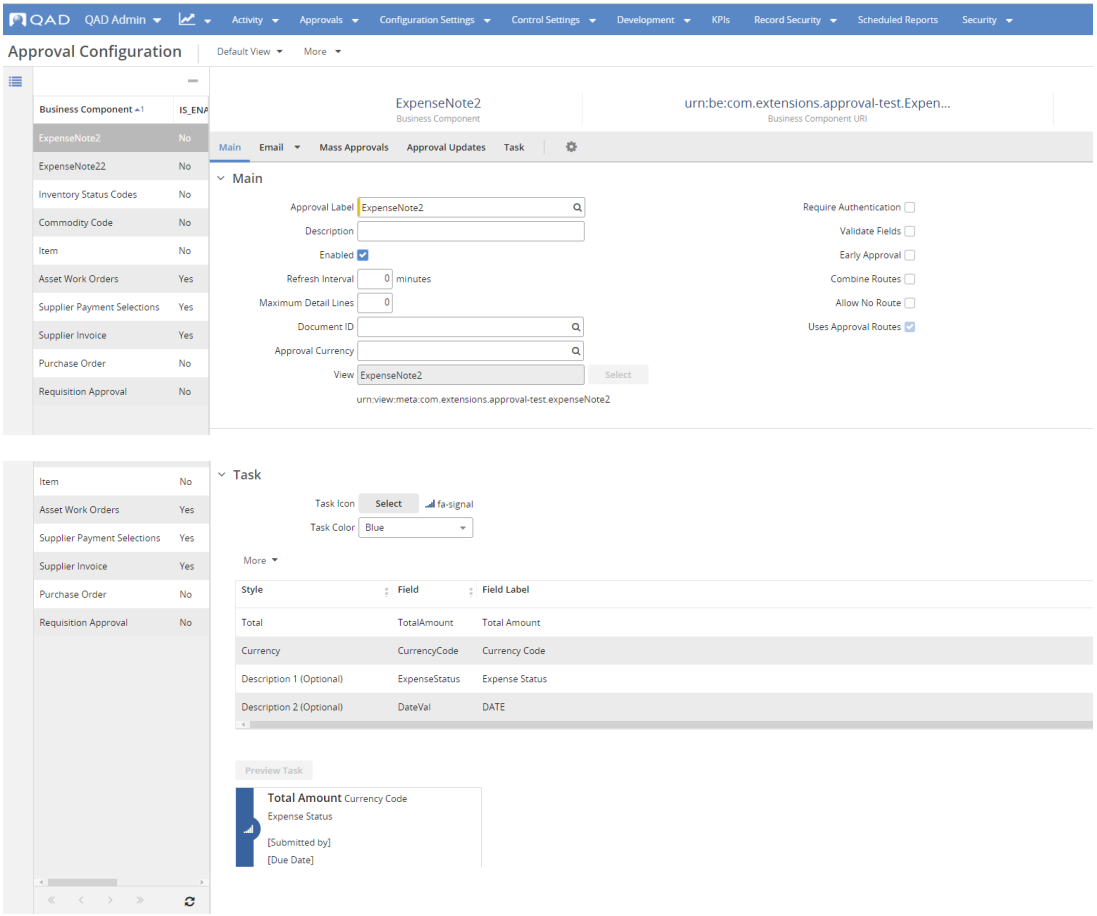
1. On the Approvals panel, click the Configure button, and then in the Approval Configuration pop-up window specify the configuration in all panels.
2. Save the configuration.





Step 7: Enable approval configuration

1. Navigate to Approval Configuration, and then search for the created business component.
2. On the Main panel, select the Enabled checkbox.
3. Save.

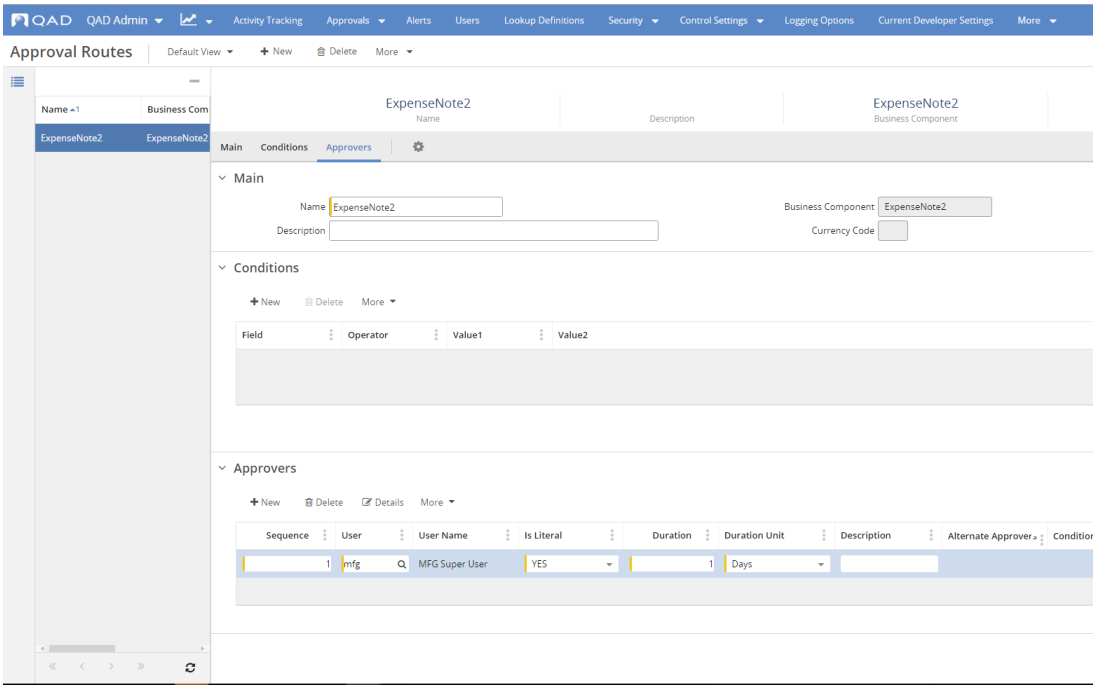


The Preview Task button allows you to preview the task according to the selected properties.

Note: As an option, approval configuration can be edited and saved directly in the current screen.

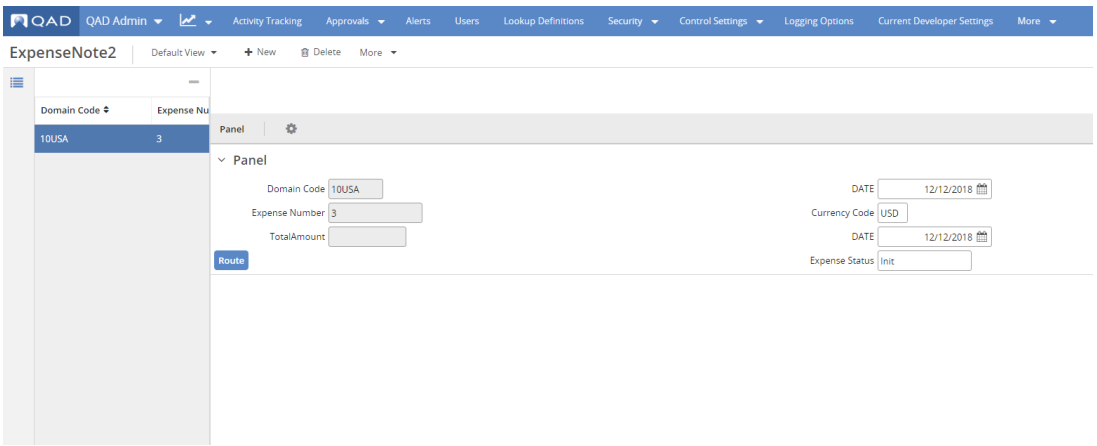
Step 8: Create approval routes

1. Navigate to Approval Routes, and then click the New toolbar button.
2. On the Main panel, specify the route's Name and select the current business component.
3. On the Approvers panel, choose a User to approve a task.
4. Save.



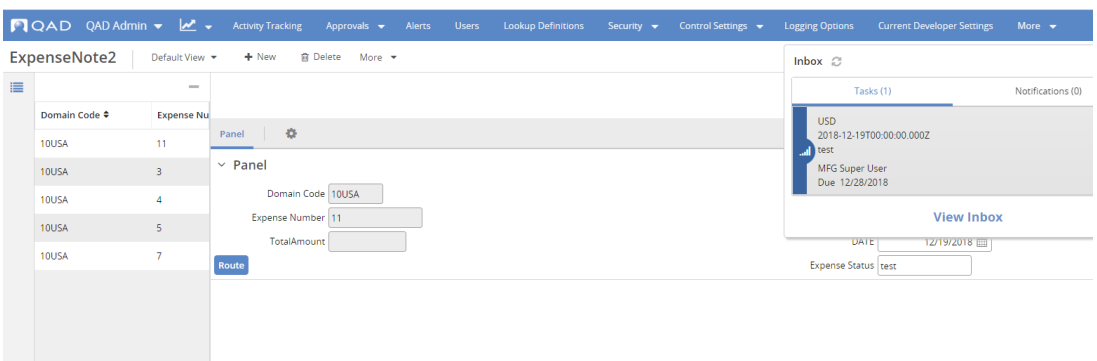
Step 9: Route action to an approver

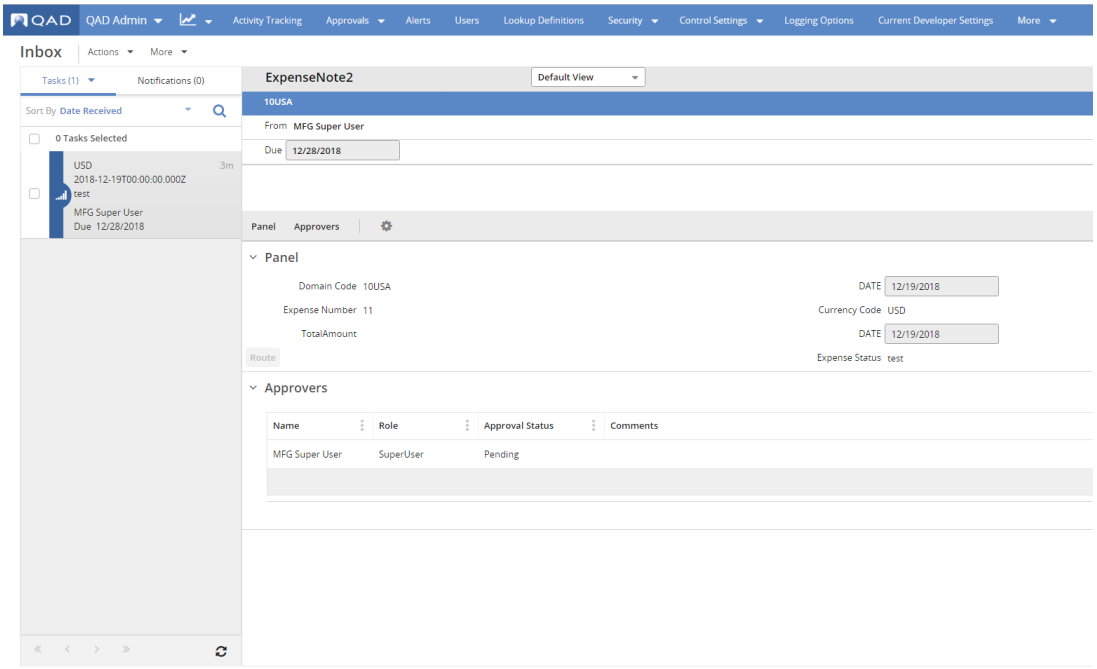
1. Open the business component in the runtime from the menu search, and then click the New toolbar button.
2. Create and save a new record.
3. Click the Route button. This will create a new task for an approver. (It can take some time, from seconds to minutes).



Step 10: Approval process

Approvers will receive a new task in their Inbox. Being logged in, they can click View Inbox and review the created task.

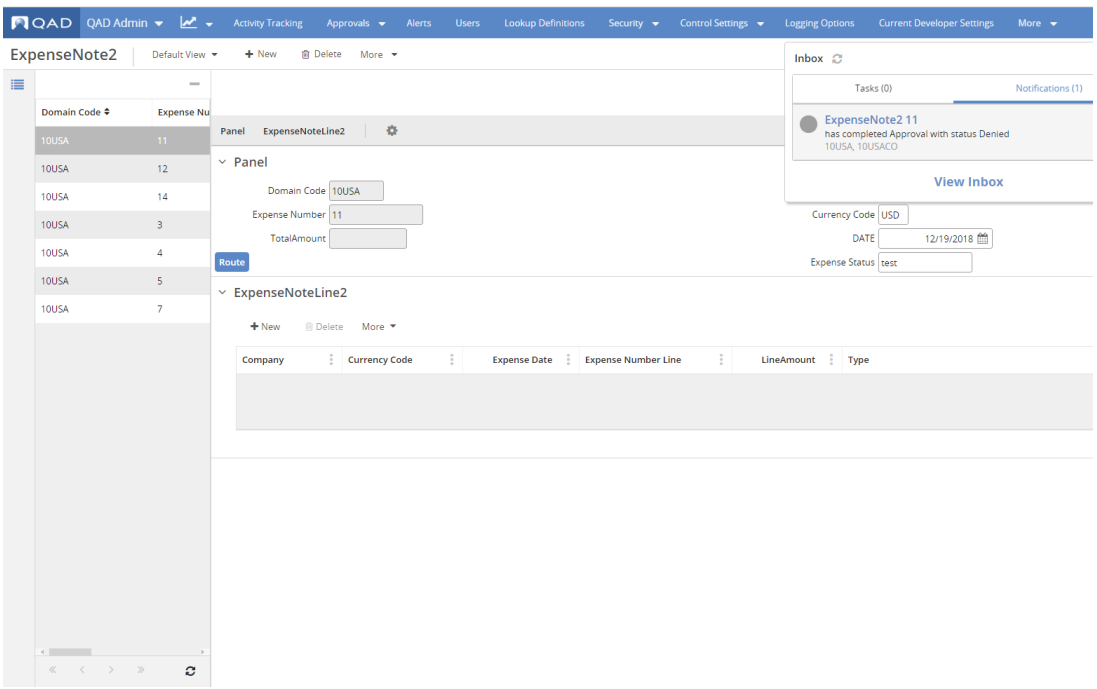




Approvers can approve or deny the task

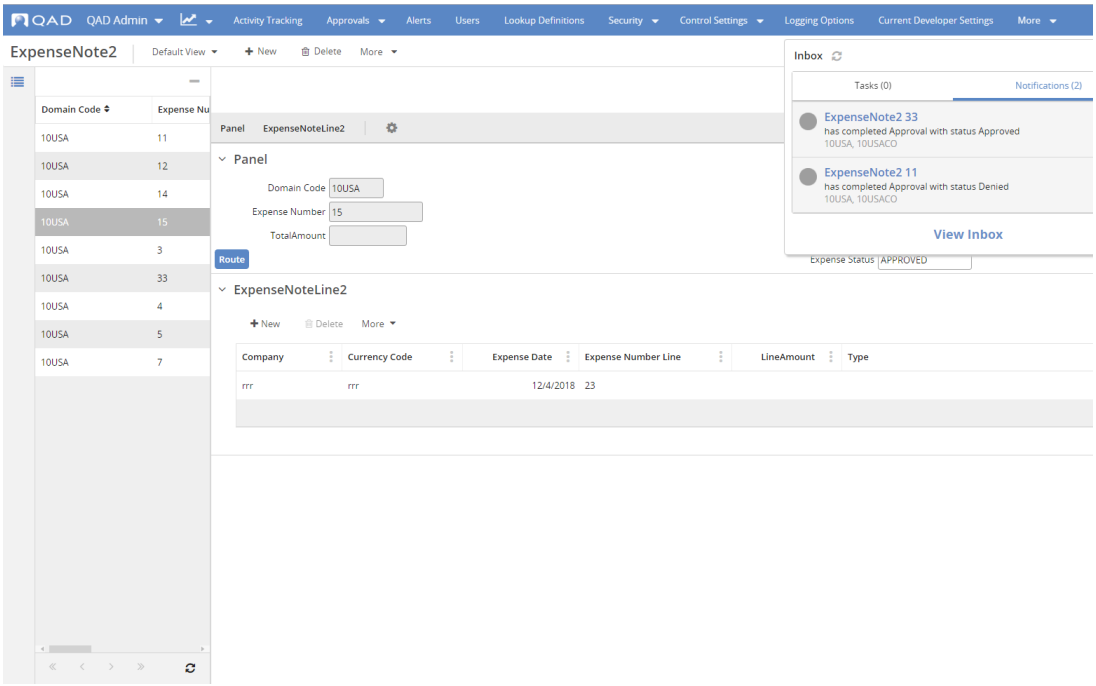
A. Task denied

After an approver denied the record, the end user will receive the notification message with the Denied status in the Inbox.



B. Task approved

After an approver approved the record, the end user will receive the notification message with the Approved status in the Inbox.


QAD
QAD Admin
Activity Tracking
Approvals
Alerts
Users
Lookup Definitions
Security
Control Settings
Logging Options
Current Developer Settings
More
ExpenseNote2
Default View
New
Delete
More
Domain Code
Expense Nu
10USA 11
10USA 12
10USA 14
10USA 15
10USA 3
10USA 33
10USA 4
10USA 5
10USA 7
Panel
ExpenseNoteLine2
Panel
Domain Code 10USA
Expense Number 15
TotalAmount
Route
Expense Status APPROVED
ExpenseNoteLine2
New
Delete
More
Company
Currency Code
Expense Date
Expense Number Line
LineAmount
Type
rrr
rrr
12/4/2018
23
Inbox
Tasks (0)
Notifications (2)
ExpenseNote2 33
has completed Approval with status Approved
10USA, 10USACO
ExpenseNote2 11
has completed Approval with status Denied
10USA, 10USACO
View Inbox

How to Remove a Business Component (Restricted)

If you want to remove (un-deploy) a business component that you have deployed, you can do so by following these steps:

1. Delete the business component tables from the extension database in the Progress Editor.

2. Call the following WSA services to find any data having anything to do with then components I wanted to delete (trial and error here!), then delete them:

- IBEBrowseObj
- IEntityBrowseObj
- IEntityMetadataV2Obj
- IViewResourceMetadata
- IViewMetadataV2Obj

3. Delete the secured resource for the business component, which also deletes its child resources, using the following WSA service, then refresh permissions:

- IResourceIdentityObj

Form Builder

This section describes the Form Builder, which you can use to build a view in the QAD Web UI.

Table of contents on this page:

Child pages of this page:

Introduction

Use Form Builder to build or edit the form for a business component. The Form Builder offers many possibilities for creating forms. In particular, the Form Builder features drag-and-drop controls for arranging the panels, sub-panels, and fields within a form.

Note that forms provided by QAD cannot be changed.

What is a Form?

A form organizes everything a user needs for some business task on a single page. In a form, a user can create, view, edit, and delete data. In a hybrid browse, which initially displays the full browse, the form opens when the user double-clicks a row (that is, a record) in the browse.

A form includes a summary panel, navigation bar, main panel, and then various detail panels and sub-panels.

Summary Panel

The summary panel displays key data about the form so that you know what you're working on at a glance. The summary panel is located between the menu bar and the navigation bar. These fields are read-only and stay in view even when you scroll down the page.

Navigation Bar

Just below the summary panel, you have a navigation bar. The navigation bar summarizes the various panels on the form, with drop-downs for the sub-panels. Use the navigation bar to jump quickly to any panel on the form. This is particularly useful when you are working with a long form and want to quickly go to a particular panel.

Main and Detail Panels

Related fields and functions are grouped within panels.

The main panel is the first panel located below the navigation bar, and is typically called Main. The main panel includes the most pertinent fields and controls, while the detail panels (and sub-panels) bring together various supporting fields and controls.

Each panel is represented on the navigation bar, with sub-panels included as drop-downs on the menu bar. You can quickly navigate to any panel or sub-panel from the navigation bar. You can also simply scroll up and down the form to access anything on the form. Everything you need is directly available on the page.

For a user, each panel is represented on the navigation bar, with sub-panels included as drop-downs on the menu bar. The user can quickly navigate to any panel or sub-panel from the navigation bar. Of course, the user can also simply scroll up and down the form to access anything on the form, or search the form using the web browser search controls (such as Ctrl + F).

Grids

Grids (or, form grids) list data in rows within a panel on a form. The QAD Web UI includes the following types of form grids:

- Single-row edit grid — only one row at a time can be edited.
- Multi-row edit grid — multiple rows can be edited at a time.
- Hierarchical grid — for hierarchical, parent-child data relationships

Personalization

Everything a user needs for a business task should be directly available on the form that you as a developer are building. However, keep in mind that an end user can personalize a form by clicking on the configuration ("gear") icon located on the navigation bar. The user can hide or show particular fields, but cannot move them around. A particular user might want to simplify the form to just show the fields they need for their task. As a developer of a form, remember that a user can hide or show particular fields as needed, but cannot change the layout itself.

Before You Begin Building

Before you begin building, you should have defined an app for the context in which your business component is built, assigned the data store, and created the business component using the Business Components view.

Building and Editing

With the View Builder, when you click the Build Form (or Edit Form) button, you can interactively build or change a form. Initially, the layout offers areas for the Summary Panel, Navigation Bar, and then the area for the various panels, sub-panels, and fields.

To begin building a form:

1. In the View Builder browse, search for the business component that you have created.
2. In the View Builder's form for the business component, click Build Form.
3. Build the form in the Build Form window (see Form Builder - Build Form).

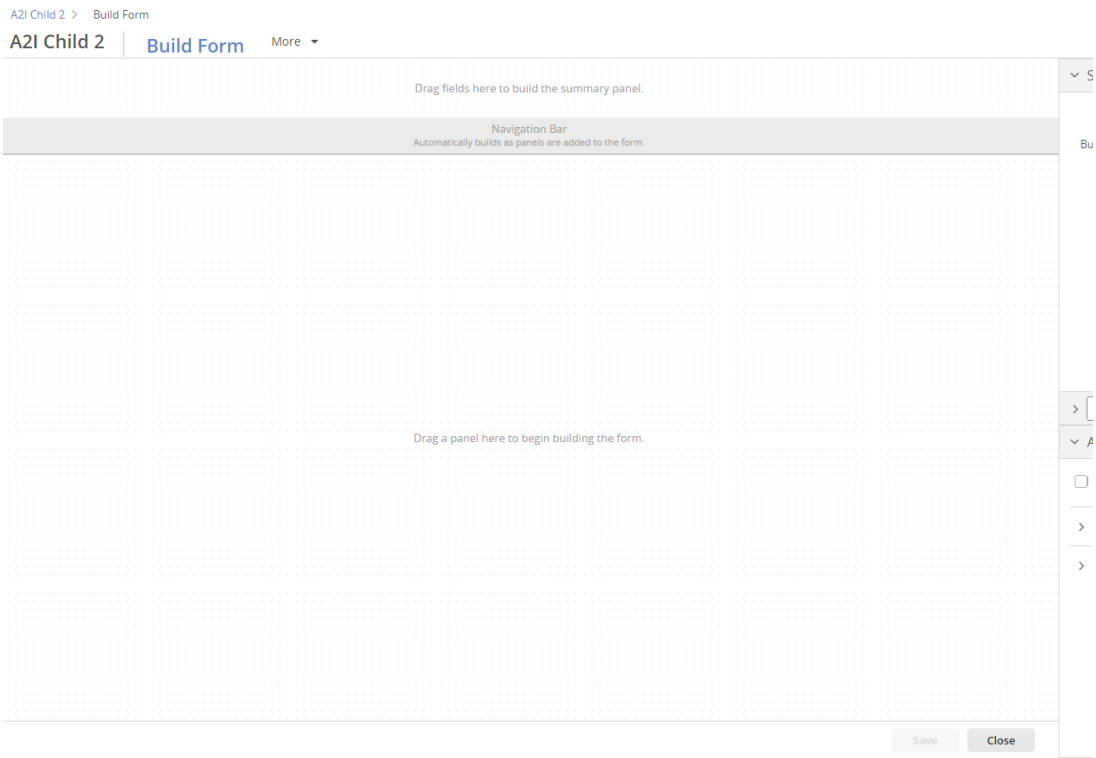
Form Builder - Build Form

Introduction

You can develop a form in the Build Form pop-up window, dragging-and-dropping UI Elements to position them as desired.

Note that forms provided by QAD cannot be changed.

Example Screen Shot



Building and Editing

With the View Builder, when you click the Build Form (or Edit Form) button, you can interactively build or change a form in the Build Form window.

Initially, the layout offers areas for the Summary Panel, Navigation Bar, and then the area for the various panels, sub-panels, and fields.

On the right-hand side, under Add To Layout, the UI Elements you can drag-and-drop to the form include:

- Panel
- Button
- Label
- Grid
- Group
- Custom

Under Fields, the Default and UI-Only Fields that are available for the business component are listed.

To build the form, in the Build Form window:

1. Drag-and-drop the panels you want to have on the form. Label these panels accordingly. The system will position the fields automatically based on the column view specifications.
2. Drag-and-drop the fields on to the respective panels. All of the relevant information you want to eventually maintain through the business component should be dragged on to these two panels.
3. Once you have set up the form, click Save. Once the form is created, you must add at least one hybrid browse to ensure the function can be accessed from the menu.
4. Under the Views panel, click New.
5. Choose Hybrid Browse and give it a name.
6. Link the browse.
   Verify that the key fields are mapped correctly to the fields on the browse in the business component.
   Ensure that Eligible for Menu is selected so this option can be seen from the menu.
7. Click Save. The form and hybrid browse are now available for the new business component.

You have now created the form with which users will interact with your business component.

UI Quick Reference

The Build Form window shows the layout of a form, including the Summary panel area, Navigation Bar area, and the form area for various panels and fields.

You can drag-and-drop fields to the Summary panel area, and drag-and-drop panels, fields, buttons, and other elements to the form area. The Navigation Bar automatically builds as panels are added to the form.

Along the right-hand side, you have panels for setting the layout properties, the properties for each element you have added to the form, and add elements to the form layout itself.

Set Layout Properties

App

Specifies the app for this view.

Business Component

Specifies the business component for this view.

Include Summary Panel

Specifies whether to include the Summary Panel in the layout.

Select drop-down

From the drop-down (initially named **<select>**), you can select an element currently on the form and view its details, which are displayed in a box below the drop-down option.

Panels

Panel Label

Specifies the label for the panel's name. Use the lookup to open the Labels pop-up window and select an existing label.

Columns

Indicates the number of field columns in the panel (set to 2).

Buttons

Element Name

Specifies the identifier for the button.

Button Label

Specifies the display label for the button. Use the lookup to select an existing label from the Labels pop-up window.

Width

Specifies the width of the button in pixels; currently, the width adjusts automatically and the setting is "auto".

Alignment

Specifies the alignment for a button within a cell. The following options are available:

- Left (Default). Left aligns a button in a cell.
- Right. Right aligns a button in a cell.
- Center. Center aligns a button in a cell.

Margin Left

Specifies the margin area on the left side of a button. The default value is 0 pixels.

Margin Right

Specifies the margin area on the right side of a button. The default value is 0 pixels.

State

Specifies whether the button is enabled or disabled.

Column Span

Specifies the number of columns that contain the button (set to 1).

Row Span

Specifies the number of rows that contain the button (set to 1).

Labels

Element Name

Specifies the identifier for the label.

Label

Specifies the display label. Use the lookup to select an existing label from the Labels pop-up window.

Width

Specifies the width of the label in pixels; currently, the width adjusts automatically and the setting is "auto".

Alignment

Specifies the alignment for a label within a cell. The following options are available:

- Left (Default). Left aligns a label in a cell.
- Right. Right aligns a label in a cell.
- Center. Center aligns a label in a cell.

Margin Left

Specifies the margin area on the left side of a label. The default value is 0 pixels.

Margin Right

Specifies the margin area on the right side of a label. The default value is 0 pixels.

Column Span

Specifies the number of columns that contain the label (set to 1).

Row Span

Specifies the number of rows that contain the label. Select from 1 to 10.

Grids

Detail Table

Specifies the detail (data) table for the grid.

Element Name

Specifies the identifier for the grid.

Grid Label

Specifies the display label for the grid.

Width

Indicates the width of the grid. This is set to auto: the grid automatically uses the area provided by the panel in which the grid is located.

Max Display Lines

Specifies the maximum number of grid lines to display.

Allow New

Specifies whether the New button is displayed for the grid, which allows users to add new lines.

Allow Select

Specifies whether the user can select lines in the grid.

Allow Edit

Specifies whether the Edit button is displayed for the grid, which allows users to edit lines.

Allow Delete

Specifies whether the Delete button is displayed for the grid, which allows users to delete lines.

Add Child Grid – Click to open the Grid Definition pop-up window, where you can specify a child grid.

Grid Columns

In Build Form, when you click a grid column header, the column's properties are listed in the right-hand panel.

Similar to the properties for fields, the column properties are organized into the Business Component Properties and the Display Properties. The Business Component Properties for a given column are the same as the properties for fields because the columns consist of business component fields, but displayed in columns. The Display Properties are similar to the field display properties, but include settings for behavior such as column sort order and a way to configure column styling for the use of color and icons.

Business Component Properties

Field

Indicates the identifier for the field.

Display Label

Specifies the display label. Use the lookup to select an existing label from the Labels pop-up window.

Business Component

Indicates the field's business component.

Detail Table

Indicates the detail table for the field.

Physical Field

Indicates the physical field identifier, based on the detail table and identifier for the field. This is useful for distinguishing between fields that might have similar identifiers and labels.

Max Characters

Indicates the maximum number of characters allowed for the field's data.

Format

Indicates the data format for the field. For example, a checkbox has the `Yes/No` format, while a field of up to 80 characters has the `x(80)` format.

Default Value

Specifies the default value for the field.

Required

Specifies whether the field is required.

Display Properties

Element Name

Indicates the element identifier for the field.

Control Type

Specifies the control the field uses for accepting and displaying data. For example, a checkbox field has the `CheckBox` control type, and a text string field has the `TextEditor` control type.

Width

Indicates the width available for the column's data.

State

Specifies whether the field is Enabled, Disabled, or Read-only.

Sort

Specifies the default sorting for the column as None, Ascending, Descending, or Disabled.

Sort Order

Specifies the column's order for sorting, relative to other columns. Select 1, 2, 3, or 4.

Conditional Styling

Specifies whether conditional styling is applied to the column's display of data. Click the gear icon to open Form Builder - Build Form - Grid Conditional Styling and configure the conditional styling.

Groups

Note that these settings pertain to a group of fields organized in a grid-like display within a panel. The term "grid" can sometimes be used in this context, but here it refers to the grid-like layout of the group of fields, rather than form data grids, which offer a browse-like display of data.

Element Name

Specifies the identifier for the field grid.

Columns

Specifies the number of columns in the group. From the drop-down, choose from 1 to 9.

Click the gear icon to configure the columns. In the **Configure Columns** pop-up window, specify the number of columns from the Columns drop-down (from 1 to 9), and then the **Column Width Type** and **Value** for each column. The Column Width Type can be one of the following: Auto size, Percentage, or Pixel; the Value is then set accordingly for the selected type.

Note that pixel column widths can be overridden by fields that require more width to display content.

Rows

Specifies the number of rows in the group.

Column Span

Specifies the number of columns spanned by the group. (Set to 1.)

Row Span

Specifies the number of rows spanned by the field grid (group). Select from 1 to 10.

Custom

Element Name

Specifies the identifier for the custom element.

Display Label

Specifies the display label. Use the lookup to select an existing label from the Labels pop-up window.

Column Span

Specifies the number of columns spanned by the custom element. (Set to 1.)

Row Span

Specifies the number of rows spanned by the custom element. Select from 1 to 20.

Fields

Business Component Properties

Field

Indicates the identifier for the field.

Display Label

Specifies the display label. Use the lookup to select an existing label from the Labels pop-up window.

Business Component

Indicates the field's business component.

Detail Table

Indicates the detail table for the field.

Physical Field

Indicates the physical field identifier, based on the detail table and identifier for the field. This is useful for distinguishing between fields that might have similar identifiers and labels.

Max Characters

Indicates the maximum number of characters allowed for the field's data.

Format

Indicates the data format for the field. For example, a checkbox has the `Yes/No` format, while a field of up to 80 characters has the `x(80)` format.

Default Value

Specifies the default value for the field.

Required

Specifies whether the field is required. On the form, required fields include the gold bar along the left side of the field value box.

Display Properties

Element Name

Indicates the element identifier for the field.

Control Type

Specifies the control the field uses for accepting and displaying data. For example, a checkbox field has the `CheckBox` control type.

Input Width

Indicates the width available for the field's data input.

Label Visibility

Indicates whether the label is Visible, Hidden, or None.

- Visible (Default). When selected, the label displays as usual.
- Hidden. When selected, the label is hidden but the label width is still accounted for.
- None. When selected, the label is removed and the label width is no longer accounted for. The Label Width field is hidden when this option is selected.

Label Width

Indicates the width available for the field's display label. This field is hidden when the None option under Label Visibility is selected.

State

Specifies whether the field is Enabled, Disabled, or Read-only.

Column Span

Specifies how many columns the field occupies (1 or 2).

Row Span

Specifies the number of rows the field occupies (from 1 to 10).

Add to Layout

From the Add to Layout panel, you can choose user interface elements to add to the form you are building.

Select **Show Unused Only** to have the Add to Layout panel only show elements that are not currently used on the form. While you are building a form, this option can help to make the panel easier to navigate and can prevent you from inadvertently adding the same element (such as a field) more than once on the same form.

Select from UI Elements or Fields to add elements to the form.

The listed fields are organized by field groups, including User Defined Fields, which are listed last.

UI Elements include:

- Panel
- Button
- Label
- Grid
- Group
- Custom

Under UI Elements, you can click an element, and then drag it to the form area, placing it where you would like the element on the form.

Fields include:

- Default – These fields are business component fields or field groups and you cannot edit or delete them after deployment. These fields are located on the Business Component page.
    - Suggestions – You can add fields from related business components to a form. These fields are disabled by default. If fields from the business component are used in any view metadata, they display under the Suggestions section.
    - +More – The button displays to add additional fields from the business component that do not display in the Suggestions section.
- UI-Only Field – You can drag-and-drop this field when building a form if you need a new field or when you cannot edit some of the Default fields.

Field Properties

Field properties include Business Component Properties and Display Properties.

Form Builder - Views

Introduction

Views provide the most common way to interact with a BC from the Web UI. The following view types are available:

- *Hybrid Browse*. A hybrid browse is comprised of a browse in which every line represents a BC instance (for example customer with code "10-100" is an instance of the Customer BC), and a form opens when a line in the browse is selected (by double-clicking the line or by clicking the Edit or New button). The form represents the BC and visualizes all information available for the BC, as defined in the BC definition itself.
- *Form Only*. When Form Only is selected, the Browse and Drill Downs panels are hidden, and only the form is available.

You can define one or more views for a BC. The different hybrid browses will all have the same form because you can only define one form, but you can define different browses.

BCs can have two types of browses: either a *standard browse* or a *business component browse*. In order to deploy a BC, there must be at least one browse associated with it.

What is a Hybrid Browse?

Hybrid browses are browses that also include a form for viewing and editing records. A hybrid browse includes a toolbar along the top, a grid (or browse) displaying a list of records, and a form for completing work. (Note that the term "hybrid view" is equivalent to "hybrid browse".)

A hybrid browse in the QAD Web UI is similar to a hybrid maintenance screen in the QAD .NET UI that combines a browse and frame-based form.

Keep in mind that there is not necessarily a direct one-to-one mapping between a screen in the QAD .NET UI and a related view in the QAD Web UI. A view in the QAD Web UI could combine and consolidate the functions offered by several screens in the QAD .NET UI.

What is a Browse?

A browse displays records in a grid layout of rows and columns. Users can quickly search for data, add search conditions, sort on columns, manage the display using paging controls, and hide and show columns. Further, users can save column configurations and search filters as stored searches as part of a stored view.

Actions

Actions are only available for a Hybrid Browse view. The Actions drop-down can offer actions you can take for a record or set of records:

- *Individual actions* are actions applied to individual records.
- *Bulk actions* are actions applied to a set of records that are selected based on browse search criteria, and any automatically applied required criteria. For bulk actions, once you select the records and set options, you can simulate the action or proceed to submit the action.

You can define actions on the Actions panel included in the Form Builder - Views pop-up window.

Understanding Browse Types

Standard Browse

This browse is a standard application browse (a QAD Enterprise Applications ".p" browse). A standard browse cannot be modified using the Platform tools in Channel Islands; however, these standard browses can be modified using Browse Maintenance in the QAD .NET UI.

The URI of a standard browse has the following pattern: `uri:browse:*`

Business Component Browse

A business component browse is a browse view on the BC and its relations. This type of browse is data driven and can be configured using the Platform tools in Channel Islands.

There can be more than one business component browse for a given BC.

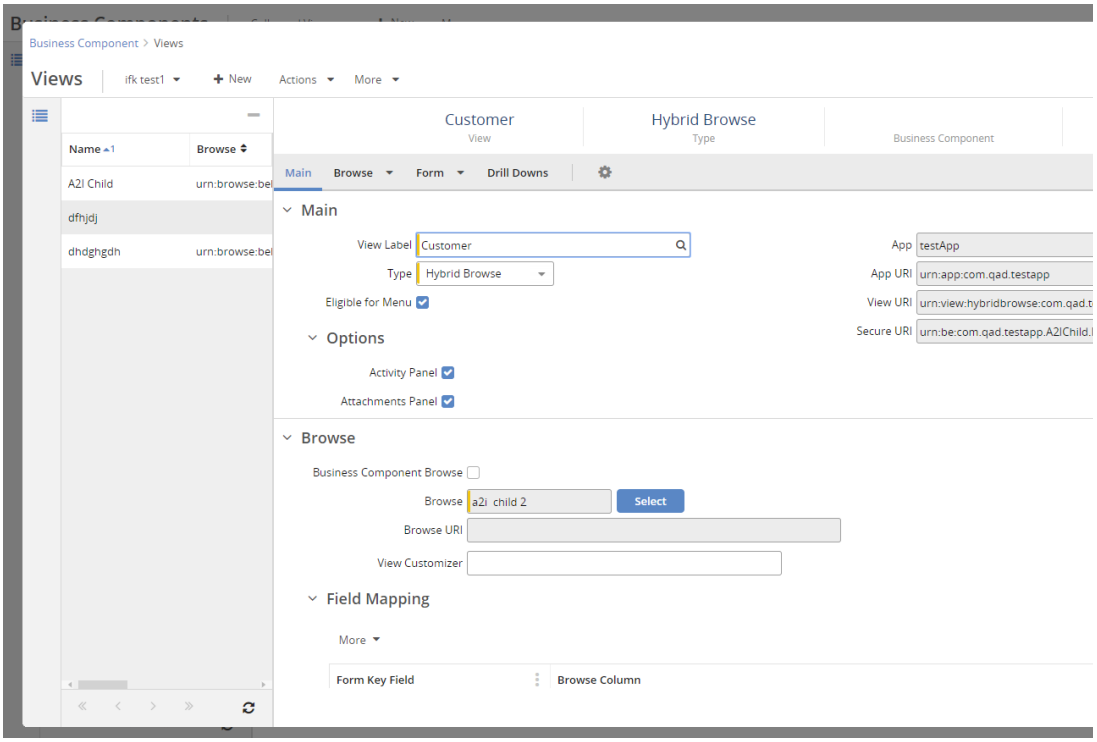
The URI of a business component browse has the pattern: urn:browse:be:*

The BC browse can include the following fields:

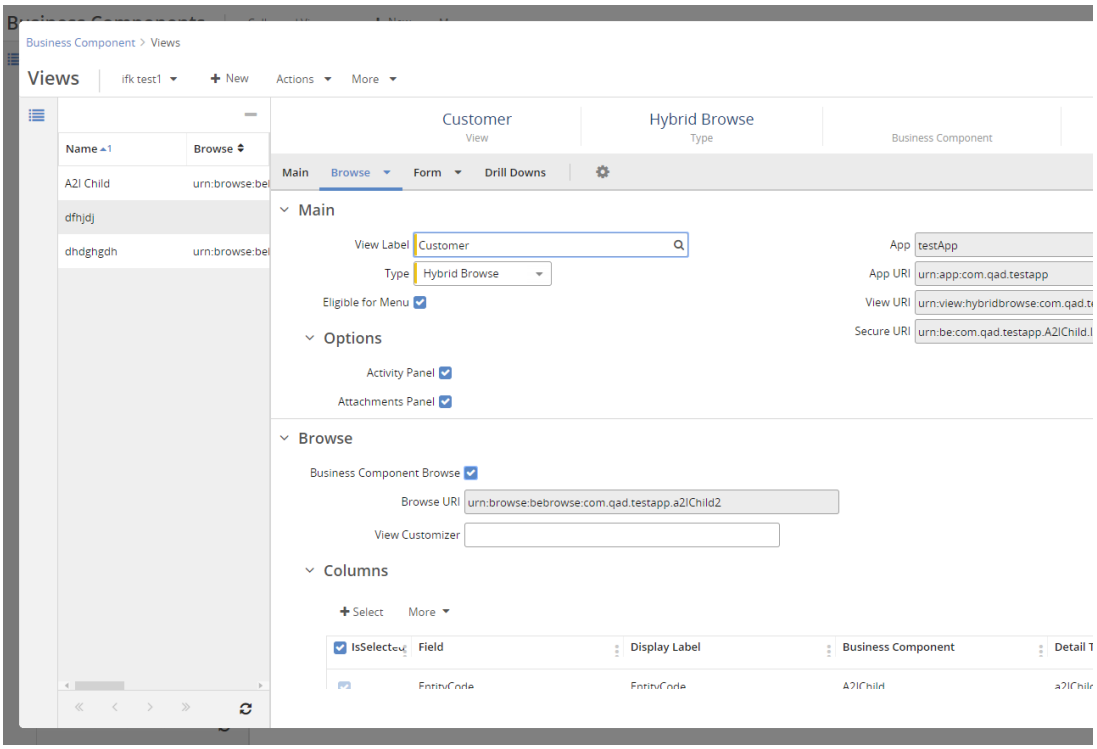
- Fields from the main table of the BC
- Fields of embedded BCs for which there is an extension relationship
- Fields of the main table of BCs that are related in 1-1 or N-1 fashion
- The key fields of the main table of the BC are always selected, and they cannot be removed from the browse

Example Screen Shots

Standard Browse Example



Business Component Browse Example



Drill Downs

As a user, you can access the drill downs for a hybrid browse from the right-hand "drill down links" icon, which opens a panel along the right listing the available drill downs. As a developer, with the Form Builder, you can manage drill down links from the Drill Downs panel on Form Builder - Views, from which you can use the Drill Downs pop-up window to add drill downs.

Creating, Editing, and Deleting

You create a hybrid browse by first selecting the type of browse: standard or business component browse.

If you selected business component browse, you can then select the columns you want to have the browse display for the user.

Next, the browse needs to be linked to the form by selecting the correct ID fields.

After the hybrid browse is saved, you can edit it further as needed.

Deleting a hybrid browse is only possible if it was created in your app. The QAD-provided hybrid browse for a coded business component can never be deleted. For a virtual BC, there must always be at least one hybrid browse.

UI Quick Reference

The Views pop-up window includes the following panels and fields:

Main panel

View Label

The label to display for the view.

Type

Select the view type: Hybrid Browse (includes a browse and a form) or Form Only (only includes a form).

The Business Component's Hybrid Browse view includes the following panels:

- Main
- Browse
- Form
- Drill Downs

The Business Component's Form Only view includes the following panels:

- Main
- Form

Eligible for Menu

Specifies whether this view will be available on the menu (and menu search).

App

Indicates the app to which this view belongs.

App URI

Indicates the Uniform Resource Identifier (URI) for the app to which this view belongs.

View URI

Indicates the URI of the view.

Secure URI

Indicates the URI for security and menu linking.

Options panel

Activity Panel

Specifies if this view includes an Activity panel. With the Activity panel, you can see a history of transactions and follow the things you care about most.

Attachments Panel

Specifies if this view includes an Attachments panel. With the Attachments panel, you can attach files to views to keep relevant documents together for a particular transaction.

Browse panel

Includes settings for the Hybrid Browse view.

Business Component Browse

Specifies whether the hybrid browse is based on a business component or is a standard browse.

The standard browse includes the following panels:

- Field Mapping
- TS Handlers
- Actions

The business component browse includes the following panels:

- Columns
- Column Order
- TS Handlers
- Actions

Browse

Specifies the standard browse for this hybrid browse. Click **Select** to choose a standard browse. This field is displayed only if Business Component Browse is not selected.

Browse URI

The URI of the browse.

View Customizer

Specifies a Java resource that applies view metadata changes before the browse is rendered (example: com.qad.acme.mvc.CustomerBrowseViewCustomizer).

Field Mapping panel

This panel is displayed if you are working with a standard browse.

Mapping grid for mapping form key fields with browse key fields, with following columns:

Form Key Field

Specifies the key field of the form.

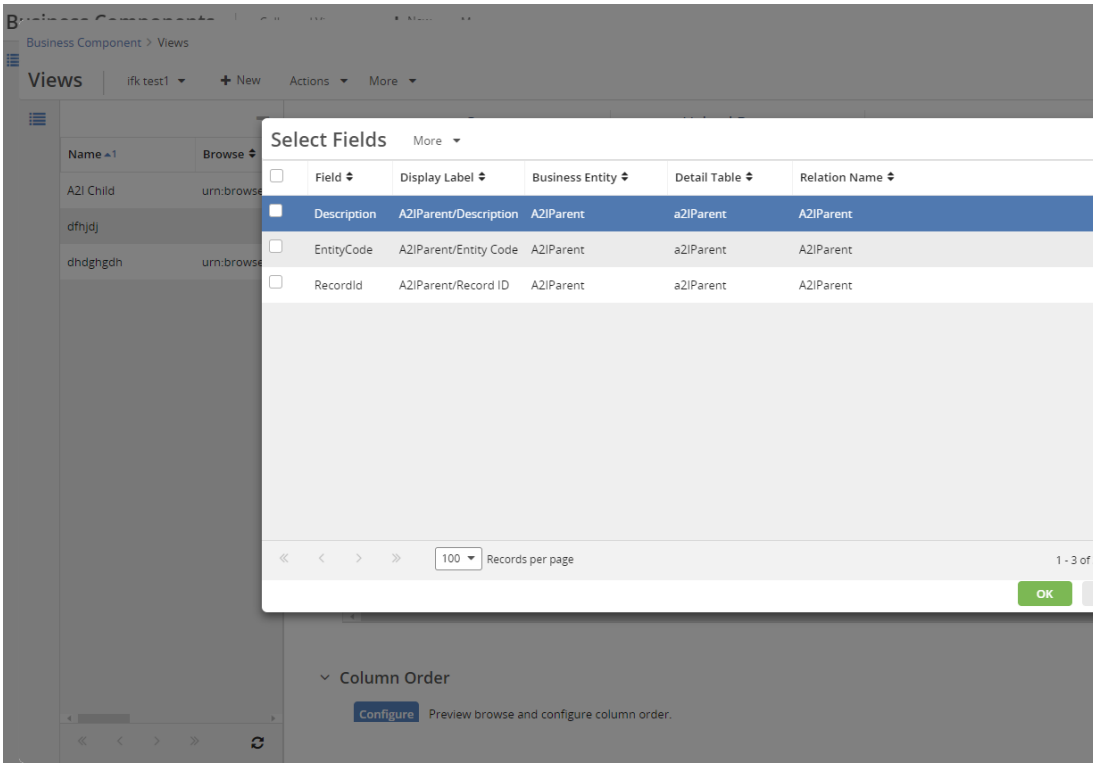
Browse Column

Specifies the corresponding key field on the browse.

Columns panel

This panel is displayed if you are working with a business component browse.

The grid lists the browse columns, initially including all the business component fields. You can remove columns by selecting the checkbox for each corresponding field. You can add fields by clicking the +Select button. The fields you select can be from the current business component or from other business components that have a lookup relationship with the current business component.

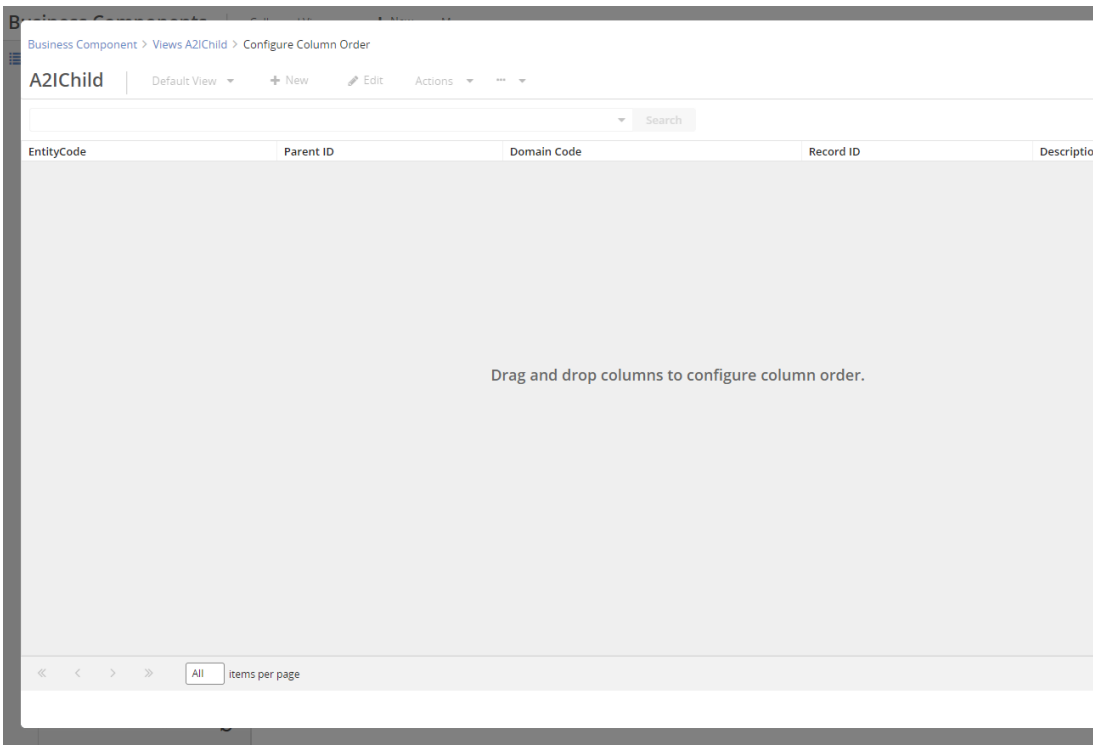


Column Order panel

This panel is displayed if you are working with a business component browse.

By default, the browse column order is based on the order of appearance of the fields on the form. However, depending on your requirements, the first field that displays on the form might not be the best choice for the first column in the browse (and so on).

Click **Configure** to preview and configure the browse column order. In the Configure Column Order pop-up window, you can drag-and-drop columns to configure the column order.



TS Handlers panel

Specifies TypeScript (TS) handlers located on the server.

This panel is for adding coded Browse TS handlers. These TS handlers should exist on the server in .js files.

Actions panel

Specifies bulk or individual actions that can be included as part of this Hybrid Browse view. The actions then become available from the Hybrid Browse view's Actions drop-down.

The action itself can currently only be written in standard code.

To add a new action, click **New** and complete the new row in the grid.

Type

The type of action: Bulk or Individual.

ID

The unique identifier for the action.

Action Label

The display label for the action. This value for this label is displayed in the Actions drop-down.

Secure URI

The URI for the action's resource, typically starting with the format urn:view:maint:com.qad.com.*

Required Permission

The access permission required for the action (example: Read).

Form panel

View Customizer

Specifies a Java resource that applies view metadata changes before the form is rendered (example: com.qad.erp.base.mvc.ViewCustomizer). This applies only to coded BCs.

Uses Domain

Specifies whether to use the current domain.

Resource URL Prefix

Specifies resource URL prefix, such as "erp".

Data Resource

Specifies data resource. Typically this is in camel-case format (example: salesOrders).

Table

Specifies a database table.

TS Handlers panel

Specifies coded form TypeScript (TS) handlers located on the server.

These TS handlers should exist on the server in .js files.

Drill Downs panel

Specifies the drill downs for this Hybrid Browse view.

Currently, the Views screen only supports *Hybrid Browse* and *Form Only* views. It does not yet support *Browse Only* views.

Because configuring *Browse Only* views are not yet supported by the Views screen, note that it is not yet possible to specify drill downs for *Browse Only* views using this Drill Downs panel on the Views screen.

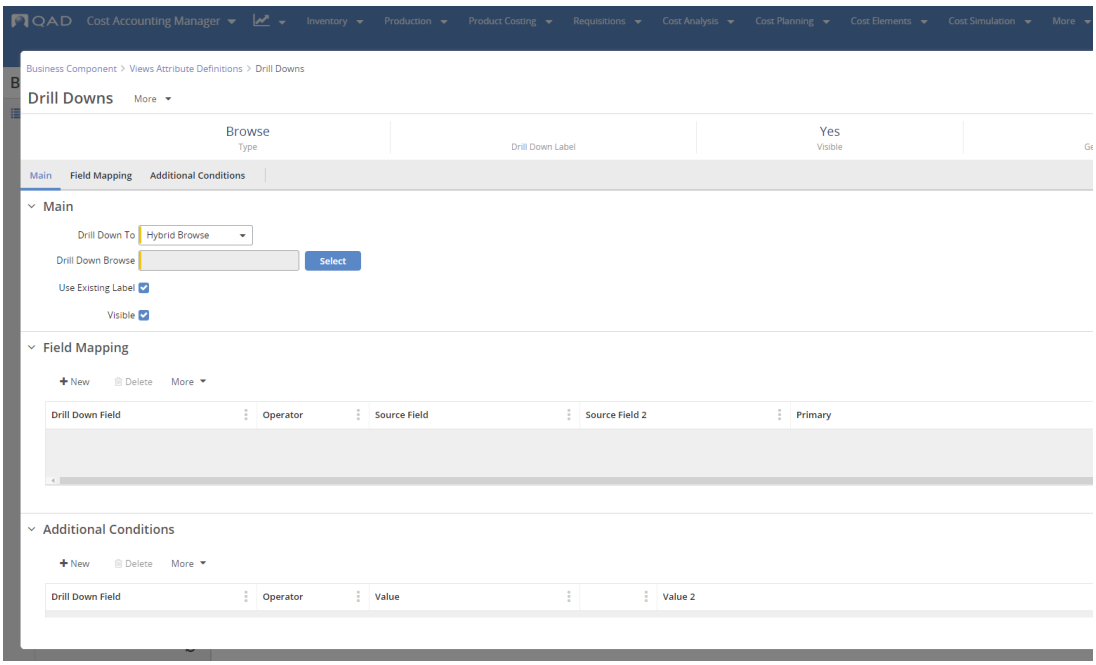
The grid displays a list of the drill downs. You can add, modify, or delete drill downs using the Drill Downs pop-up window; see Form Builder - Views - Drill Downs.

Form Builder - Views - Drill Downs

Introduction

From the Drill Downs pop-up window, you can add, modify, or delete drill downs.

Example Screen Shot



UI Quick Reference

Main panel

Drill Down To

You can have the drill down go to a Hybrid Browse, Browse Only, Report, or External Link.

Note: If you choose the External Link type, you will need to enter a link template to the external site on the URL panel. If you select the Modal Window option from the Open In drop-down menu, then the same-origin mechanism will be applied. In case the external resource does not allow to be opened in iframe, in the runtime you will see an empty modal window.

Drill Down Hybrid Browse

(Displayed if Drill Down To is set to Hybrid Browse.) Specifies the hybrid browse for the drill down. Click **Select** to open the Resource pop-up window, from which you can select a hybrid browse. Note that the URI for hybrid browses uses the format `urn:view:hybridbrowse:*`.

Drill Down Browse

(Displayed if Drill Down To is set to Browse Only.) Specifies the browse only for the drill down. Click **Select** to open the Resource pop-up window, from which you can select a browse. Note that the URI for browses only uses the format `urn:browse:*`.

Drill Down Report

(Displayed if Drill Down To is set to Report.) Specifies the report for the drill down. Click **Select** to open the Resource pop-up window, from which you can select a report. Note that the URI for report uses the format `urn:report:*`.

Use Existing Label

Select whether to use the existing display label for the hybrid browse, browse only, or report. Clear the checkbox to choose a different label.

Visible

Select this checkbox to make the drill down visible in the runtime.

Drill Down Label

(Displayed if Use Existing Label is not selected.) Use the lookup to open the Labels pop-up window and select a different label.

Field Mapping panel

This panel includes the grid to map source browse fields with drill down browse fields.

Source Field

Specifies the field from which you want to map the drill down.

Source Field 2

Specifies the second source field for range operators when the first Source Field is a range type. This source field is disabled for all other operator selections.

Drill Down Field

Specifies the field to which you want to map the drill down.

Operator

Shows a drop-down menu with available operators based on the selected Drill Down Field.

Primary

The primary field drives the drill down order. Select one field as the primary field for the drill down.

Additional Conditions panel

This panel is optional and includes the additional conditions for the hardcoded values of drill down fields.

Drill Down Field

Specifies the field to which you want to map the drill down.

Operator

Shows a drop-down menu with available operators based on the selected Drill Down Field.

Value

Specifies the hardcoded value.

Value 2

Specifies the second value for range operators when the first Value is a range type. This value is disabled for all other operator selections.

Form Builder - Event Handlers

Introduction

The Event Handlers maintenance UI allows for creating, enabling, or disabling custom Event Handlers for specified Business Component.

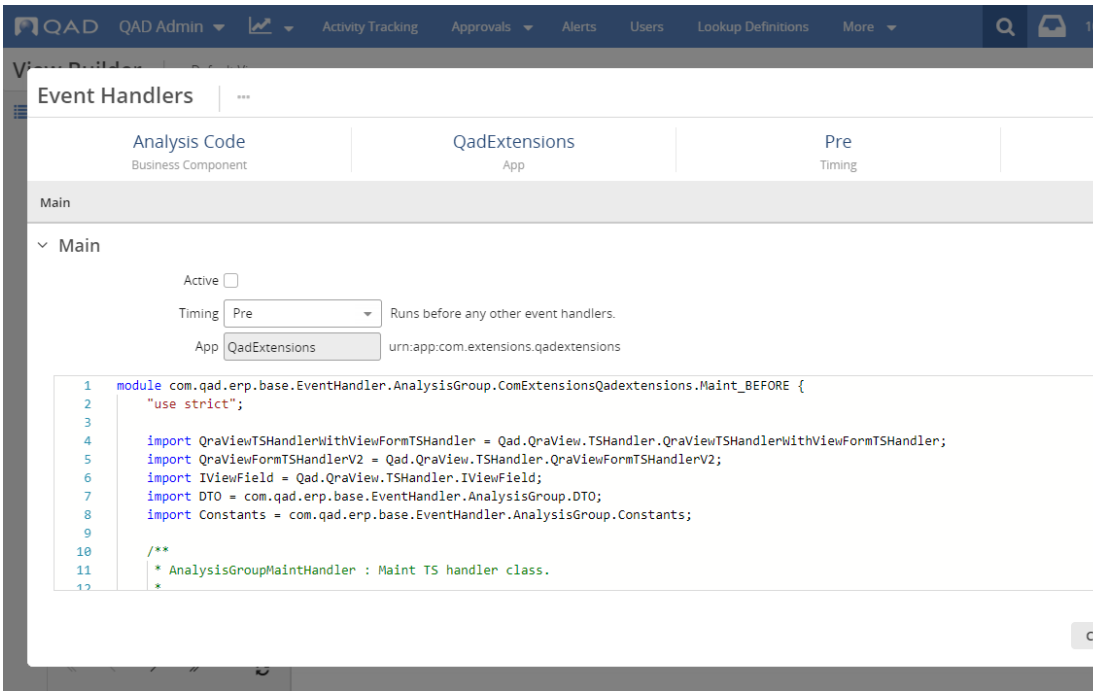
What is an Event Handler?

An Event Handler is a TypeScript code which is compiled into JavaScript at runtime and is saved to database so can be executed for corresponding Business Component.

There are three types of Event Handlers:

1. Primary – primary event handler for the corresponding business component. Can be created only for virtual BC and is the same as an event handler for coded BC. DB value: PRIMARY.
2. Pre – runs before Primary or coded event handler. DB value: BEFORE.
3. Post – runs after Primary or coded event handler. DB value: AFTER.

Example Screen Shot



Creating, Editing, and Deleting

The developer can create new Event Handlers by following rules:

Intended action	Is virtual BC	Definition is in same App	Behavior
	yes	yes	Not possible

Add “pre/post” event handler			
Add “pre/post” event handler	yes	no	Possible - after selecting new, the developer needs to be able to specify “pre” or “post”
Add “pre/post” event handler	no	no	Possible - after selecting new, the developer needs to be able to specify “pre” or “post”
Add “pre/post” event handler	no	yes	Possible - after selecting new, the developer needs to be able to specify “pre” or “post”
Add “primary” event handler	yes	yes	Possible - after selection new, the developer does not have to specify “pre” or “post”, as it is always “primary”.
Add “primary” event handler	yes	no	Not possible
Add “primary” event handler	no	no	Not possible
Add “primary” event handler	no	yes	Not possible

Table description:

- If BC is coded, then the developer can create one Pre and Post Event Handlers for each App which is active at the moment.
- If BC is virtual and is created in the same App which is active at the moment, then the developer can create only Primary Event Handler.
- If BC is virtual and is created in another App compared to active one at the moment, then the developer can create Pre and Post Event Handlers.

As for now, the developer can edit and delete any Event Handler without any restrictions.

UI Quick Reference

Main panel

Active

Indicates whether the event handler is active and will be executed at runtime.

Timing

Indicates the selected event handler type.

App

Indicates the app name and URI in which the event handler will be created (that is, the current, active app) or is already created.

TypeScript code area

The code that will be compiled into JavaScript and executed at runtime.

Form Builder - Grid Definition

Introduction

With the Grid Definition pop-up window, you specify a grid that you are adding to a form.

Data Grid Types

From a developer's perspective, grids can be either ***internal*** or ***external***. An internal grid displays data from the same business component upon which the form itself is based. In contrast, an external grid displays data from some business component other than the one upon which the form is based.

From a user's perspective, internal grids are "multi-row edit grids" and external grids are "single-row edit grids". The internal, multi-row edit grids are grids where one can edit multiple rows at a time and then save the various changes when the form itself is saved. The external, single-row edit grids are grids where one can only edit and then save only one row at a time, independently from the data for the rest of the form.

As a developer using the View Builder, when you add a grid, the View Builder determines whether the grid is an external grid or internal grid based on the grid detail table's business component code. For a given grid, if the selected detail table's business component code is the same as the form's business component code, then the grid is an internal grid. If not, the grid is an external grid.

Grids, whether internal or external, can be ***hierarchical***, where the grid data is based on parent-child relationships.

Further, hierarchical grids can be either ***recursive*** or ***non-recursive***:

- Recursive hierarchical grids are grids where the child rows are from the same table as their parent rows.
- Non-recursive hierarchical grids are grids where the child rows are not from the same table as their parent rows.

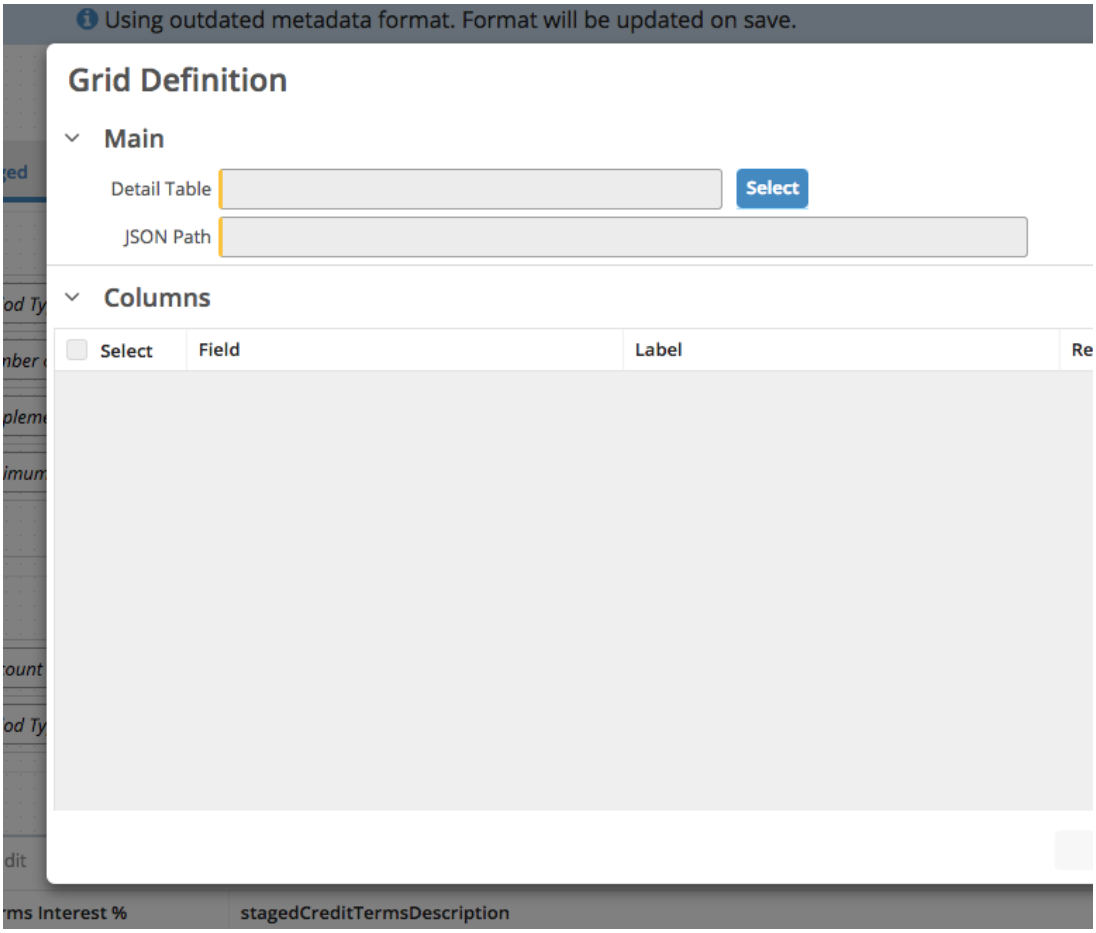
Currently, only external grids can be recursive hierarchical grids. Internal recursive hierarchical grids are not supported at this time.

Grid Conditional Styling

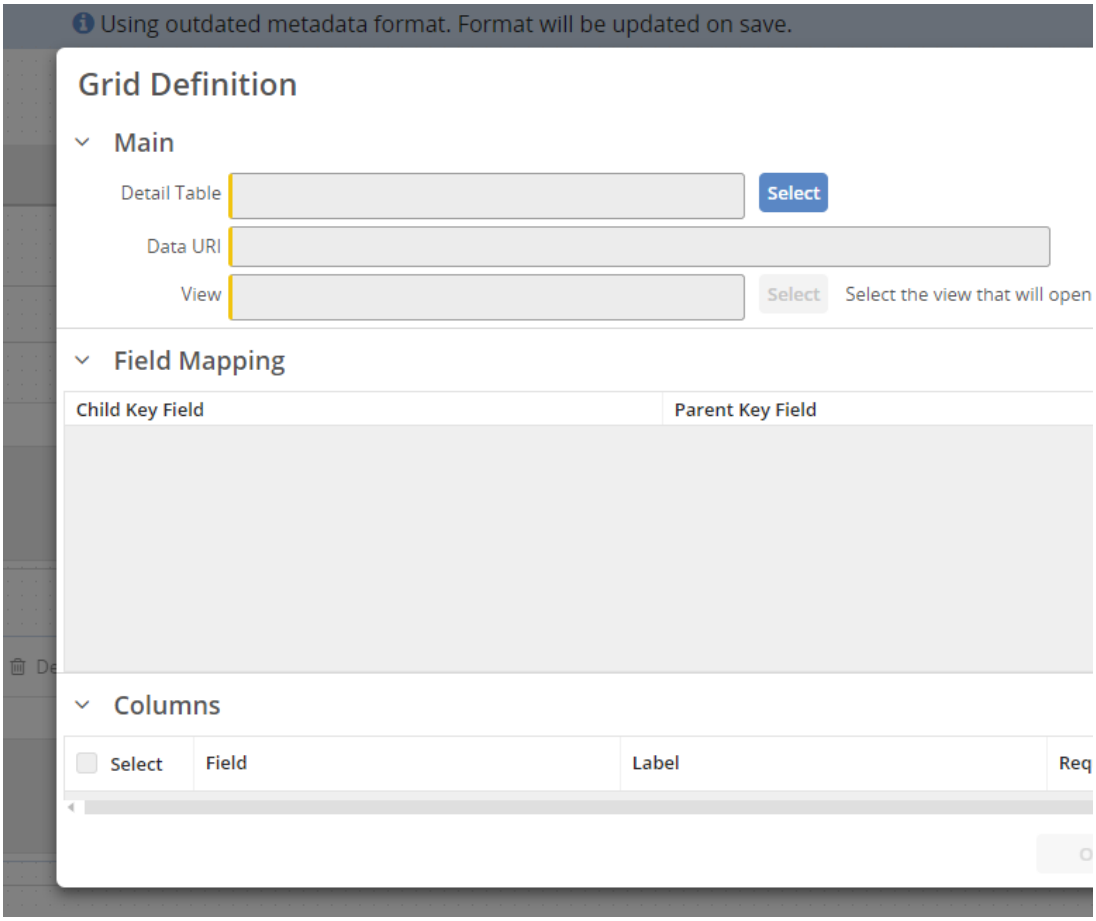
Note that you can apply styling, including background color and icons, to grid columns (see Form Builder - Build Form - Grid Conditional Styling).

Example Screen Shots

Adding grid from the Build Form window:



Adding Child Grid:



UI Quick Reference

Grid Definition

These settings define a data grid in a form.

Main panel

Detail Table

Indicates the business component data table for the grid. Click Select to open the Detail Grids pop-up window for selecting a data table.

JSON Path

For an internal grid, the JSON path to the top level of the data to bind to the grid. Note: If you are converting a Java View Controller, this value can be taken directly from setRecordListFieldName(). In the view resource metadata, this is the value of <RecordListFieldName>.

For an external grid, specifies parameters for grids that display data that is external to the form's business component (single-row edit grids). In the view resource metadata, this is the value of <ExternalGridParameters>.

Data URI

Indicates the URI for data based on the business component detail table.

View

Select the view that will open from the Details button.

Field Mapping panel

Child Key Field

Indicates the child key field for a given column.

Parent Key Field

Indicates the parent key field for a given column.

Columns panel

Select

Indicates whether the column is displayed in the grid.

Field

The field identifier.

Label

The field display label for the field, displayed in the grid column header.

Required

Indicates whether data is required.

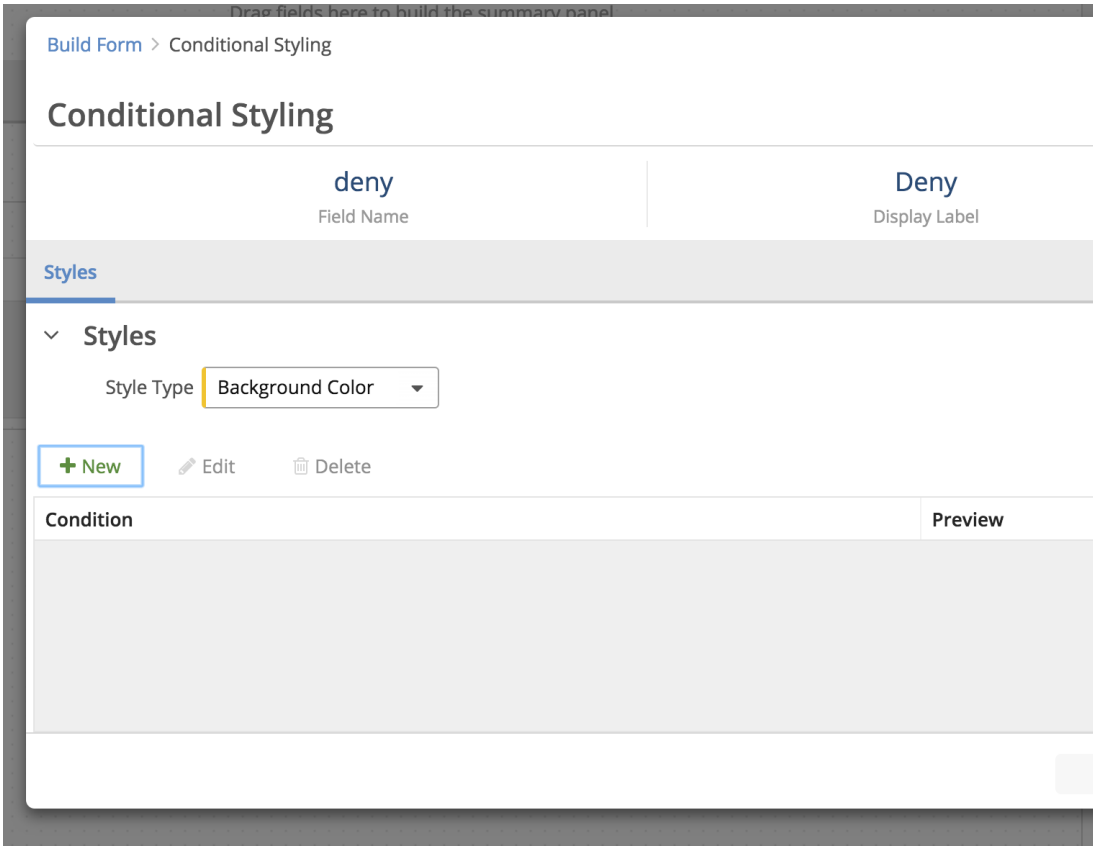
Form Builder - Build Form - Grid Conditional Styling

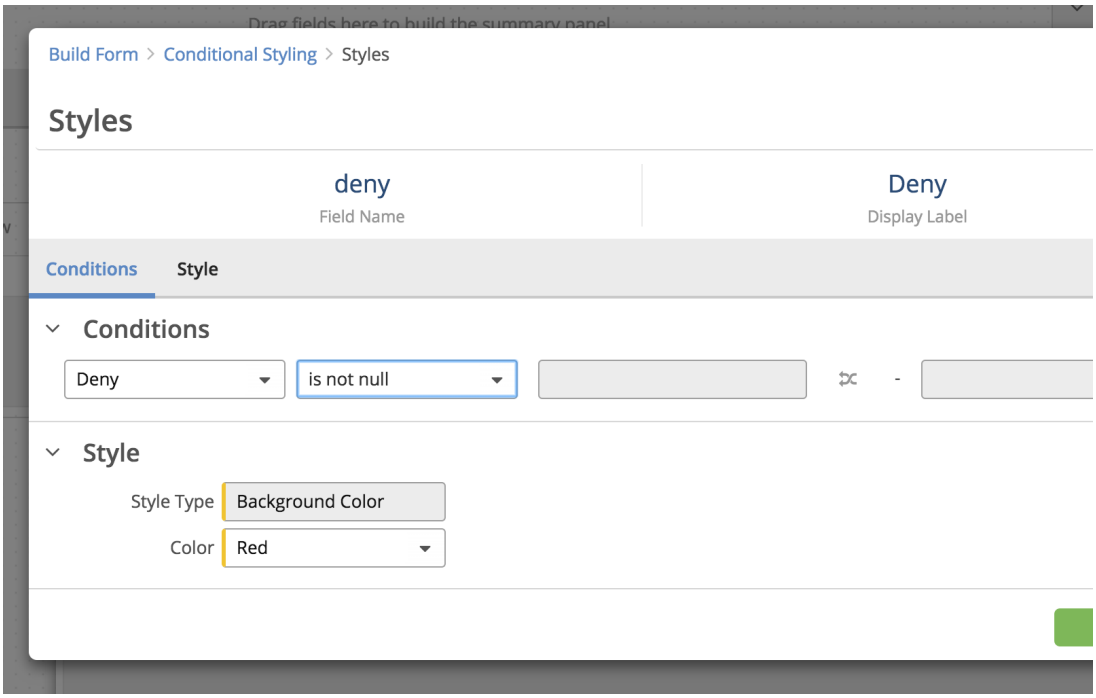
Introduction

With the Conditional Styling pop-up window, you can configure conditional styling for a grid column. You can specify conditions for the background color or the use of an icon.

The Conditional Styling pop-up can be accessed from Form Builder - Build Form. To do so, click a grid's column header, and then, from the right-hand properties panel, by Conditional Styling, click the gear icon.

Example Screen Shots





UI Quick Reference

Conditional Styling

From the Conditional Styling pop-up window, you can get started with specifying conditional grid column styling.

Styles

Style Type

Choose the type of styling as Background Color or Icon.

The grid displays any currently specified styles. Click **New** to add a new style, **Edit** to modify an existing style, or **Delete** to remove a style.

Condition

The condition being applied.

Preview

Shows a preview of the applied condition.

Background Color Styles

When you click **New** with Background Color selected (or **Edit** an existing style for background color), the Conditional Styles - Styles pop-up window opens, where you can specify conditions for styling the background color for the field's column.

Conditions

In Conditions, for the current field's column, you can specify conditions using any of the fields on the grid. You can specify multiple conditions for the current field's column by clicking the **+** icon (and remove a condition by clicking the **x** icon), in the same way that you specify browse search conditions.

Field drop-down

From the drop-down listing the available fields for the grid, select a field that you want to use to create a condition.

Operator drop-down

From the drop-down listing, choose:

- equals
- greater than
- greater or equal to
- is not null, is null
- less than
- less or equal to
- contains
- starts with
- ends with
- does not equal
- range

In the adjacent two fields, enter the value(s) for the conditions.

Use the **+** and **x** icons to add or remove conditions.

Style

Style Type

Set to Background Color.

Color

Select from the following colors: Blue, Green, Magenta, Orange, Purple, Red, Turquoise, or Yellow.

Icon Styles

When you click **New** with Icon selected (or **Edit** an existing style for an icon), the Conditional Styles - Styles pop-up window opens, where you can specify conditions for styling the icon for the field's column.

Conditions

In Conditions, for the current field's column, you can specify conditions using any of the fields on the grid. You can specify multiple conditions for the current field's column by clicking the **+** icon (and remove a condition by clicking the **x** icon), in the same way that you specify browse search conditions.

Field drop-down

From the drop-down listing the available fields for the grid, select a field that you want to use to create a condition.

Operator drop-down

From the drop-down listing, choose:

- equals
- greater than
- greater or equal to
- is not null, is null
- less than
- less or equal to
- contains
- starts with
- ends with
- does not equal
- range

In the adjacent two fields, enter the value(s) for the conditions.

Use the **+** and **x** icons to add or remove conditions.

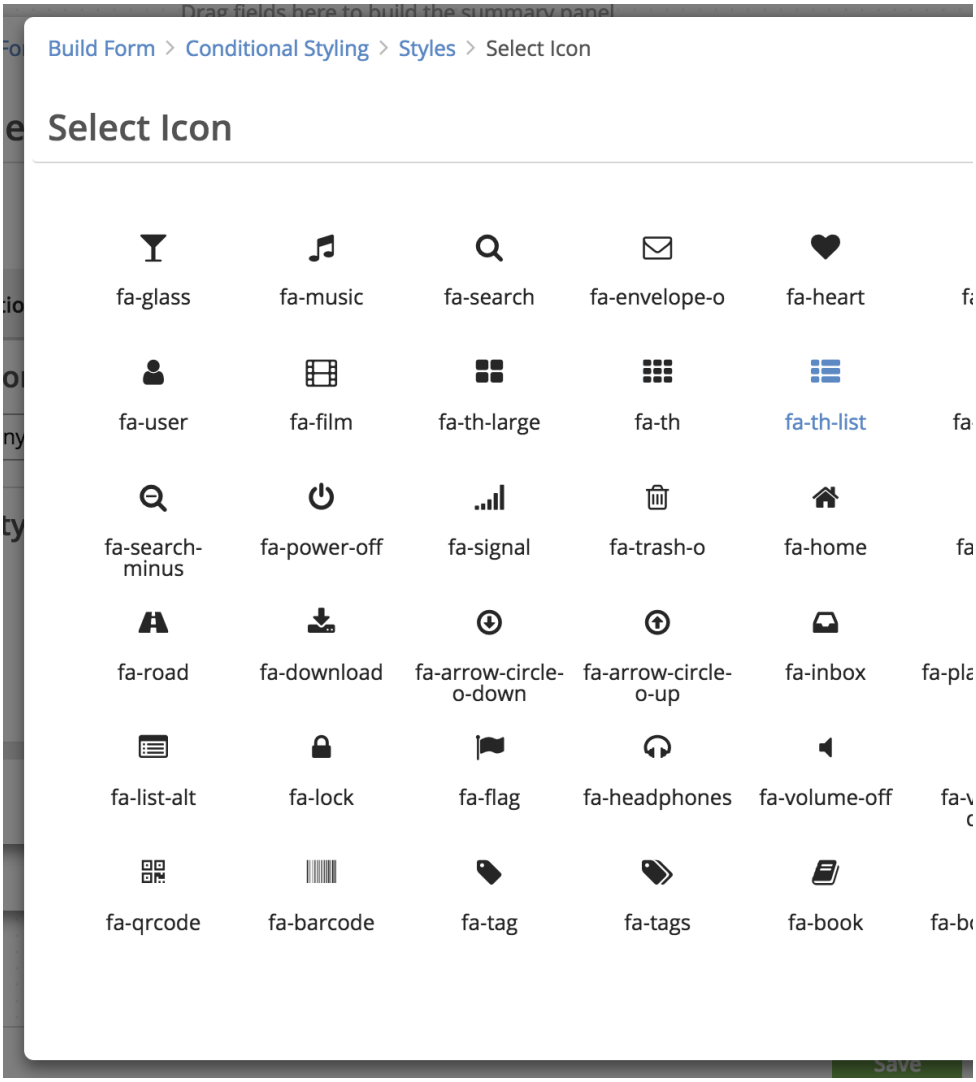
Style

Style Type

Set to Icon.

Icon

Click Select to open the Select Icon pop-up window that offers many icons to use.



Icon Color

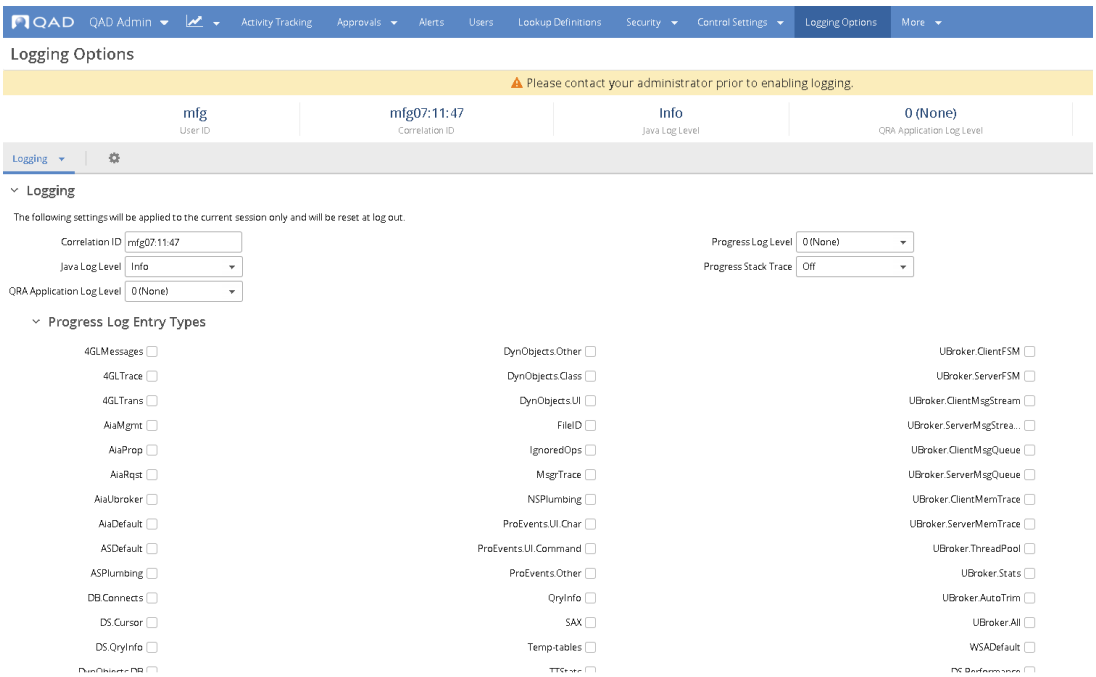
Select from the following colors: Blue, Green, Grey, Magenta, Orange, Purple, Red, Turquoise, or Yellow.

Logging Options view

IN PROGRESS

Introduction

The Logging Options page allows a user to set up specific logging levels for Java and Progress code only for current user session. It also provides the ability to set up a "Correlation ID" marker, which will then be present in each log line in Java logs and in specific lines in Progress logs so that we can correlate Java and Progress logs related to current user session. When users sign off from the Web UI, those settings are then gone; if they log in again, they will get the default logging settings.



UI Quick Reference

Logging panel

Correlation ID

A marker that you can set and then search for this marker in the logs to identify the logs from the current session.

Java Log Level

Specify the log level (info, warning, and so on) for Java logging.

QRA Application Log Level

Specify the log level for QRA application logging.

Progress Log Level

Specify the Progress log level.

Progress Stack Trace

Turn the Progress stack trace on or off.

Progress Log Entry Types panel

Set additional log entry types of Progress logs, which can be switched on or off by selecting each checkbox. Typically, 4GLTrace is selected.

Lookup Definitions view

Introduction

A lookup gives users a way to select a value from a browse instead of having to enter the value manually. On the user interface, a lookup can be included on various input fields, indicated by a magnifying glass icon located on the right side of the field.

With Lookup Definitions, you can define these lookups: you can view lookup definitions, create new lookup definitions, and edit lookup definitions.

As an administrator, you can configure lookups to apply to specified fields under certain conditions. Further, you can have the lookup on a particular field change depending on application conditions.

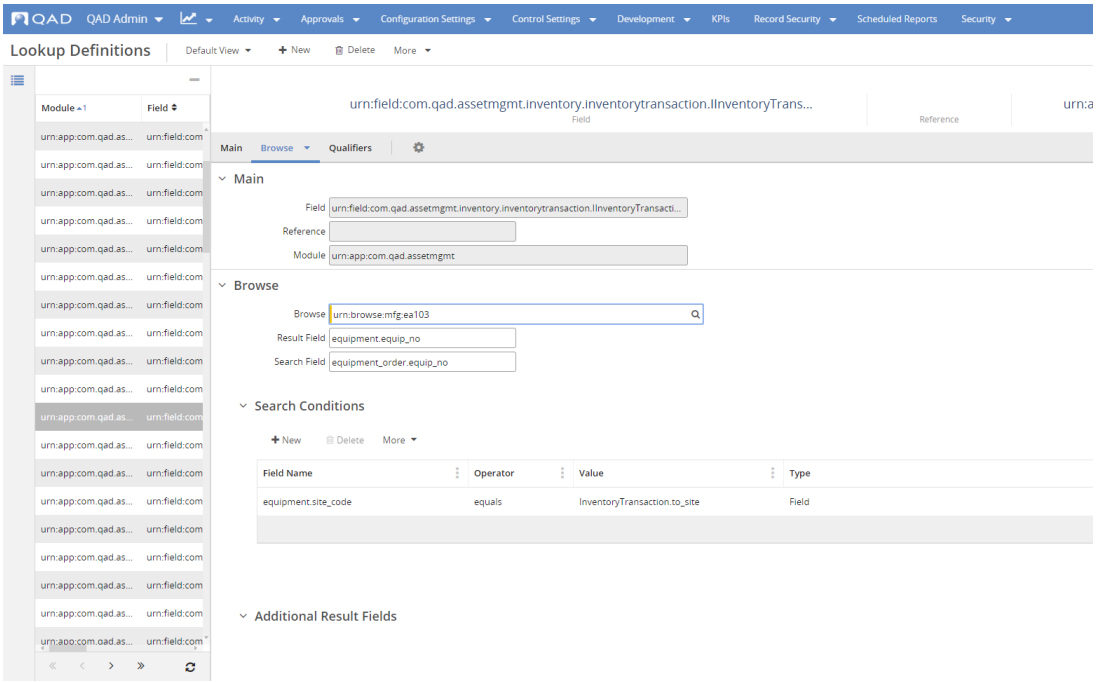
You can define lookups for a business component field or at the level of a view field. Lookups defined for a business component field can then be used by all the view fields that are based on the business component field. Lookups defined for a view field can only be used by the field in that particular view.

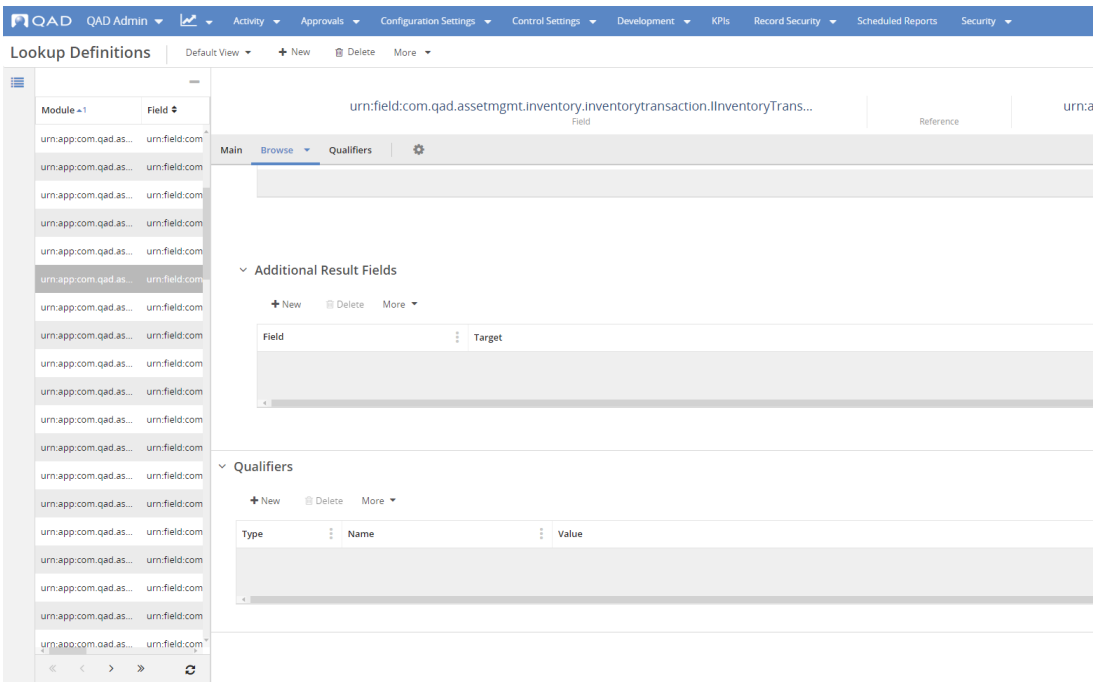
When you open Lookup Definitions, you first get a listing of the settings for the currently defined lookups, including:

- Module
- Field
- Reference
- Browse
- Result Field
- Search Field

Click **New** to create a new lookup or **Edit** to modify a lookup. Lookup Definitions includes Main, Qualifier, and Browse panels.

Example Screen Shots





UI Quick Reference

Main panel

A lookup definition is uniquely identified by Field, Reference, and Module values. The Main panel includes:

Field

Enter the URI value for a business component field, a view field, or a browse search field. A business component field takes the following format: urn:field:<business component interface namepace>:<TableName.FieldName

For example, for the Site Code field in Sales Order Line business entity: urn:field.qad.sales.salesorder.ISalesOrderLine:SalesOrderLine.SiteCode

Note: The lookup on this field only provides a browse of business component fields. View field URIs must be manually entered.

Reference

This is a free-form text field that can be used to differentiate between multiple lookup definitions for the same field value. For a single lookup definition for a field, this value can be left blank.

Module

The URI for the module that owns the lookup definition. For example, the Sales module is: urn:module:com.qad.sales

Note: Because the Field, Reference, and Module values specify the primary key for a lookup, they cannot be changed after the lookup is created.

Browse panel

On this panel, you specify the configuration of the browse data for the lookup definition.

Browse

The URI for the browse to be used for the lookup. This is in the format: urn:browse:<browse type>:<browse identifier>. For example, for standard browses, enter urn:browse:mfg:gp072 and for Financials browses, enter urn:browse:fin:BGL.SelectGL.

Result Field

Specifies the result field in the browse whose value will be assigned to the field from which the lookup is launched.

- For standard browses, the lookup result field is specified in the format <table.field>. For example: pt_mstr.pt_part. If this value is left blank, then the default from the browse definition is used.
- For Financials browses, the lookup result field must be specified and in the format <field>. For example: tcGLCode.

Search Field

If there is a value in the field from which the lookup is launched, a browse search condition will be created for the specified search field and the value.

- For standard browses, the lookup search field is specified in the format <table.field>. For example: pt_mstr.pt_part. If this value is left blank, then the default from the browse definition is used.
- For Financials browses, the lookup search field must be specified and in the format <table-name>.<field-name>. For example: tGL.GLCode.

Search Conditions panel

This panel allows the configuration of search conditions for the browse.

Enter search conditions for the browse, including:

- Field Name — the name of the search field to which the search condition applies.
- Operator — the operator to apply for this search condition. Currently, only Equals is supported.
- Value — the field value to use in the search condition. If Type is set to Field and the lookup is on a grid field, then the referenced field should be in the format: jsonTableName[0].jsonFieldName (for example: items[0].unitOfMeasure).
- Type — set to Literal or Field. A type of Literal means the actual value will be used. A type of Field means the value of the business component field referenced by Value will be used.

Additional Result Fields panel

This panel allows the configuration of additional result fields to populate with values from the selected browse record.

Field

The name of the browse display field.

- For standard browses, the field is specified in the format <table.field>. For example: pt_mstr.pt_part.
- For Financials browses, the field must be specified and in the format <field>. For example: tcGLCode.

Target

The identifier of the target field on the view. This is the value in the id attribute for that element in the document model. This can be found in Chrome by right-clicking the field and choosing Inspect.

For example, the value of the id attribute is SiteCodeAutoField here: <input autocomplete="off" type="text" class="formFldInputWithLookup k-input ng-pristine ng-invalid ng-invalid-required ng-touched" id="SiteCodeAutoField"

name="siteCode" ng-trim="false" ng-blur="ngBlur($event);" ng-focus="ngfocus($event);" size="8" ng-model="
ngData.items[0].siteCode" q-set-empty="" required="" tabindex="1539" focusableobjectuid="
viewfield_3ea2364d6dd94c61b74aa32887db1f33">

Qualifiers panel

The Qualifiers panel specifies any conditions that could determine whether to apply the lookup.

Qualifiers can be used to define conditional lookups. They are used to limit the set of lookups that can be used with a field and to select a particular lookup when there are multiples available. Each qualifier has a type, name, and value.

The Qualifier types include Field and View.

Field

A Field qualifier prevents a lookup from being used for a field unless another field in the view has a particular value. The Name identifies a field in the view and Value specifies the value that the field should hold in order for the lookup to be used.

View

A View qualifier limits the use of a lookup to within a specific view. For a View qualifier, the value is a view URI. The Name is not used. The view URI is typically the URI of the view containing the field from which the lookup will be launched.

Note: Multiple Qualifier records are combined together using a logical AND. For example if there are qualifiers (Type=Field,Name=X,Value=2), (Type=Field, Name=X,Value=3), (Type=Field, Name=Y, Value=4), this is evaluated as (X=2) AND (X=3) AND (Y=4). In a future release, qualifiers with the same Name will be combined using logical OR.

Extending Business Components

Introduction

This chapter explains the features in the platform that allows extension developers to extend the code flows, both on the back-end business logic as well as on the UI logic. For the business logic this is mainly focused on extending the CRUD flow, but also other external methods in the OOABL layer are available. If a standard App has made hook interfaces available these can also be implemented, and by doing so the business logic can be tweaked to meet the extension needs.

Business logic extension code is only supported in **OOABL**.

On the UI side the extension developer can use **TypeScript** to extend the UI triggers (create new ones, or extend existing ones).

Child pages:

Extending the Business Logic

Table of Contents:

Introduction

This section contains detailed information that can be used by an extension developer needing to extend the business logic flows for existing business components.

This is typically required for:

- more complex validations
- extra updates during the save of the business component data.

Channel Islands business component patterns and hooks

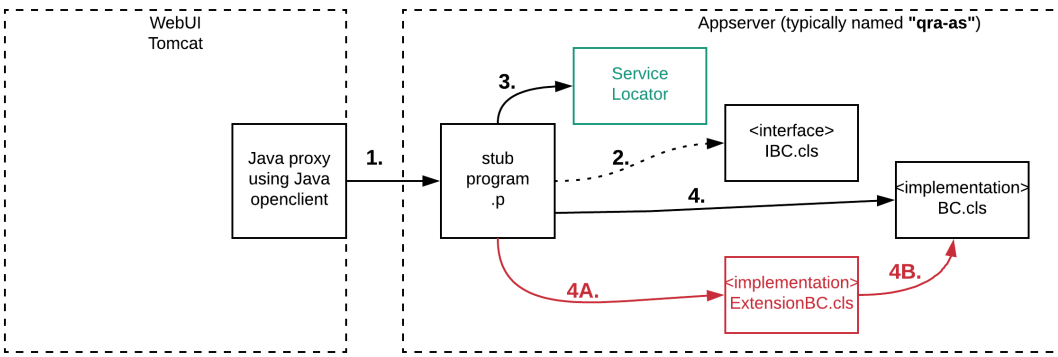
Most of the business components in the system can be accessed through a hybrid browse on the menu. For the correct workings of the screen, the system uses **fixed patterns** to work with the data. Both coded and platform (virtual) components use methods "Fetch", "Create", "Update", and "Delete" to perform the maintenance activities. All of these methods are exposed via interface classes on the backend OOABL side, which runs on the appserver.

Each of the methods that are exposed via the interface classes can be extended by adding extra code **before** and **after** the execution of the normal methods.

Executing the extension code

Technically the execution of the extension code is realized by injecting a new class that adds code to the method you want to extend. Injecting this new class is done via the **Service Locator**. This is the component that determines what OOABL implementation class is used for executing a call through an interface. By instructing the service locator that your special class needs to be executed instead of the original class, you can add any code you need.

The following overview makes this more clear:



Normal flow

1. The Web UI screen is always using a Java service to get the operation done. Whether it is a Fetch, a Create, or an Update. The call to these methods goes through a separate layer running a java proxy on Tomcat. The Java proxy is a generated set of programs that use the Progress Java OpenClient underneath to be able to call to the Appserver. This layer is also responsible for serializing / de-serializing datasets to JSON representations. Typically the data is passed as dataset-handles to the backend appserver side.

2. On the Progress Appserver there are generated Stub programs. These are simple .p files that represent the method that gets executed. They are responsible for accepting the dataset-handles and other parameters and call the correct implementation. These stub programs are using the interface of the business component to

understand the signature. Typical example is `com.qad.base.item.IItem`.

3. The stub program requests the Service Locator for an implementation of the BC interface. It is the responsibility of the Service Locator to return this. The way how that is done is by using the internal service registry that keeps a list of all interfaces and the possible implementation classes. Each entry has a key, and the implementation mapping with the empty key is the implementation that gets used by the stub. In a typical scenario, for the example of IItem, the service locator will return an instantiation of the class `com.qad.base.item.Item`.

4. Using the instance the stub got from the Service Locator, the right method is called and all lower level implementation code gets executed.

Extended flow

4A. The stub program gets an alternative implementation from the Service Locator. This is the Example 4: Add data extension to Countries to store Capital, Population, and Currency Code class that contains the extension code. This Extension code needs to implement the same interface than the standard implementation class. The system comes with a set of generated base classes that should be used for these extension implementations.

4B. The extension implementation class executes the extension code (before or after) and calls the standard implementation. This is just using the "super" calls via the normal OOABL inheritance.

Examples

Examples of OOABL code extensions can be found as part of other samples in these pages:

- Extend a standard coded business component: Example 3 - Extend the OOABL: Add extra validation to Sales Order
- Extend a platform (virtual) business component: Example 4: Add data extension to Countries to store Capital, Population, and Currency Code
- Extend a platform (virtual) business component - delete an instance: Add BL validation to check if one of the tax categories is used when deleting

Extending the UI

Introduction

This section provides more information for an extension developer for adding formula fields in business components and for adding UI triggers in the client side code. The platform provides editors for both cases. The formula field definition is a structured expression, the UI triggers are developed as TypeScript code in UI event handlers for the business component.

Extension capabilities

- Formulas
- UI Event Handlers

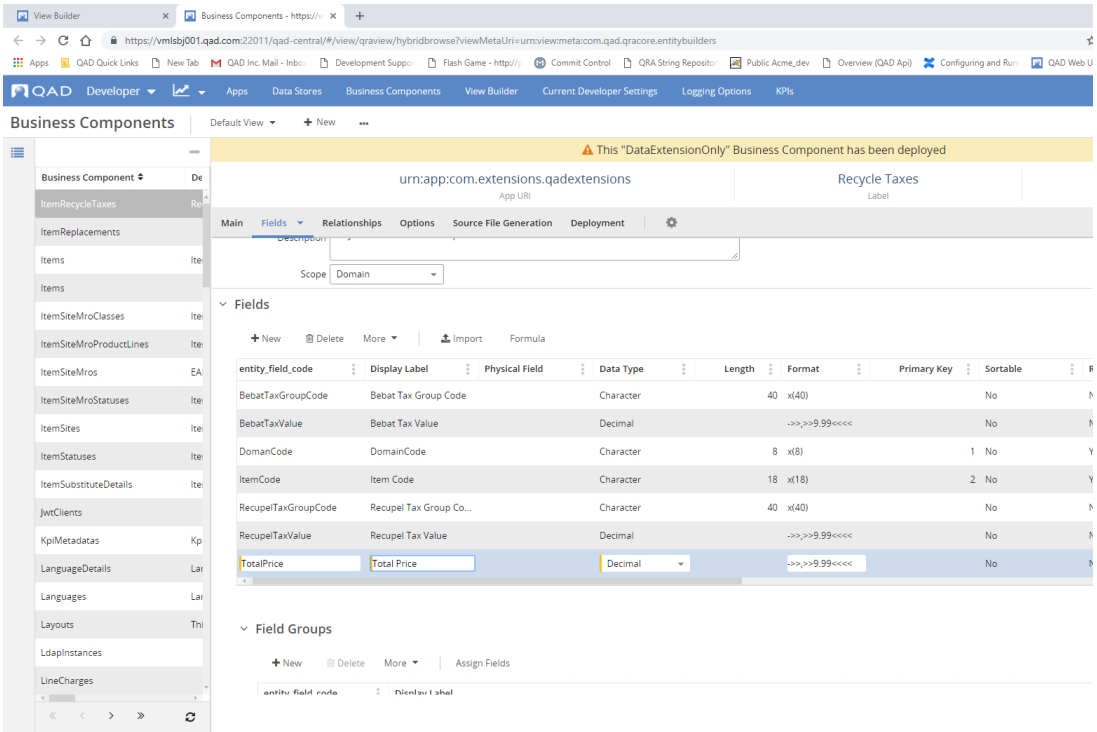
Formulas

Introduction

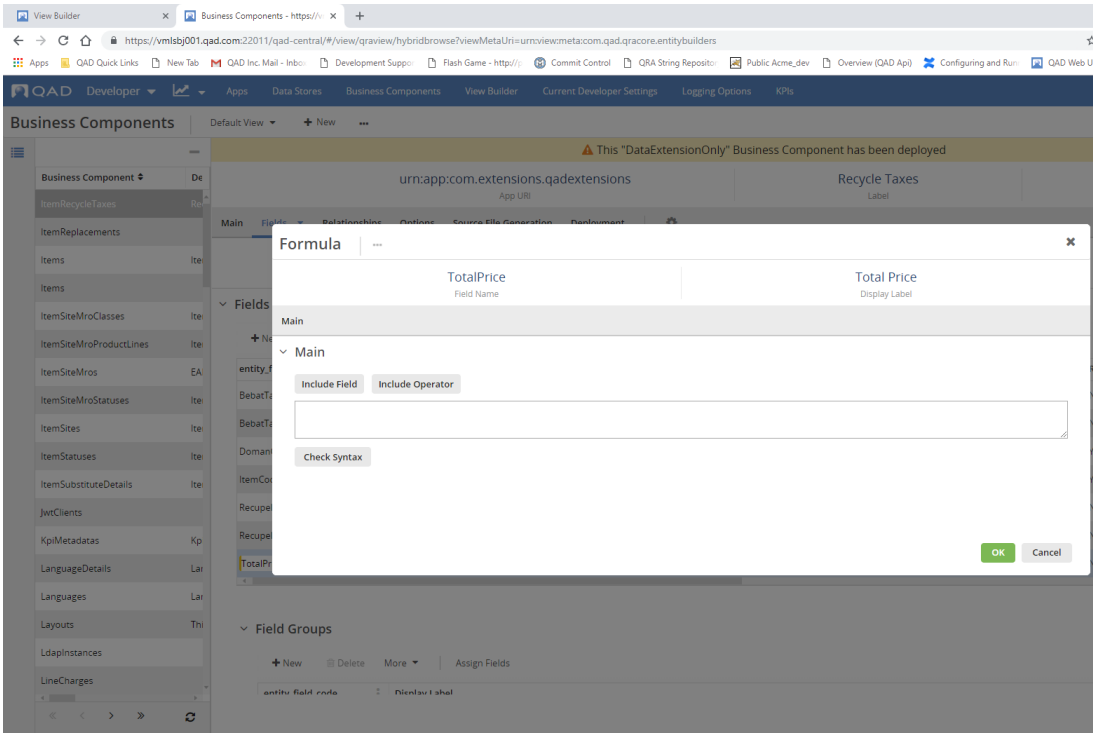
The business component builder provides the capability to add fields that are calculations based on data from other fields and records. We call these fields "formula fields" (see BC builder - formulas). This page describes how to extend a BC with a formula field.

Adding a formula to a BC

In order to add a formula field, open the BC builder and add a new field with the flag formula set to yes :

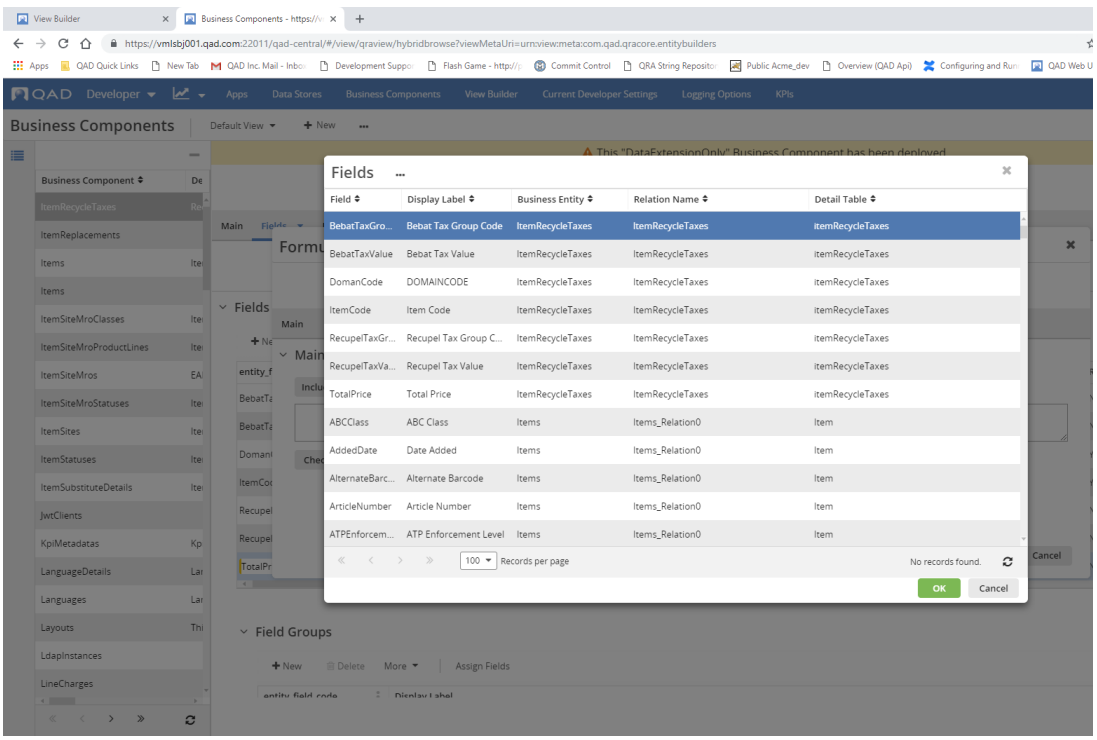


Note that you can do this on an already deployed Business Component. After adding the field press on the formula button on top of the grid :

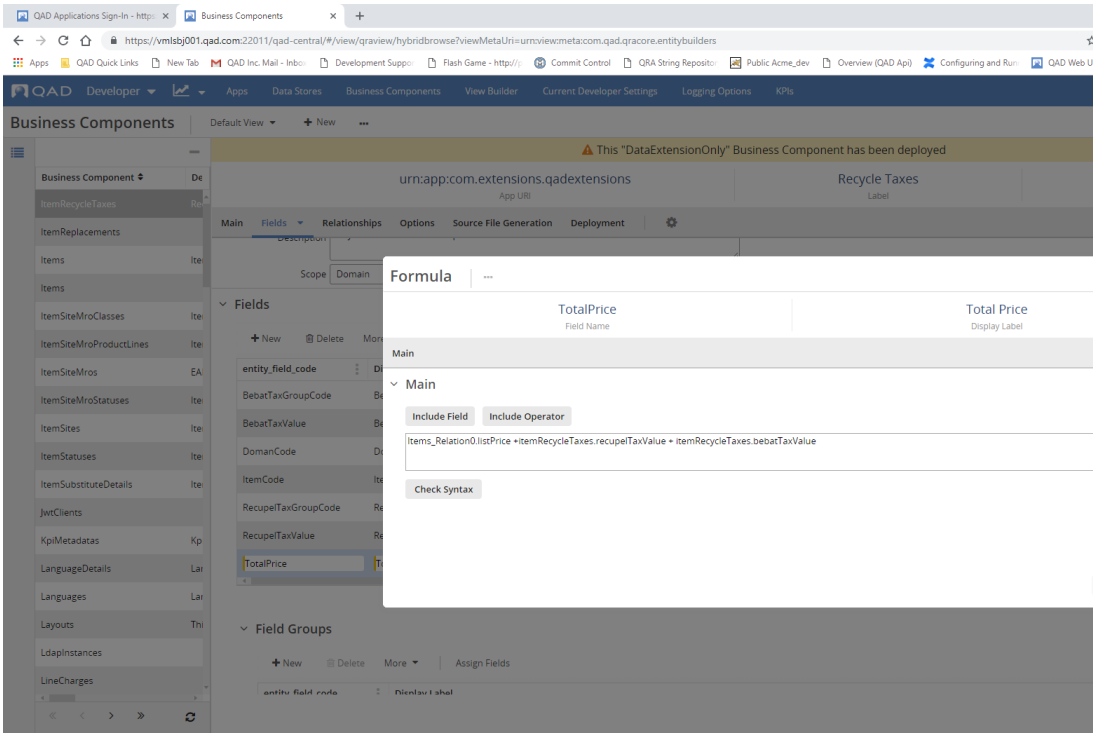


On the above screen you can start adding fields and operators to build a formula. You can also type the field names and operators without selecting them. The check syntax button always needs to be pressed before you are able to save the formula by pressing the ok button.

When selecting a field, not that you can take fields from the BC itself, but also from the extension relations :



Also note that the format of the field name is table.field for fields from the own BC table, and relation.field for fields from a relation :



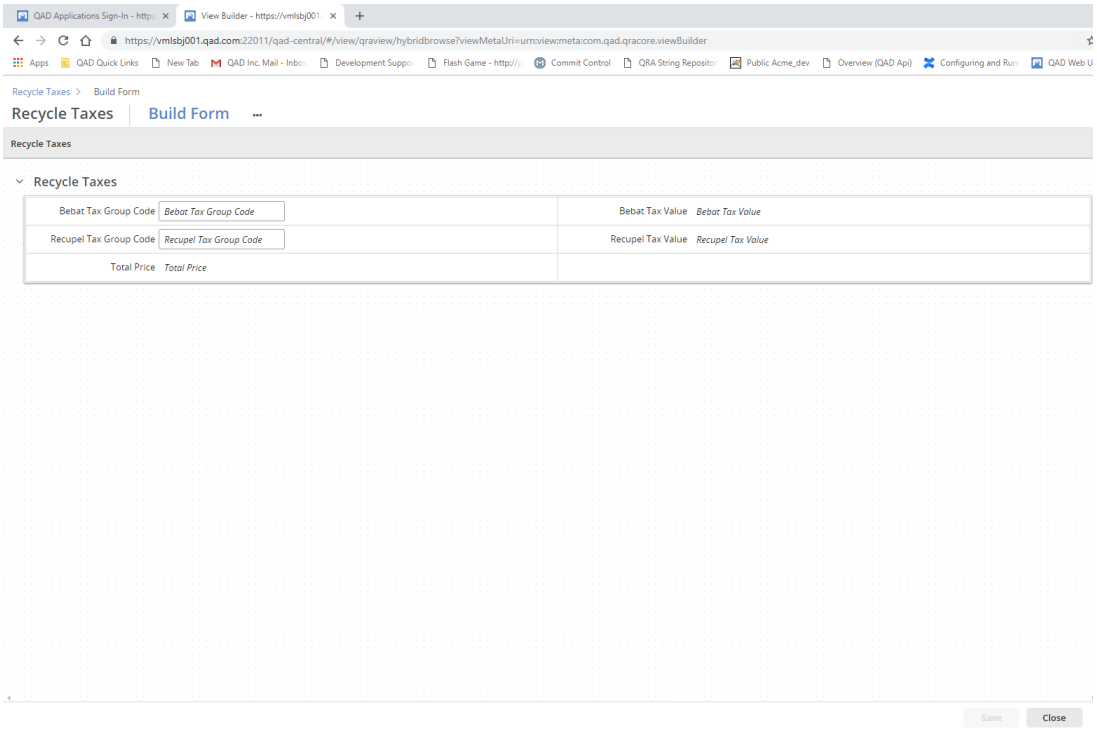
Aggregation functions

When the business component has a 1-N extension relationship , we can add aggregation formulas on child data. The syntax checker will automatically check if the field that is specified is a field from a 1-N relation. However, it is import to know that the notation for this kind of field is like an array notation with square braces. So, an aggregate function looks like this :

```
SUM([items.listPrice])
```

Adding the formula field to the UI

In order to add the formula field to the UI, open the View Builder and press the form button. The form builder will open and you will be able to drag and drop the formula field to the layout :



After saving the layout, you can see the field if you run the BC from the menu (it might be necessary to press the refresh button on the browses because some meta data is cached and only reloaded on page refresh).

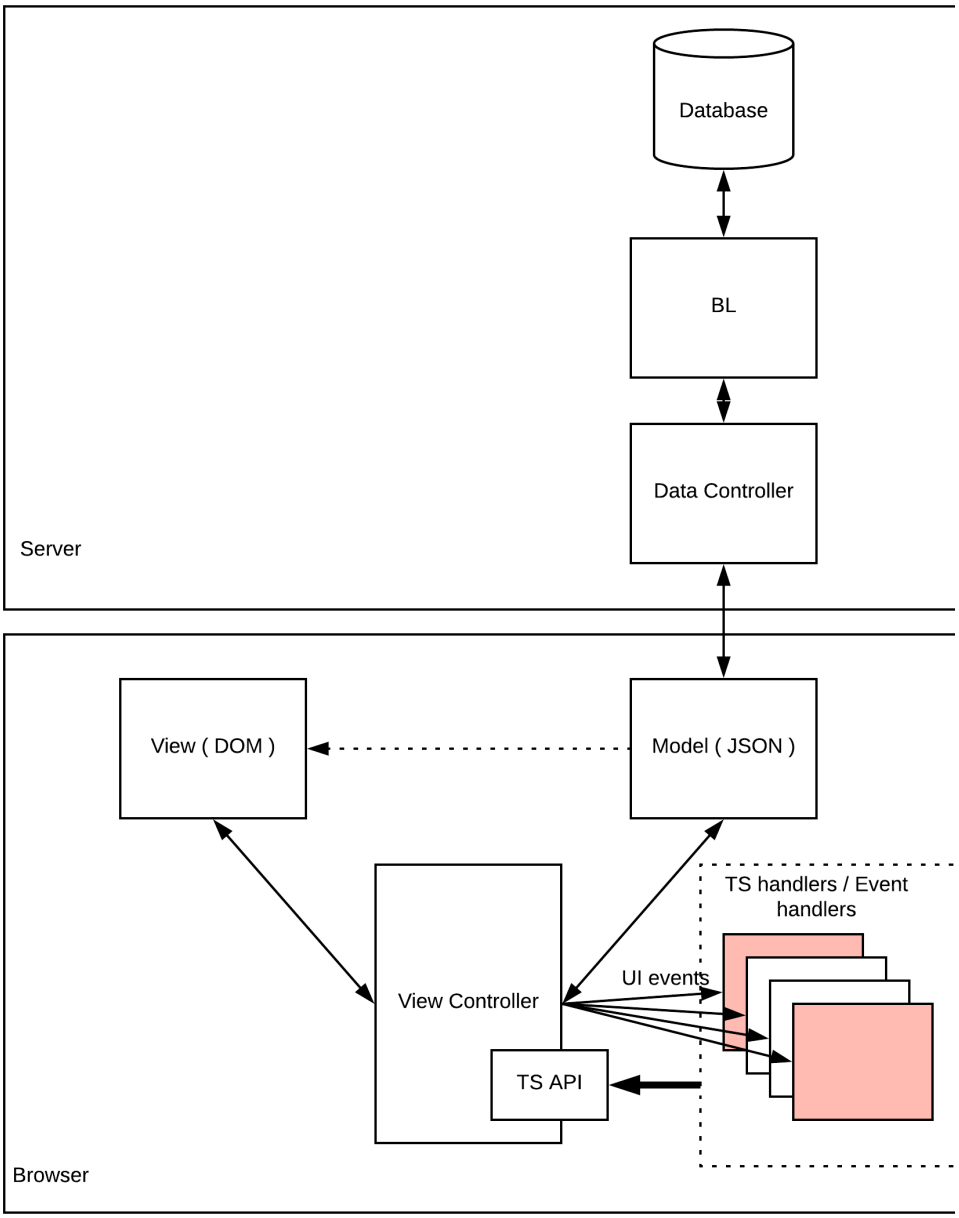
UI Event Handlers

Introduction

The view builder provides the capability to add event handlers. Event handlers are a way to customize the behavior of the UI. This page explains how this can be done.

What is an Event Handler?

An Event Handler is TypeScript code which is compiled into JavaScript and is saved to the database so that it can be executed on the UI for the corresponding UI of the Business Component. These event handlers run in combination with the application UI code which we call TS handlers. TS handlers are the same type of code as event handlers, but they are not stored in the database, instead they are part of the application code itself. In the following diagram, you can see how they fit in the MVC model on the client side :



In the diagram you can see that the controller runs event handlers (red) and TS handlers. Event handlers always run before or after the existing application code (TS handler).

Event handler timing

Because both TS handlers and event handlers act on (possibly the same) UI events, we can define a run time order. If an event handler is a PRE type, it runs before the existing TS handlers. A POST event handler runs after the existing TS handlers. Because a virtual business component has no existing TS handlers, we can't run an event handler before or after the UI logic. So, the Event handler defined in the same app as the virtual business component itself is the main UI logic packaged together with the business component. That's why we call that type of event handler a PRIMARY event handler. PRIMARY event handlers can only be created in the same app as the virtual business component itself. If that same business component is extended with UI events in another app then the one the business component belongs to, then you can specify PRE or POST again, and that determines if the event handler runs before or after the PRIMARY event handlers.

So, there are three types of Event Handlers:

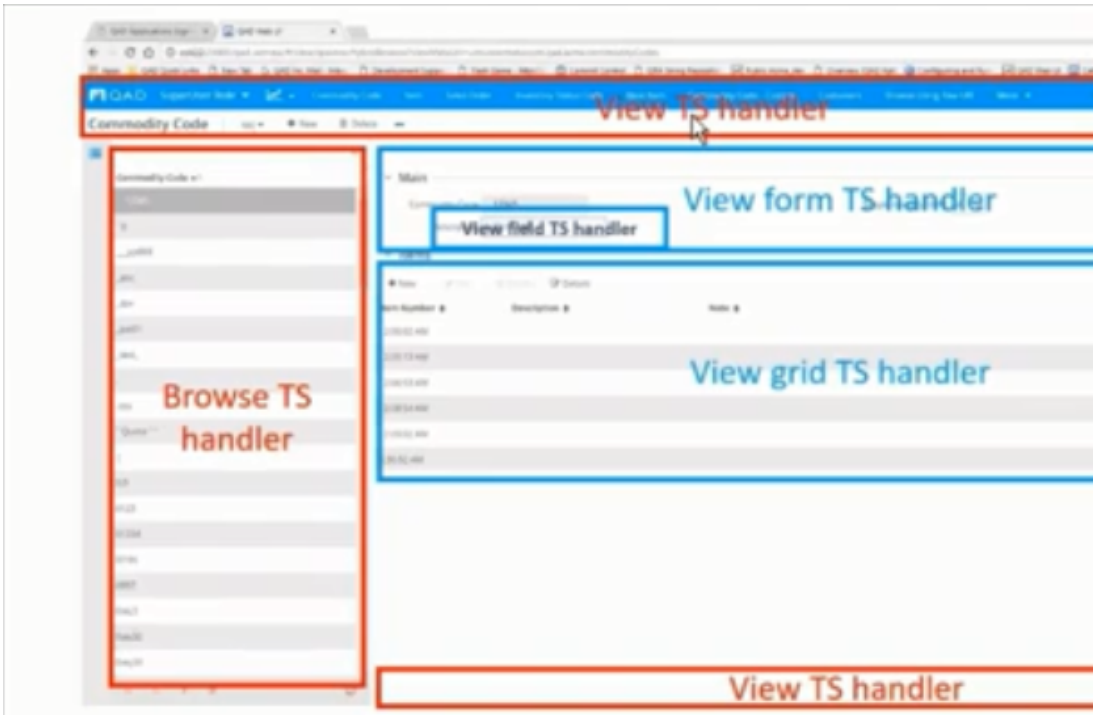
1. Primary – primary event handler for corresponding business component. Can be created only for virtual BC and has the same purpose as an event handler for coded BC. DB value: PRIMARY.
2. Pre – runs before Primary or coded event handler. DB value: BEFORE.

3. Post – runs after Primary or coded event handler. DB value: AFTER.

Since multiple Apps can bring in more event handlers, it is possible that the system has multiple stacked levels of this type of code, for example "App1-EventHandler-Pre","App2-EventHandler-Pre","StandardApp-TShandler","App3-EventHandler-Post","App1-EventHandler-Post".

Event handler class structure

Because the UI elements on the screen can vary depending on the layout of the UI, also the possible events to act on can vary. If there is for example a grid on the UI, then also grid events can be handled. To handle these different situations, different types of classes are available. The following picture shows the different types of TS handlers:



Note that we talk about TS handlers here. That is because event handlers are working the same as TS handlers, we use the same classes to develop them.

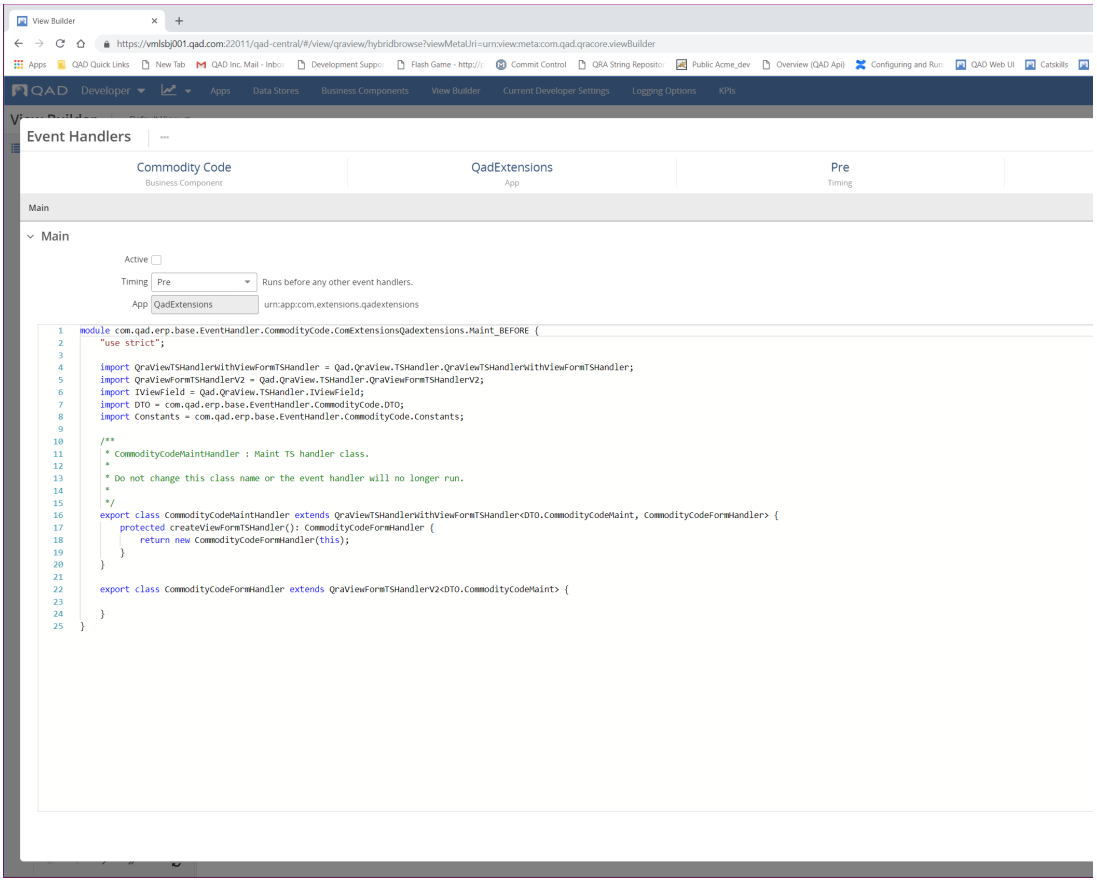
In the picture above, you see the different types of TS handlers. The different types handle UI events from different parts of the UI. This is the same for event handlers. However, UI event handlers can NOT act as a Browse TS handlers (so, only the TS handlers with "View" in their name are relevant for UI event handlers). This means that we can write event handlers that act as these types of TS handlers :

1. View TS handler : This is the main TS handler, and also the main UI event handler. This type is always needed as a starting point for event handlers. It can act on several more general UI events.
2. View form TS handler : This is an event handler that can act on all UI events related to form elements. Form elements are labels,input fields, group panels and buttons
3. View field TS handler : This is a very fine grained event handler, only handling on events coming from one specific field on the screen.
4. View grid TS handler : This type of event handler handles events coming from grids.

For all the above TS handler types, there are different Typescript base classes and interfaces that can be used to develop these types of event handlers. The following are important for event handlers :

1. QraViewTSHandlerWithViewFormTSHandler : This is a class that can be used to easily develop a view TS handler and easily connect to a view form TS handler. This class is auto generated for event handlers.
2. QraViewFormTSHandlerV2 : Class that can be used to develop a view form TS handler. This class is also auto generated for event handlers.
3. ViewGridTSHandler: Class that can be used to develop a view grid TS handler. Each grid on the form needs to have its own TS handler.

The above 3 are the ones that are used for event handlers, and when you add a new event handler, you will see that there are some standard classes generated that inherit from one of the above classes :



As you can see, there are already 2 classes generated, one class inheriting from QraViewTSHandlerWithViewFormTSHandler and one inheriting from QraViewFormTSHandlerV2. These 2 classes can now be extended with code to act on all events except grid events. For handling grid events, we need to add a class that inherits from ViewGridTSHandler (see Handling grid events).

Why base classes ?

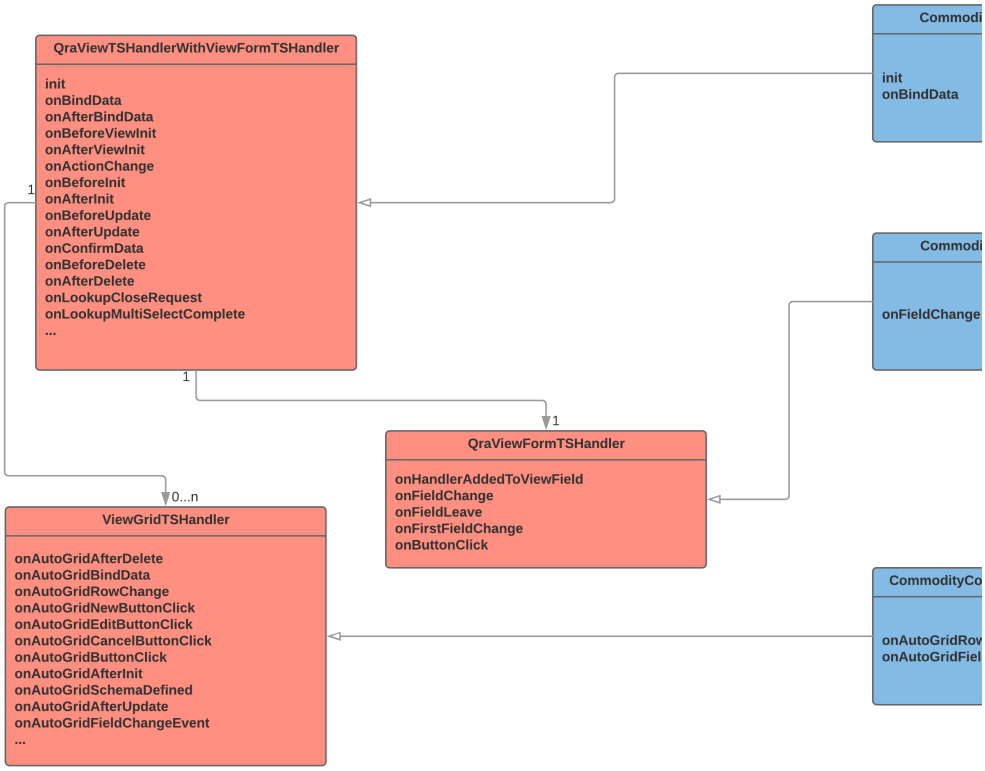
The reason why we need to inherit from base classes to develop an event handler is because the base classes do the following :

- They implement an interface that is called by the controller when a UI event is fired. All you need to do is write code that needs to run when an event happens is to override a base method, and add your own code.
- They have the necessary references and API methods to be able to call back from the event handler into the view controller to be able to manipulate the view. For example, the is a property ViewController on all base classes. This property is a reference to the view controller API.

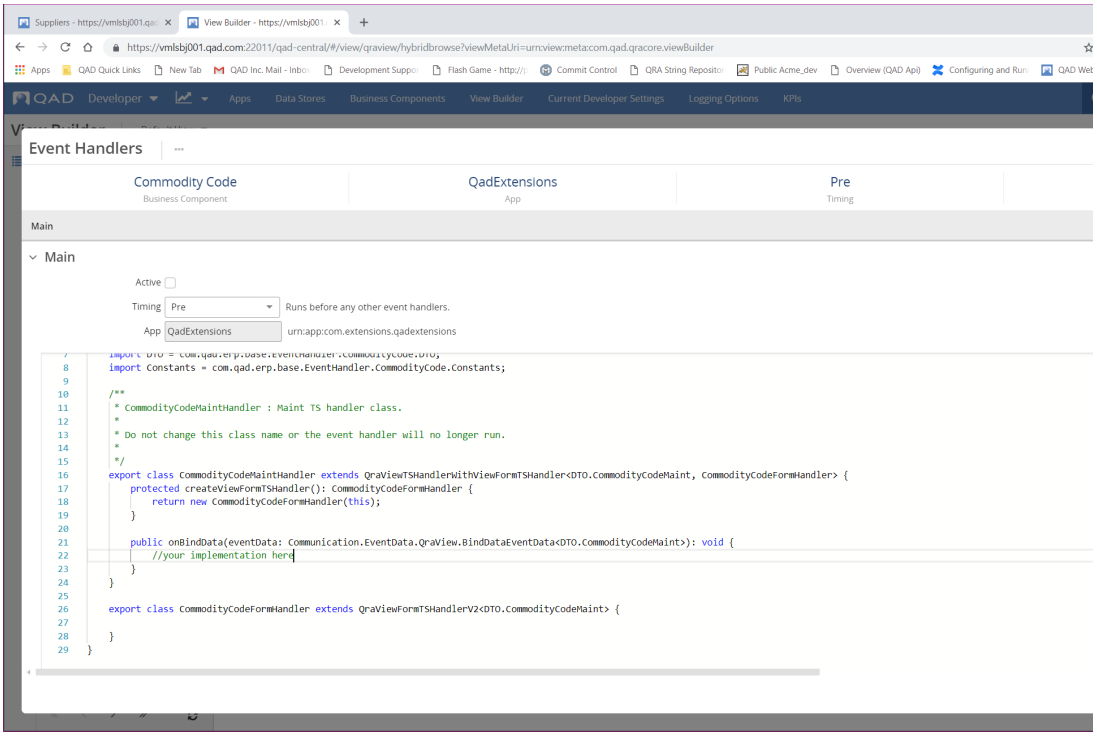
In a schematic view, an event handler looks like this (for example for the CommodityCode screen):

EVENT HANDLER CLASS DIAGRAM

Stef



As seen above, to write an event handler all you need to do is inherit from a class (partially automatically done), and override the necessary methods and implement them:

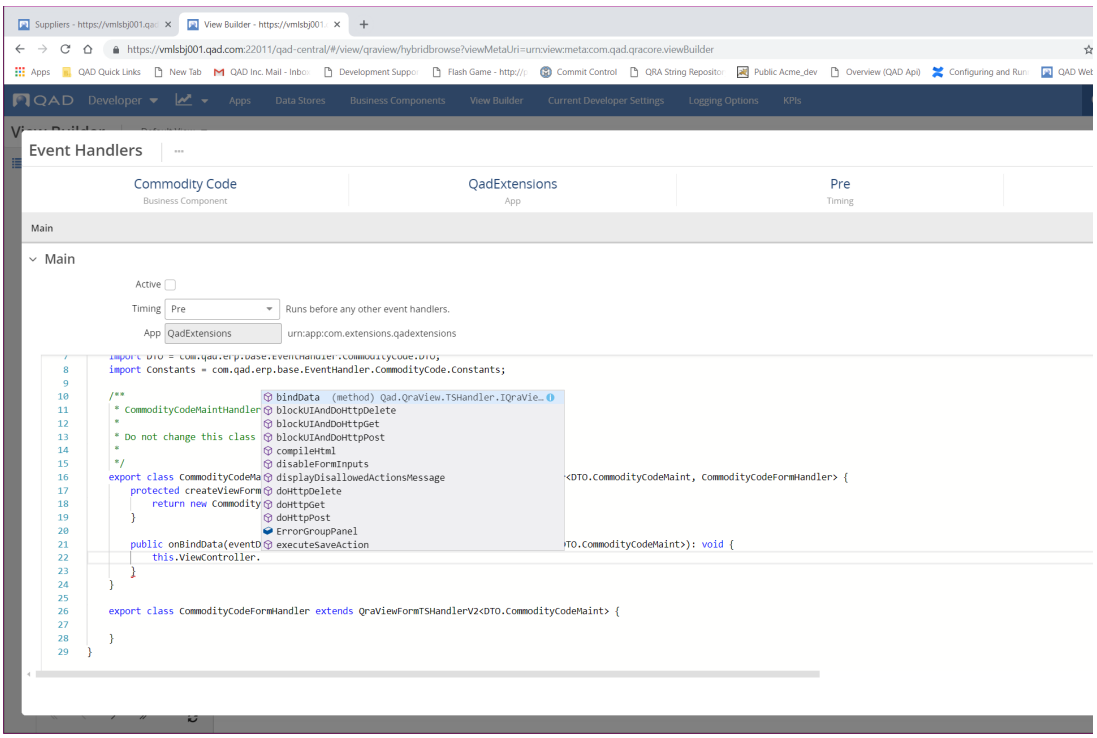


In order to add an event handler for a grid, a few lines of code need to be added. More detail can be found here : Handling grid events

A list of all available event functions can be found here : Event handlers API reference

API

Event handlers have an api available to manipulate the view, and to communicate with the server. The most important one is the interface to the view controller. Every event handler has a property ViewController :



That ViewController property is an interface with a lot methods to manipulate the view and to call other systems via http calls.

A few of the more important methods in this interface are :

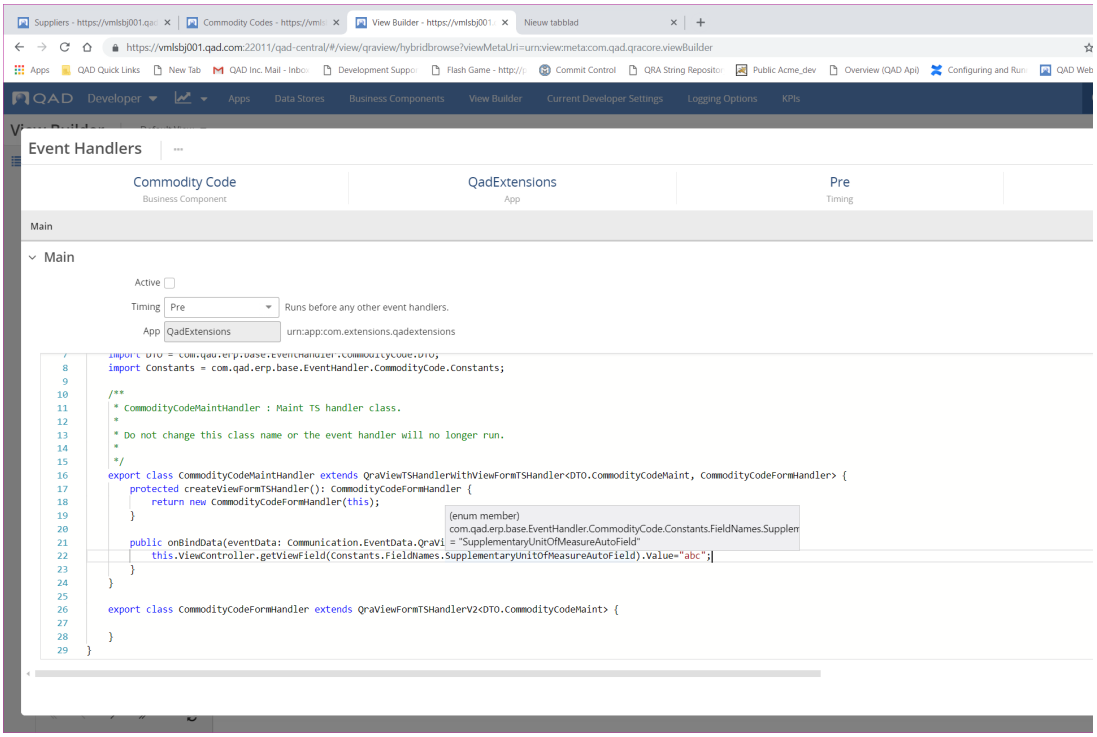
- getViewGrid
- getViewLabel
- getViewButton
- getViewField

The above methods all give you a reference back to an objects that represents a grid, a label, a button or a field on the screen. The returned object has it's own interface for manipulating it. These UI elements are reference by their id (can be found in the html). For fields and grids, there are constants with the name for the developer's convenience.

This is an example of the html of a field :

```
<input autocomplete="off" type="text" class="formFldInputWithLookup k-input  ng-pristine ng-valid
ng-valid-maxlength ng-touched" id="SupplementaryUnitOfMeasureAutoField" name="
supplementaryUnitOfMeasure" ng-trim="false" ng-blur="ngBlur($event);" ng-focus="ngfocus($event);"
size="2" maxlength="2" ng-model="ngData.commodityCodeMasters[0].supplementaryUnitOfMeasure"
tabindex="1508" focusableobjectuid="viewfield_6c48db142e7c4fc485961463a736fd3b" placeholder="">
```

So, in code it can be reference as following :

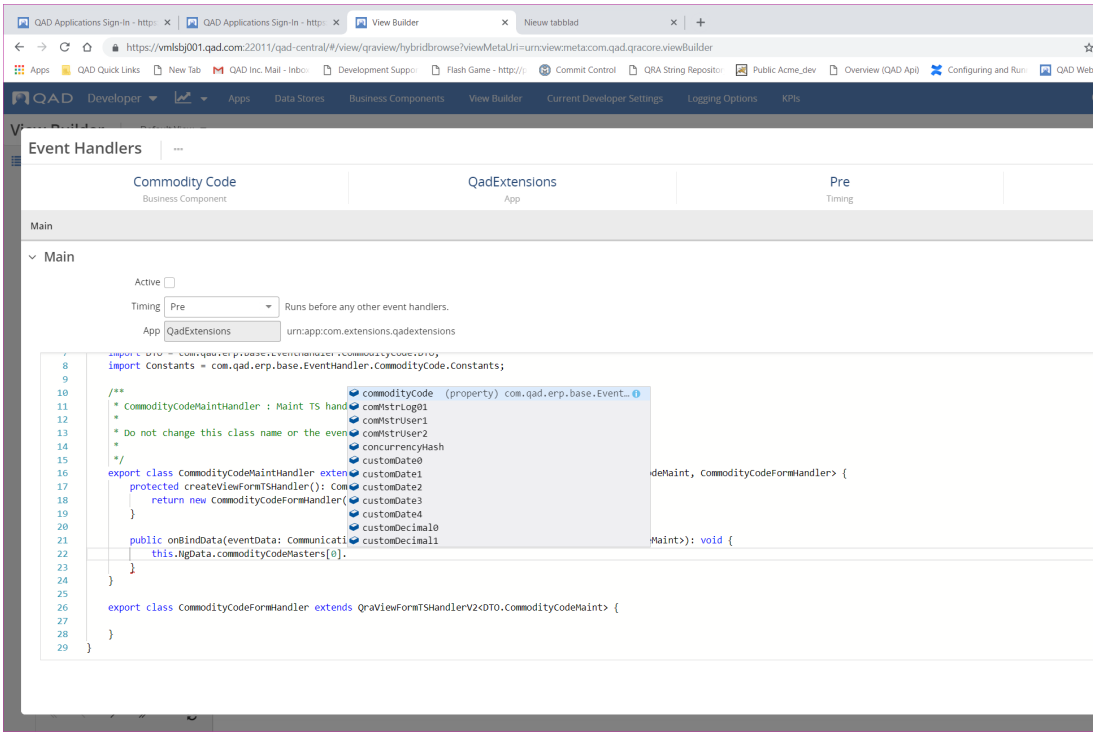


Note that we use the constant here, but we could have used the id in string too : this.ViewController.getViewField ("SupplementaryUnitOfMeasureAutoField"). ... .

A complete reference of the API can be found here : Event handlers API reference

DTO classes

The model part of the MVC model is a JSON object. This JSON object is bound to the view (bi-directional) and can be referenced by the NgData property :



As you can see above, the NgData object is correctly type, it holds 1 main record in this case, and all fields of that record are accessible on the first element of the array. All changes done on this object are immediately reflected on the UI (except for grid data, the grid holds it's own copy).

JQuery , AngularJS and KendoUI

The WebUI heavily makes use of :

- JQuery : Mostly used to manipulate the htlm and to search for html elements. JQuery can be used in the event handler to manipulate the html if the API is not sufficient. This needs to be done with care so that the standard UI logic is not broken.
- AngularJS : Internally used in the framework. Should be avoided to directly use in event handlers.
- KendoUI : Internally used in the framework to display UI controls. Can be used in event handlers, but needs to be done with care and might become incompatible with newer versions of the WebUI when new versions of KendoUI are intoduced.

References

The following pages provide more detail and references on how to develop UI event handlers :

- Handling grid events
- Event handlers API reference
- Event Handlers "How To"
- TypeScript Best Practices
- TypeScript recommended coding standards

TypeScript recommended coding standards

Console logging

Summary

Standard for excluding console statements

ID	TYPESCRIPT-0001
Version	1
Language	TypeScript
Status	PUBLISHED
Categories	Typescript standards
Applicability	All development activities

Description

Remove all console.log and console.dir statements from TypeScript files.

Rationale

Having these debug statements does not add any real benefit to the user/developer. Also, console.log may throw an exception in browsers that don't have support for it. Also, it will increase the size of the code base, and also increase the size of the final javascript file.

Examples

See Also

N/A

Constants

Summary

Standard for including constants or Alias's in Typescript files

ID	TYPESCRIPT-0003
Version	1
Language	TypeScript
Status	PUBLISHED
Categories	Typescript standards
Applicability	All development activities

Description

The use of hard coded string is prohibited within Typescript code. All hard code strings should be added to a constants file and exposed using the 'export const'.

The filename for this constants file should be <BusinessEntity>Constants.ts. This file should be in the same folder location as the Maint and ViewForm ts handlers.

All constants should be uppercase, and each word should be separated by an underscore.

Rationale

This technique will;

- Improve the readability for your code
- Centralize all constants - so if it changes later then its in one location
- Adds intellisense support for the IDE
- Improve quality of the code by minimizing the likelihood of typos

Examples

Right

Code

AutoFieldsConstants

```
export const CustomerPrepaymentControls = {
        CUSTOMER_CODE_AUTO_FIELD: "CustomerCodeAutoField",
        BUSINESS_RELATION_CODE_AUTO_FIELD: "BusinessRelationCodeAutoField",
        BUSINESS_RELATION_NAME_AUTO_FIELD: "BusinessRelationNameAutoField",
        INVOICE_DESCRIPTION_AUTO_FIELD: "InvoiceDescriptionAutoField",
        SUB_ACCOUNT_CODE_AUTO_FIELD: "SubAccountCodeAutoField",
        SUB_ACCOUNT_DESCRIPTION_AUTO_FIELD: "SubAccountDescriptionAutoField",
        COST_CENTRE_CODE_AUTO_FIELD: "CostCentreCodeAutoField",
        COST_CENTRE_DESCRIPTION_AUTO_FIELD: "CostCentreDescriptionAutoField",
        PROJECT_CODE_AUTO_FIELD: "ProjectCodeAutoField",
        PROJECT_DESCRIPTION_AUTO_FIELD: "ProjectDescriptionAutoField",
        AMOUNT_TC_AUTO_FIELD: "AmountTCAutoField",
        CURRENCY_CODE_AUTO_FIELD: "CurrencyCodeAutoField"
    };
```

```
If the screen has many AutoFields and Grids it is best to separate these constants into
their own const export statement.
```

Separate export consts

```
export const CustomerPaymentControls = { ....
export const CustomerPaymentStageGridFields = { ...
export const CustomerPaymentGridFields = { ....
```

Wrong

Code

Incorrect or Wrong Example

```
//Wrong capitalization of the constantname, should be uppercase
export const Create_Prepayment_Control_Name = {
        Customer_Code_Auto_Field: "CustomerCodeAutoField",

//Wrong use of a non constant to check on a certain field - should be a constant
case "BudgetGroupCodeAutoField":


//Wrong use of a non constant to check on a certain grid name - should be a constant
if (viewGrid.GridID == "CostCenterSafDefaultAutoGrid")

//Wrong use - this string should be translatable
let errorString = "Some Error string";
```

See Also

QRA Tools & Utilities (getViewAutoFieldsV2.py)

Never use var

Summary

ID	TYPESCRIPT-0004
Version	1
Language	TypeScript
Status	PUBLISHED
Categories	Typescript standards
Applicability	All development activities

Description

The use of 'var' should not be use and 'let' or 'const' should be used instead. If the variable declaration and initialization is done on the same line then 'const' should be used.

In summary:

- Use 'const' when possible
- Only use 'let' when the variable will be assigned a new value
- Never use 'var'

Rationale

In practice, there are a number of useful consequences of the difference in scope:

1. 'let' variables are only visible in their nearest enclosing block ({ ... }).
2. 'let' variables are only usable in lines of code that occur after the variable is declared (even though they are hoisted!).
3. 'let' variables may not be re-declared by a subsequent 'var' or 'let'.
4. Global 'let' variables are not added to the global window object.
5. 'let' variables are easy to use with closures (they do not cause race conditions).

Examples

Right

Code

Correct Code

```
let myString: String;


for (let i = 0; i < 5; i ++)
   ......


const myReadOnlyVar = "READONLY";
```

Wrong

Code

Correct Code

```
var myString: String;
```

```
for (var i = 0; i < 5; i ++)
   ......


var myReadOnlyVar = "READONLY";
```

See Also

Additional Explanation

Comments

Summary

Comments Standard for Typescript files

ID	TYPESCRIPT-0006
Version	1
Language	TypeScript
Status	PUBLISHED
Categories	Typescript standards
Applicability	All development activities

Description

For Typescript, we are going to follow the Java Source Code Comments standard (STD-0244)

STD-0244 Java Source Code Comments

Strict mode

Summary

Standard for using stirct mode in TypeScript files

ID	TYPESCRIPT-0007
Version	1
Language	TypeScript
Status	PUBLISHED
Categories	Typescript standards
Applicability	All development activities

Description

Always use the "use strict"; directive in your Typescript file.

Strict mode is a way to introduce better error-checking into your code. When you use strict mode, you cannot, for example, use implicitly declared variables, or assign a value to a read-only property, or add a property to an object that is not extensible.

Rationale

This will help to avoid weird TypeScript/JavaScript errors such as;

- It catches some common coding bloopers, throwing exceptions.
- It prevents, or throws errors, when relatively "unsafe" actions are taken (such as gaining access to the global object).
- It disables features that are confusing or poorly thought out.

Correct

"use strict";

```
module com.qad.erp.financials.costcenters {
      "use strict";
```

See Also

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode

Handling grid events

In order to handle grid events, we need to add a class to the event handler that can be used to handle the grid events. The class must inherit from `Qad.QraView.TSHandler.ViewGridTSHandlerV2` as in this example :

```
    export class ItemElectricalPlugTypesOneToManyAutoGridHandler extends Qad.QraView.TSHandler.
ViewGridTSHandlerV2<DTO.ItemMaint,DTO.Items_Relation1,DTO.ItemElectricalPlugTypes | kendo.data.
ObservableObject> {

    }
```

Note the different DTO classes :

- DTO.ItemMaint : this is the dto class for the maintenance object itself (e.g. items on the item maint screen)
- DTO.Items_Relation1 : this is the dto class for the grid data.
- DTO.ItemElectricalPlugTypes | kendo.data.ObservableObject : this is the type for one record in the grid.

After creating the above class, we need to create an instance and assign it to the grid object that it will handle events for:

```
    export class ItemMaintHandler extends QraViewTSHandlerWithViewFormTSHandler<DTO.ItemMaint,
ItemFormHandler> {
        public itemElectricalPlugTypesOneToManyAutoGridHandler:
ItemElectricalPlugTypesOneToManyAutoGridHandler;


        protected init() {
            this.ViewGridsToHandleList=["itemElectricalPlugTypesOneToManyAutoGrid"];
        }

        protected createViewFormTSHandler(): ItemFormHandler {
            let itemFormHandler=new ItemFormHandler(this);
            itemFormHandler.mainTSHandler=this;
            return itemFormHandler;
        }

        protected createViewGridTSHandler(viewGrid: Qad.QraView.TSHandler.IViewGrid<any, any,
kendo.data.ObservableObject>): Qad.QraView.TSHandler.ViewGridTSHandler<any, any, any, any> {
            if (viewGrid.GridID==="itemElectricalPlugTypesOneToManyAutoGrid") {
                this.itemElectricalPlugTypesOneToManyAutoGridHandler=new
ItemElectricalPlugTypesOneToManyAutoGridHandler(viewGrid,this);
                return this.itemElectricalPlugTypesOneToManyAutoGridHandler;
            }

        }
```

In the above example, the 2 parts that assign the grid handler to the grid are:

- in the init method : this.ViewGridsToHandleList=["itemElectricalPlugTypesOneToManyAutoGrid"]; : this will make sure the the createViewGridTSHandler is called for the itemElectricalPlugTypesOneToManyAutoGrid grid.
- createViewGridTSHandler method : important here is that the method return a new instance of ItemElectricalPlugTypesOneToManyAutoGridHandler.

After this you can start overriding grid event functions and implement them.

An example grid event handler can be found here : Add event handler to retrieve electrical plug info

Event Handlers "How To"

Get translated string

- Create a class "MessageKeys.ts".

```
module com.qad.erp.base.common {
    "use strict";
    export constMessageKeys = {
        ALTERNATE_CODE_NOT_UNIQUE: "mfg-857",
        ATTRIBUTE_ALREADY_EXISTS: "ATTRIBUTE_ALREADY_EXISTS"
    }

}
```

- Get translated string in the following fashion:

```
var myTranslatedString = this.ViewController.getLabel(MessageKeys.
ALTERNATE_CODE_NOT_UNIQUE);
```

Set field value from an http call

- Use view controller to do the ajax call, and also to set some field value:

```
public onFieldChange() {

let targetUrl = "api/erp/sites/...";

this.ViewController.blockUIAndDoHttpGet(targetUrl,
      (data: Qad.Common.DTO.DataResult, status, headers, config)=> {
                let receivedItem : DTO.Item = <DTO.Item> data.newValue;
                this.ViewController.getViewField("SomethingAutoField").Value =
receivedItem.newValue1;
                this.ViewController.getViewField("SomethingElseAutoField").Value
= receivedItem.newValue2;
      });

}
```

- Note that the above call is synchronous and will block the UI. A non-blocking call can be made with `this.ViewController.doHttpGet`, this however needs to be done with care. Don't use asynchronous calls for editable fields or fields that hold data needed for saving (otherwise the value from the async http call can arrive after the http post call for saving).
- This pattern is typically used for fetching the description of a code field. This can however be done automatically with Description fetching.

Hide / Show fields

- Hide / Show fields : use "IsVisible" properties :

```
this.ViewController.getViewField("SomethingAutoField").IsVisible = false;
```

Checking data for null values or empty strings (FD 19.1 or later)

The following static methods are available for checking null values or empty string :

in module Qad.Common.Util :

- Class StringUtils:
  - isNullOrEmpty: Check whether a string value is null, undefined or an empty string.
  - isNullOrWhiteSpace: Check whether a string value is null, undefined, an empty string (length 0) or a string with only white spaces.
- Class ValidationUtils:
  - isUndefinedOrNull : Check whether a value is undefined or null.
  - isNullOrEmptyString : Check whether a value is null, undefined or an empty string (length 0). Will also return false if the value is not a string.
  - isNullOrWhiteSpaceString : Check whether a value is null, undefined, an empty string (length 0) or a string with only white spaces. Will also return false if the value is not a string.

example:

```
    import StringUtils = Qad.Common.Util.StringUtils;

    /**
     * Third Main Commodity code maintenance screen TS handler
     */
    export class CommodityCodeMaintTSHandler3 extends QraViewTSHandler<DTO.
CommodityCodeMaintNgData>
    {

        public valideData() {
            if (StringUtils.isNullOrEmpty(this.NgData.commodityCodeMasters[0].domainCode)) {
                // do something
            }
        }
```

Event handlers API reference

This page contains a list of all available event functions that can be overridden to implement an event handler.

Overridable event functions

QraViewTSHandlerWithViewFormTSHandler<TNgData,TQraViewFormTSHandler extends QraViewFormTSHandler<TNgData>>

Generics types:

- TNgData : class that represents structure of ngData in view controller.
- TQraViewFormTSHandler : view form TS handler class

View events :

Method	Event Data Properties	Description
init()		Called right after the TS handler was instantiated.
onBindData (eventData: QraView.BindDataEventData<TNgData>)	**eventData.data** : an object (type TNgData) containing the data that was bound	Called when the view screen binds new data to its model/fields. Fired after the new data is bound.
onBeforeUpdate		

(eventData: QraView.BeforeUpdateEventData)	**eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing **eventData.autoSave** : boolean which indicates if the update was triggered directly by a user clicking the form save button (false) or via an auto save (true).	Called before an update submit is attempted (for either a create or update operation). TODO: need to refactor the publishing logic since it will happen too generically from button events other than update.
onAfterUpdate (eventData: QraView.AfterUpdateEventData)	**eventData.success** : boolean indicating the success of the update submit; is true only if the submit succeeded	Called after an update submit has finished, whether the update succeeded or not. Fired after data binding in the view. TODO: need to refactor the publishing logic since it will happen too generically from button events other than update.
onCancelChange (eventData: QraView.CancelChangesEventData)		Called when the user cancel's the view.
onBeforeDelete (eventData: QraView.BeforeDeleteEventData, processEvent: (processIt?: boolean) => void)	**eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	Called before a delete submit is attempted. TODO: need to refactor the publishing logic since it will happen too generically from button events other than update.
onAfterDelete (eventData: QraView.AfterDeleteEventData)	**eventData.success** : boolean indicating the success of the delete submit; is true only if the submit succeeded	Called after a delete submit has finished, whether the delete succeeded or not. Fired after data binding in the view. TODO: need to refactor the publishing logic since it will happen too generically from button events other than update.
onBeforeInit (eventData: QraView.BeforeInitEventData)	none	Called before the initialize url is called when "New" is clicked.
onBeforeInit (eventData: QraView.BeforeInitEventDataV2)	eventData.initUrl: The url that will be used for the initialize call. This can be modified by the TS handler.	Available in FD22.2. Called before the initialize url is called when "New" is clicked. An example of a TS handler that modifies the eventData.initUrl can be found here, search for "onBeforeInit".
onAfterInit (eventData: QraView.BeforeInitEventData)	**eventData.success** : boolean indicating the success of the initialize; is true only if the initialize succeeded. eventData.data: the data returned by the initialize call.	Called after the initialize url call has been made when "New" is clicked.
onButtonClick (eventData: Qad.Common.Service.Communication.EventData.Button.ButtonClickEventData, processEvent: (processIt?: boolean) => void)	null	Called when a button defined in the view meta data is clicked.
onBeforeToolbarInitialized (eventData: QraView.BeforeToolbarInitializedEventData)	**eventData.toolbarData**: toolbar configuration data	Called prior to the view toolbar being initialized. This allows the data in eventData.toolbarData to be customized.
onToolbarInitialized (eventData: QraView.ToolbarInitializedEvent	null	Called when the toolbar is initialized.

Data)		
onActionChange (eventData: QraView. ActionChangeEventDat a)	**eventData.oldAction**: null (when this is the first action), "CREATE" or "UPDATE" **eventData.newAction**: "CREATE" or "UPDATE"	Called when the view switches from "CREATE" to "UPDATE" or "UPDATE" to "CREATE" or when the first first view comes up. For example, when the "New" button is clicked the action switches to "CREATE" or after "Save" is clicked on a "New" record the action switches to "UPDATE".
onConfirmData (eventData: QraView. ConfirmDataEventData <TNgData>, processEvent: (processIt?: boolean) => void)	**eventData.eventData. responseData.data. <nameoftheconfirmdataset>** : this is the extended data (e. g. confirmation data) whose format will vary according to the entity requirements. **eventData.action**: "update", "create", or "delete" **eventData.dataConfirmed()**: The function handle to be called when the data has been successfully confirmed **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	Called when a create, update, or delete is processed (i.e. via click of the 'Save' button).
onBeforeViewInit (eventData: QraView. BeforeViewInitEventDat a)		Called before the view starts initializing.
onAfterViewInit (eventData: QraView. AfterViewInitEventData)		Called after the view initialized.
onAutoGridBeforeInit (eventData: AutoGrid. BeforeInitEventData)	**eventData.gridId** = the ID of the auto grid **eventData.dataWiring** = the data wiring object **eventData.viewSchema** = the view schema object	Called prior to the initialization of the auto grid.
onBeforeRequery (eventData: QraView. RequeryEventData)		Called when the view re-queries it's data.
onBeforeSendData (eventData: QraView. SendDataEventData, processEvent: (processIt?: boolean) => void)	**eventData.url**= url of the request **eventData.requestMethod** = the request method (get /post/put/delete) **eventData.buttonProperties** = button properties if applicable **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	Called prior to sending data to the server.
onGetKeyData (eventData: QraView. GetKeyDataEventData)	**eventData.keyValuePairs**= k ey/value pair object with key data	Called when getKeyData is called on the Qra view controller.
onAttachmentsDataBou nd(eventData: QraView.	**TODO: complete** **eventData.keyset**=	Called when the attachments panel it's data is bound.

AttachmentsDataBound EventData)	**eventData.action**= **eventData.complete**=	
onBeforeAttachmentDataBound(eventData: QraView. BeforeAttachmentsDataBoundEventData)	**eventData.dataRow**= binding data	Called prior to binding the attachments panel it's data.
onActivityFeedDataBound(eventData: QraView. ActivityFeedDataBoundEventData<TNgData>)	**eventData.data**= view data	Called when the view it's data is bound, and the activity feed panel can start loading.

Button events (toolbar buttons):

Event Type	Event Data Properties	Description
onButtonClick (eventData: Button. ButtonClickEventData,processEvent: (processIt?: boolean) => void)	**eventData.buttonProperties** : object containing all of the properties that the button was configured with, including its action URL **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing **eventData.autoSave** : (only for the "Save" button) boolean which indicates if the save was triggered directly by a user clicking the form save button (false)or via an auto save (true).	Published when a button is clicked (or otherwise invoked, e.g. by a keypress on the button). Fired before the action for the button is taken, with the ability for the subscriber to cancel this action if desired. Note: This event is fired from Browse tool buttons and maintenance screen buttons. Note: Not fired for Auto Grid buttons. Use the viewEvents.autoGridButtonClick event for those.

Group panel events :

Event Type	Event Data Properties	Description
onGroupPanelSelectorClick(groupPanel: IGroupPanel, eventData: QraView. GroupPanelSelectorClickEventData)	**eventData.groupPanelId** : ID of group panel that was selected **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	Called when a group panel selector button is invoked to navigate to a desired group panel.
onGroupPanelToggleExpand(groupPanel: IGroupPanel, eventData: QraView. GroupPanelToggleExpandEventData)	**eventData.groupPanelId** : ID of group panel that was selected **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing **eventData.expand**: integer equal to 0 if the panel is being un-expanded, 1 if the panel is being expanded	Called when a group panel's expansion toggle control is clicked.
onErrorGroupPanelToggleExpand (errorGroupPanel: IErrorGroupPanel, eventData: Qad.Common.Service. Communication.EventData.QraView. GroupPanelToggleExpandEventData)	**eventData.groupPanelId** : ID of group panel that was selected **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	Called when the error group panel's expansion toggle control is clicked.

	eventData.expand: integer equal to 0 if the panel is being un-expanded, 1 if the panel is being expanded	
onGroupPanelsConfigOpen (groupPanelNavigator: IGroupPanelNavigator,eventData: QraView. GroupPanelConfigOpenEventData)	**eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	Called when a group panel configuration dialog is opened.
onGroupPanelsConfigApply (groupPanelNavigator: IGroupPanelNavigator,eventData: Qad. Common.Service.Communication. EventData.QraView. GroupPanelConfigOpenEventData)	**eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	Called when a group panel configuration dialog's Apply action is invoked.
onGroupPanelsInit(groupPanelNavigator: IGroupPanelNavigator,eventData: Qad. Common.Service.Communication. EventData.QraView. GroupPanelsInitEventData)	null	Called once when a screen is first launched and has initialized the group panels.

Hybrid view events :

Event Type	Event Data Properties	Description
onHybridBrowseViewStateChange(eventData: EventData. QraHybridBrowse. ViewStateChangeEventData)	**eventData.viewState** : an integer value representing the new state of the browse (or maintenance screen) view: 0: Contracted (hybrid) view 1: Browse view 2: Maint View	Called from hybrid browses when the browse or maintenance view state widgets are clicked to expand or contract the browse view.

QraViewFormTSHandler<TNgData>

Generics types:

- TNgData : class that represents structure of ngData in view controller.

View events :

Method	Event Data Properties	Description
onButtonClick (viewButton: IViewButton, eventData: Button. ButtonClickEventData)	null	Called when a button defined in the view meta data is clicked or when a button defined using addBottomButton() is clicked.
onFirstFieldChange(viewField: IViewField<any>,eventData: EventData. QraView.FirstFieldChangeEventData)	**eventData. fieldName**: the ID of the changed field **eventData. fieldValue**: the new value of the changed field **eventData. fieldValueOrig**: the original value of the changed field	Called when the first field in the form is changed in a standalone (non-data-grid) field. Fired as soon as the UI focus is moved away from the changed field. Note: only fired in response to field changes in the UI (e.g. a user entering data), not when fields are bound to data from the model.
onFieldChange(viewField: IViewField<any>,eventData: EventData. QraView.FieldChangeEventData,		

processEvent: (processIt?: boolean) => void)	**eventData. fieldName**: the ID of the changed field **eventData. fieldValue**: the new value of the changed field **eventData. fieldValueOrig**: the original value of the changed field	Called when a field value is changed in a standalone (non-data-grid) field. Fired as soon as the UI focus is moved away from the changed field. Note: only fired in response to field changes in the UI (e.g. a user entering data), not when fields are bound to data from the model. Note: in order to cancel the fieldChange event (and restore the prior value to the field) it is necessary for the subscriber to call "processEvent(false)".
onFieldLeave(viewField: IViewField<any>,eventData: EventData. QraView.FieldLeaveEventData)	**eventData. fieldName**: the ID of the field **eventData. fieldValue**: the current value of the field	Called when a field loses focus, regardless of any change to the field value.

ViewGridTSHandler<TNgData,TGridNgData, TGridRecord, TGridRecordObservableRecord extends kendo.data.ObservableObject>

Generics types:

- TNgData : interface that represents structure of ngData in view controller.
- TGridNgData : interface that represents structure of ngData in the grid (same structure as json that's being transferd)
- .TGridRecord : interface that represents on grid row.
- TGridRecordObservableRecord : merged TGridRecord interface and kendo.data.ObservableObject interface.

Grid events:

Event Type	Event Data Properties	Description
onAutoGridBindData(eventData: EventData.AutoGrid. BindDataEventData<TGridRecord>)	**eventData. gridId:** the ID of the auto grid **eventData. dataRows** : an array containing the data rows that were bound	Called when the viev new data is bound. TODO: Commodity (Update maint screer just the correct detai
onAutoGridRowChange(eventData: EventData.AutoGrid. RowChangeEventData<TGridRecordObservableRecord>)	**eventData.gri dId**: the ID of the auto grid **eventData. dataRow**: selected data row	Called whenever the TODO: Need to supp
onAutoGridFieldChangeEvent(eventData: EventData.AutoGrid. FieldChangeEventData<TGridRecordObservableRecord> , processEvent: (processIt?: boolean) => void)	**eventData. fieldName**: the name of the changed field **eventData. fieldValue**: the new value of the changed field	Called when an auto focus is moved away Note: only fired in re data), not when field Note: in order to can the field) it is necess TODO: There is a to thrown when trying t TypeError: Cannot re

	eventData.gridId: the ID of the auto grid **eventData.dataRow**: the data of the row that the field is in.	
onAutoGridFieldLeaveEvent(eventData: EventData.AutoGrid.FieldLeaveEventData)	**eventData.fieldName**: the ID of the field **eventData.fieldValue**: the current value of the field	Called when an auto field value.
onAutoGridNewButtonClick(eventData: EventData.AutoGrid.NewButtonClickEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row **eventData.previousSelectedDataRow**: previously selected data row	Called when the New Edit mode. TODO: Need to supp

onAutoGridBeforeEdit(eventData: EventData.AutoGrid.BeforeEditEventData<TGridRecordObservableRecord>, processEvent: (processIt?: boolean) => void)	**eventData.gridId**: the ID of the auto grid
	
	eventData.dataRow: selected data row
	
	eventData.eventProcessed : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing
		(Available as of FD8
		
		Called before the row following example us or if the edit should b
		
		`public onAutoGr Service.Communi BeforeEditEvent processEvent: ( if(this.NgD == "API-6876")`
		
		`{ eventDa var tes`
		
		`closeButtonText`
		
		`selection back`
		
		`(eventData.data`
		
		`} this.l }` `}`

onAutoGridAfterEdit(eventData: EventData.AutoGrid.AfterEditEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row **eventData.returnData**: the data returned from the initialize call which will include supplementaryMessages if any exist. This property is only valid for a new grid line. **eventData.action**: "UPDATE" or "CREATE" indicating whether this event is the result of editing an existing row or going into edit on a new row.	This event fires only mode on an existing row it fires after the i dataRow) to the row
onAutoGridEditButtonClick(eventData: EventData.AutoGrid.ButtonClickEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row	Called when the Edit row into Edit mode. TODO: Need to supp
onAutoGridCancelButtonClick(eventData: EventData.AutoGrid.CancelButtonClickEventData)	**eventData.gridId**: the ID of the auto grid	Called when the Car
onAutoGridBeforeAddNew(eventData: EventData.AutoGrid.BeforeAddNewEventData<TGridRecordObservableRecord>, processEvent: (processIt?: boolean) => void)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: currently seleced data row **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken	Called before a new

	control of the event processing	
onAutoGridBeforeUpdate(eventData: EventData.AutoGrid.BeforUpdateEventData<TGridRecordObservableRecord>, processEvent: (processIt?: boolean) => void)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row **eventData.source:** indicates if the event was triggered by the "Next Line" button or the "Done Lines" button. The value will be set to "nextLine" or "doneLines". (available as of FD6 patch 2) **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	Called before an aut update operation). TODO: Need to supp
onAutoGridAfterUpdate(eventData: EventData.AutoGrid.AfterUpdateEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row **eventData.source:** indicates if the event was triggered by the "Next Line" button or the "Done Lines" button. The value will be set to "nextLine" or "doneLines". (available as of FD6 patch 2) **eventData.success:** boolean indicating the success of the update	Called after an auto succeeded or not. Fi TODO: Need to supp

onAutoGridBeforeDelete(eventData: EventData.AutoGrid.BeforeDeleteEventData<TGridRecordObservableRecord>, processEvent: (processIt?: boolean) => void)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	Called before an aut TODO: Need to supp
onAutoGridAfterDelete(eventData: EventData.AutoGrid.AfterDeleteEventData)	**eventData.gridId**: the ID of the auto grid TODO: Add a success boolean once we find a way to determine that in the code	Called after an auto succeeded or not. Fi
onAutoGridDetailsLinkClick(eventData: EventData.AutoGrid.DetailsLinkClickEventData<TGridRecordObservableRecord>, processEvent: (processIt?: boolean) => void)	**eventData.gridId**: the ID of the auto grid **eventData.linkUrl**: URL of the default navigation link **eventData.linkLabel**: translated label for the default navigation link **eventData.dataRow**: selected data row **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	Called when the auto attempted. To override the defa code, and be sure to the default navigatio
onAutoGridDetailsLinkClose(eventData: EventData.AutoGrid.DetailsLinkCloseEventData<TGridRecordObservableRecord>, processEvent: (processIt?: boolean) => void)	**eventData.gridId**: the ID of the auto grid	Called when the auto The default behavior refresh. If for some r

	eventData. linkUrl: URL of the default navigation link **eventData. linkLabel**: translated label for the default navigation link **eventData. dataRow**: selected data row **eventData. childState**: contains a reference to the child view's QraView JS controller; used to pass child state back to the parent view (added in FD 7). **eventData. isCascading Close**: will be set to true if and only if the details popup just closed is part of a cascading close that is about to also close the parent window receiving this event, in which case the parent may wish to avoid extra processing (like refreshing grids) since the screen is about to be closed. **eventData. eventProcess ed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	**eventProcessed=tr** in the event handler duplicate refresh of t For example, if you v grid that called the d **`eventData.event this.ViewContro all external en`** If you wish to refresh **`eventData.event this.ViewContro master data and grids`**
onAutoGridAfterInit(eventData: EventData.AutoGrid. AfterInitEventData)	**eventData. gridId** = the ID of the auto grid	Called after the auto

onAutoGridToolbarSetState(eventData: EventData.AutoGrid.ToolbarSetStateEventData<TGridRecordObservableRecord>)	**eventData.gridId = the ID of the auto grid** **eventData.dataRow: selected data row** **eventData.buttonState: grid toolbar button state**	**Called when the g to enabled or disa change. The purp customize the ena in the selected gri** **`public onAutoG Service.Commun ToolbarSetStat void { if(event == "API-6053") event } `** **WARNING: Do not handler. Only use a dataRow value a can potentially be**
onAutoGridSelectionStateChange(eventData: EventData.AutoGrid.SelectionStateChangeEventData)	**eventData.gridId** = the ID of the auto grid **eventData.hasSelectedRows**= boolean, true if the grid has at least on row selected, false if no rows selected.	This event is only pu "selectionColumn". T no rows selected to a for this is to enable/d selection in the grid.
onAutoGridButtonClick(eventData: EventData.AutoGrid.ButtonClickEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row	Called when custom can be added using TODO: Need to supp
onAutoGridSchemaDefined(eventData: EventData.AutoGrid.SchemaDefinedEventData)	**eventData.gridId** = the ID of the auto grid **eventData.gridSchema**= grid schema object	Called when the grid
onAutoGridConfirmDataEvent(eventData: EventData.AutoGrid.ConfirmDataEventData<TGridNgData> ,processEvent: (processIt?: boolean) => void)	**eventData.gridId** = the ID of the auto grid **eventData.eventData.responseData.data.<nameoftheconfirmdataset>**: this is the extended data (e.g. confirmation data) whose format will vary according to the entity requirements.	Called when a create

	eventData.action: "update", "create", or "delete" **eventData.dataConfirmed()**: The function handle to be called when the data has been successfully confirmed **eventData.confirmationCancelled()**: The function handle to be called when the data has been cancelled. **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	
onAutoGridGetFieldSchema(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.GetFieldSchemaEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: the data row used to get the fields schema **eventData.fieldSchema**: the default field schema, may be modified directly by the handlers **eventData.editing**; whether to get the schema for rendering for editing	Called when getting with dynamic columr

ViewFieldTSHandler<TNgData>

View events :

Method	Event Data Properties	Description

onFieldChange(viewField: IViewField<any>,eventData: EventData.QraView.FieldChangeEventData, processEvent: (processIt?: boolean) => void)	**eventData.fieldName**: the ID of the changed field **eventData.fieldValue**: the new value of the changed field **eventData.fieldValueOrig**: the original value of the changed field	Called when a field value is changed in a standalone (non-data-grid) field. Fired as soon as the UI focus is moved away from the changed field. Note: only fired in response to field changes in the UI (e.g. a user entering data), not when fields are bound to data from the model. Note: in order to cancel the fieldChange event (and restore the prior value to the field) it is necessary for the subscriber to call "processEvent(false)".
onFieldLeave(viewField: IViewField<any>,eventData: EventData.QraView.FieldLeaveEventData)	**eventData.fieldName**: the ID of the field **eventData.fieldValue**: the current value of the field	Called when a field loses focus, regardless of any change to the field value.

Generics types:

- TNgData : interface that represents structure of ngData in view controller.
- TGridNgData : interface that represents structure of ngData in the grid (same structure as json that's being transferd)
- .TGridRecord : interface that represents on grid row.
- TGridRecordObservableRecord : merged TGridRecord interface and kendo.data.ObservableObject interface.

Grid events:

Event Type	Event Data Properties	Description
onAutoGridBindData(eventData: EventData.AutoGrid.BindDataEventData<TGridRecord>)	**eventData.gridId:** the ID of the auto grid **eventData.dataRows** : an array containing the data rows that were bound	Called when the view new data is bound. TODO: Commodity C Update maint screen just the correct detai
onAutoGridRowChange(eventData: EventData.AutoGrid.RowChangeEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row	Called whenever the TODO: Need to supp
onAutoGridFieldChangeEvent(eventData: EventData.AutoGrid.FieldChangeEventData<TGridRecordObservableRecord> , processEvent: (processIt?: boolean) => void)	**eventData.fieldName**: the name of the changed field **eventData.fieldValue**:	Called when an auto focus is moved away Note: only fired in re data), not when field Note: in order to can the field) it is necess

	the new value of the changed field **eventData.gridId**: the ID of the auto grid **eventData.dataRow**: the data of the row that the field is in.	TODO: There is a to thrown when trying t TypeError: Cannot re
onAutoGridFieldLeaveEvent(eventData: EventData.AutoGrid.FieldLeaveEventData)	**eventData.fieldName**: the ID of the field **eventData.fieldValue**: the current value of the field	Called when an auto field value.
onAutoGridNewButtonClick(eventData: EventData.AutoGrid.NewButtonClickEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row **eventData.previousSelectedDataRow**: previously selected data row	Called when the Nev Edit mode. TODO: Need to supp

onAutoGridBeforeEdit(eventData: EventData.AutoGrid.BeforeEditEventData<TGridRecordObservableRecord>, processEvent: (processIt?: boolean) => void)	**eventData.gridId**: the ID of the auto grid
	
	eventData.dataRow: selected data row
	
	eventData.eventProcessed : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing
		(Available as of FD8
		
		Called before the rov
		following example us
		or if the edit should b
		
		`public onAutoGr`
		`Service.Communi`
		`BeforeEditEvent`
		`processEvent: (`
		`if(this.NgD`
		`== "API-6876")`
		
		`{`
		`eventDa`
		`var tes`
		
		`closeButtonText`
		
		`selection back`
		
		`(eventData.data`
		
		`}`
		`this.l`
		`}`
		`}`

onAutoGridAfterEdit(eventData: EventData.AutoGrid.AfterEditEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row **eventData.returnData**: the data returned from the initialize call which will include supplementaryMessages if any exist. This property is only valid for a new grid line. **eventData.action**: "UPDATE" or "CREATE" indicating whether this event is the result of editing an existing row or going into edit on a new row.	This event fires only mode on an existing row it fires after the i dataRow) to the row
onAutoGridEditButtonClick(eventData: EventData.AutoGrid.ButtonClickEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row	Called when the Edit row into Edit mode. TODO: Need to supp
onAutoGridCancelButtonClick(eventData: EventData.AutoGrid.CancelButtonClickEventData)	**eventData.gridId**: the ID of the auto grid	Called when the Car
onAutoGridBeforeAddNew(eventData: EventData.AutoGrid.BeforeAddNewEventData<TGridRecordObservableRecord>, processEvent: (processIt?: boolean) => void)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: currently seleced data row **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken	Called before a new

	control of the event processing	
onAutoGridBeforeUpdate(eventData: EventData.AutoGrid.BeforUpdateEventData<TGridRecordObservableRecord>, processEvent: (processIt?: boolean) => void)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row **eventData.source:** indicates if the event was triggered by the "Next Line" button or the "Done Lines" button. The value will be set to "nextLine" or "doneLines". (available as of FD6 patch 2) **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	Called before an aut update operation). TODO: Need to supp
onAutoGridAfterUpdate(eventData: EventData.AutoGrid.AfterUpdateEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row **eventData.source:** indicates if the event was triggered by the "Next Line" button or the "Done Lines" button. The value will be set to "nextLine" or "doneLines". (available as of FD6 patch 2) **eventData.success:** boolean indicating the success of the update	Called after an auto succeeded or not. Fi TODO: Need to supp

onAutoGridBeforeDelete(eventData: EventData.AutoGrid.BeforeDeleteEventData<TGridRecordObservableRecord>, processEvent: (processIt?: boolean) => void)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	Called before an aut TODO: Need to supp
onAutoGridAfterDelete(eventData: EventData.AutoGrid.AfterDeleteEventData)	**eventData.gridId**: the ID of the auto grid TODO: Add a success boolean once we find a way to determine that in the code	Called after an auto succeeded or not. Fi
onAutoGridDetailsLinkClick(eventData: EventData.AutoGrid.DetailsLinkClickEventData<TGridRecordObservableRecord>, processEvent: (processIt?: boolean) => void)	**eventData.gridId**: the ID of the auto grid **eventData.linkUrl**: URL of the default navigation link **eventData.linkLabel**: translated label for the default navigation link **eventData.dataRow**: selected data row **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	Called when the auto attempted. To override the defa code, and be sure to the default navigatio
onAutoGridDetailsLinkClose(eventData: EventData.AutoGrid.DetailsLinkCloseEventData<TGridRecordObservableRecord>, processEvent: (processIt?: boolean) => void)	**eventData.gridId**: the ID of the auto grid	Called when the auto The default behavior refresh. If for some r

	eventData.linkUrl: URL of the default navigation link **eventData.linkLabel**: translated label for the default navigation link **eventData.dataRow**: selected data row **eventData.childState**: contains a reference to the child view's QraView JS controller; used to pass child state back to the parent view (added in FD 7). **eventData.isCascadingClose**: will be set to true if and only if the details popup just closed is part of a cascading close that is about to also close the parent window receiving this event, in which case the parent may wish to avoid extra processing (like refreshing grids) since the screen is about to be closed. **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	**eventProcessed=tr** in the event handler duplicate refresh of t For example, if you v grid that called the d **`eventData.event`** **`this.ViewContro`** **`all external en`** If you wish to refresh **`eventData.event`** **`this.ViewContro`** **`master data and`** **`grids`**
onAutoGridAfterInit(eventData: EventData.AutoGrid.AfterInitEventData)	**eventData.gridId** = the ID of the auto grid	Called after the auto

onAutoGridToolbarSetState(eventData: EventData.AutoGrid.ToolbarSetStateEventData<TGridRecordObservableRecord>)	**eventData.gridId = the ID of the auto grid** **eventData.dataRow: selected data row** **eventData.buttonState: grid toolbar button state**	**Called when the g to enabled or disa change. The purp customize the ena in the selected gri** **`public onAutoG Service.Commun ToolbarSetStat void { if(event == "API-6053") event } `** **WARNING: Do not handler. Only use a dataRow value a can potentially be**
onAutoGridSelectionStateChange(eventData: EventData.AutoGrid.SelectionStateChangeEventData)	**eventData.gridId** = the ID of the auto grid **eventData.hasSelectedRows**= boolean, true if the grid has at least on row selected, false if no rows selected.	This event is only pu "selectionColumn". T no rows selected to a for this is to enable/d selection in the grid.
onAutoGridButtonClick(eventData: EventData.AutoGrid.ButtonClickEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: selected data row	Called when custom can be added using TODO: Need to supp
onAutoGridSchemaDefined(eventData: EventData.AutoGrid.SchemaDefinedEventData)	**eventData.gridId** = the ID of the auto grid **eventData.gridSchema**= grid schema object	Called when the grid
onAutoGridConfirmDataEvent(eventData: EventData.AutoGrid.ConfirmDataEventData<TGridNgData> ,processEvent: (processIt?: boolean) => void)	**eventData.gridId** = the ID of the auto grid **eventData.eventData.responseData.data.<nameoftheconfirmdataset>**: this is the extended data (e.g. confirmation data) whose format will vary according to the entity requirements.	Called when a create

	eventData.action: "update", "create", or "delete" **eventData.dataConfirmed()**: The function handle to be called when the data has been successfully confirmed **eventData.confirmationCancelled()**: The function handle to be called when the data has been cancelled. **eventData.eventProcessed** : boolean which, if set to true by a subscriber, designates that the subscriber has taken control of the event processing	
onAutoGridGetFieldSchema(eventData: Qad.Common.Service.Communication.EventData.AutoGrid.GetFieldSchemaEventData<TGridRecordObservableRecord>)	**eventData.gridId**: the ID of the auto grid **eventData.dataRow**: the data row used to get the fields schema **eventData.fieldSchema**: the default field schema, may be modified directly by the handlers **eventData.editing**; whether to get the schema for rendering for editing	Called when getting with dynamic columr

ViewFieldTSHandler<TNgData>

View events :

Method	Event Data Properties	Description

onFieldChange(viewField: IViewField<any>,eventData: EventData.QraView.FieldChangeEventData, processEvent: (processIt?: boolean) => void)	**eventData.fieldName**: the ID of the changed field **eventData.fieldValue**: the new value of the changed field **eventData.fieldValueOrig**: the original value of the changed field	Called when a field value is changed in a standalone (non-data-grid) field. Fired as soon as the UI focus is moved away from the changed field. Note: only fired in response to field changes in the UI (e.g. a user entering data), not when fields are bound to data from the model. Note: in order to cancel the fieldChange event (and restore the prior value to the field) it is necessary for the subscriber to call "processEvent(false)".
onFieldLeave(viewField: IViewField<any>,eventData: EventData.QraView.FieldLeaveEventData)	**eventData.fieldName**: the ID of the field **eventData.fieldValue**: the current value of the field	Called when a field loses focus, regardless of any change to the field value.

Api reference

BaseTSHandler

This is the base class for BaseQraViewTSHandler and QraBrowseTSHandler. All public and protected methods of this class are available for the inherited classes:

```
    export interface IBaseTSHandler extends IQraViewTSHandlerCallBack,IDestroyable {

         /**
          * Display a kendo window.
          *
          * @param kWindow: jquery element to create window on. When null, an element will
automatically be created.
          * @param popupId:  id of the dialog. When null, "qPopupWindow" will be used.
          * @param options: KendoWindowOptions object.
          * @param initFunction: callback init function.
          * @param closeFunction: callback close function.
          * @param initFocusObj: jquery element to put initial focus on.
          * @param controllerOptions: KendoWindowControllerOptions object.
          */
         launchKendoWindow(kWindow: JQuery, popupId: string, options: KendoWindowOptions,
initFunction: Function, closeFunction: Function, initFocusObj: JQuery, controllerOptions?:
KendoWindowControllerOptions);
         /**
          * display a modal (confirmation) dialog.
          *
          * @param modalOptions: QModalDialogModalOptions object.
          */
         launchQModalDialog(modalOptions: Qad.WebShell.UI.QModalDialogModalOptions);
         /**
          * Gets the kendo window angular controller instance for the specified childLevel.
          *
          * @param childLevel: optional childlevel,if not specified, then the controller at the
          * current level will be returned.
          *
          * @return Kendo window controller or null if there is no controller at the specified
level.
          */
         getKendoWindowCtrlr (childLevel?: number);
```

```
    }
```

BaseQraViewTSHandler

This is the base class for QraViewTSHandler, QraViewFormTSHandler,ViewGridTSHandler and ViewFieldTsHandler. All public and protected methods of this class are available for the inherited classes:

```
    export interface IBaseQraViewTSHandler<TNgData> extends IBaseTSHandler {
        /**
         * Get the view controller
         *
         * @return: view controller object.
         */
        ViewController: IQraViewController<TNgData>;
        /**
         * Get the view controller data object.
         *
         * @return: data object.
         */
        NgData: TNgData;
        /**
         * Get the browse TS handler ( if available ).
         */
        getBrowseTSHandler(): IQraBrowseTSHandler<any>;
        /**
         * Get the maint TS handler.
         */
        getMaintTSHandler(): IQraViewTSHandler<TNgData>;

    }
```

QraViewTSHandler

This is the base class for maintenance view TS handlers, it implements both BaseTSHandler and BaseQraViewTSHandler and in addition the following:

```
    export interface IQraViewTSHandler<TNgData> extends IQraViewTSHandlerEvents<TNgData>,
IBaseQraViewTSHandler<TNgData> {
    }
```

It also has the following methods for helping with view grids :

```
                /**
                * get/set list of view grid handlers that need to be created. If not set, all
view grids will be handled.
                */
        ViewGridsToHandleList(): String[];

        /**
         * Called when a view grid ts handler needs to be created.
         */
        createViewGridTSHandler(viewGrid: IViewGrid<any,any,kendo.data.ObservableObject>):
ViewGridTSHandler<any,any,any,any>;
```

It also implements empty overridable methods for QraViewTSHandler events .

QraViewFormTSHandler

This is the base class for maintenance view forms (and form fields)TS handlers, it implements both BaseTSHandler and BaseQraViewTSHandler and in addition the following:

```
    export interface IQraViewFormTSHandler<TNgData> extends IQraViewFormTSHandlerEvents<TNgData>,
IBaseQraViewTSHandler<TNgData> {
    }
```

It also implements empty overridable methods for QraViewFormTSHandler events .

QraViewTSHandlerWithViewFormTSHandler

This base class implements the same properties,functions and events as QraViewTSHandler, but it has additional properties for creating a view form TS handler and for creating View Grid TS handlers:

```
                /**
                * Method called to create view form ts handler.
                */
        createViewFormTSHandler(): TQraViewFormTSHandler;

                /**
                * List of view fields that will be handled but the view form ts handler. If not
set, all fields are handled.
                */
        ViewFieldsToHandleList(): string[];
```

ViewFieldTSHandler

This is the base class for maintenance view field TS handlers, it implements both BaseTSHandler and BaseQraViewTSHandler and in addition the following:

```
        export interface IViewFieldTSHandler<TNgData> extends IViewFieldTSHandlerEvents<TNgData>,
IBaseQraViewTSHandler<TNgData> {
        /**
         * get the associated view field.
         */
        ViewField: IViewField<TNgData>;
    }
```

It also implements empty overridable methods for QraViewFieldTSHandler events .

ViewGridTSHandler

This is the base class for maintenance view grid TS handlers, it implements both BaseTSHandler and BaseQraViewTSHandler and in addition the following:

```
        export interface IViewGridTSHandler<TNgData,TGridNgData, TGridRecord,
TGridRecordObservableRecord extends kendo.data.ObservableObject> extends
IViewGridTSHandlerEvents<TNgData,TGridNgData, TGridRecord,TGridRecordObservableRecord>,
IBaseQraViewTSHandler<TNgData> {
        /**
         * get the associated view grid.
         */
        ViewGrid: IViewGrid<TGridNgData, TGridRecord,TGridRecordObservableRecord>;
    }
```

It also implements empty overridable methods for QraViewGridTSHandler events .

IQraViewController

This is the interface to the maintenance view controller :

```
    export interface IQraViewController<TNgData> extends IHttpService {
        /**
         * Execute save action.
```

```
         */
        executeSaveAction();
        /**
         * Reset the from.
         */
        resetForm();
        /**
         * Display a modal dialog with a dissalowed messag.
         *
         * @param disallowedActionsMessage: message to display
         */
        displayDisallowedActionsMessage(disallowedActionsMessage: string);
        /**
         * Bind data to the form.
         *
         * @param data: data object to bind.
         * @param refreshSingleRowGrids: refresh all single row grids yes / no
         */
        bindData(data, refreshSingleRowGrids);

        /**
         * Shows the loading image. It will be a modal image, with rotating icon.
         *
         * To cancel this call: hideLoadingImage().
         * It is crucial to cancel this image in all logic paths after it is shown, since it is a
modal overlay which
         * will otherwise prevent the user from further actions on the current view screen.
         *
         * @param onlyDisplayInDialog: optional, true if only the current dialog should have a
waiting cursos, default is false.
         *
         **/
        showLoadingImageFullCoverage(onlyDisplayInDialog?):void
        /**
         * Hides the loading image that was shown by showLoadingImage() call.
         *
         **/
        hideLoadingImageFullCoverage();
        /**
         * Shows the loading image. It will be a modal image, with rotating icon.
         *
         * To cancel this call: hideLoadingImage().
         * It is crucial to cancel this image in all logic paths after it is shown, since it is a
modal overlay which
         * will otherwise prevent the user from further actions on the current view screen.
         *
         * @param opacity: optional, default is 0
         *
         **/
        showLoadingImage(opacity?):void;
        /**
         * Hides the loading image that was shown by showLoadingImage() call.
         *
         **/
        hideLoadingImage():void;
        /**
         * gets the asociated GroupPanelNavigator
         */
        GroupPanelNavigator: IGroupPanelNavigator;
        /**
         * Compile html for Angular.
         *
         * @param htmlToCompile: html to compile
         * @param ngScope: optional Angular scope object to compiule with
         */
        compileHtml(htmlToCompile: string,ngScope?: ng.IScope) : JQuery;
        /**
         * Refreshes all data grids, if there are any.
         * "refreshSingleRowGrids" determines if single row grids should be refreshed,
         * when false, this can be overridden by the "alwaysRefresh" grid setting from the Java
view controller.
         * Note: this is likely used by the Application plugins so should be treated as a public
API.
         */
        refreshDataGrids(refreshSingleRowGrids: boolean);
        /**
        * Causes the view to re-query its data.  The re-query of data based on the input row of
data fields
        * that contain the new key field(s) to query. e.g. the selected row of a browse. Note:
```

```
This function does not reload the HTML content, just the data.
        * Binds the data in the input DataRow to the form fields.
        * The optional linkFieldMap maps source field names (i.e. browse fields) to target field
names (i.e. maintenance view fields) to
        * get proper data linkage between the two in cases where the data row has different field
names (e.g. coming from a browse).
         */
        requery(dataRow, linkFieldMap?);
        /**
         * Disable all form inputs
         *
         * @param isDisabled: disable or enable
         */
        disableFormInputs(isDisabled: boolean);
        /**
         * Get a view grid.
         *
         * @param gridId: id of the grid in the view.
         *
         * @return: view grid object.
         */
        getViewGrid(gridId: string): IViewGrid<TNgData, any, kendo.data.ObservableObject>;
        /**
         * Get a view label object for a certain label name.
         *
         * @param name: name of the label in the view.
         *
         * @return: view label object.
         */
        getViewLabel(name: string): IViewLabel;
        /**
         * Get a view button object for a certain button name.
         *
         * @param name: name of the button in the view.
         *
         * @return: view button object.
         */
        getViewButton(name: string): IViewButton;
        /**
         * Get a view field object for a certain field name.
         *
         * @param fieldName: name of the field in the view.
         *
         * @return: view field object.
         */
        getViewField(fieldName: string): IViewField<any>;
        /**
         * set flag that form fields have been edited.
         *
         * areFormFieldsEdited: true / false.
         *
         * note: this method does not enable the save button. It does however prevent the
firstFieldChange event from firing.
         *
         */
        setQraViewFormFieldsEdited(areFormFieldsEdited: boolean): void;
        /*
         * setQraViewFormModified: set form pristine & dirty flag, also add/remove class ng-dirty
to indicate form is modifed.
         *
         * @param isFormModified, determens whether to set form in modified state or not
         * @param clearEnableSaveFlag: optional, default=true : clear flag that enables/disables
saving ability
         *
         */
        setQraViewFormModified (isFormModified, clearEnableSaveFlag?);
        /**
         * get the Error group panel.
         *
         * @return: Error group panel object.
         */
        ErrorGroupPanel : IErrorGroupPanel;
        /**
         * Get a label translation
         *
         * @param key: key of the string to translate.
         * @param pluginname: optional, name of the plugin to get the translation from.
         *
         * @return: translation for key.
```

```
         */
        getLabel(key: string,pluginName?: string): string;
        /**
         * Get a set of label translations
         *
         * @param terms: array of keys of strings to translate.
         * @param pluginname: optional, name of the plugin to get the translation from.
         *
         * @return: key/value set of translations.
         */
        getLabels(terms: string[], plugin?: string):{ [key:string]:string; };
        /**
         * Maintenance screen view toolbar object.
         *
         * @return: toolbar object.
         */
        ToolBar: IQraViewToolBar;
        /*
         * reset the Focus to the last known focused field.
         * if no field is know, sets focus to first focusable field.
         *
         * @param onlySelectIFocusableObjects: optional boolean default true, only select objects
that implement IFocusable.
         *                                     this are all form input fields and grids.
         * @param resetFocusAfterDataBind boolean, optional : reset the focus after the data is
loaded in the form.
         */
        resetLastFocus(onlySelectIFocusableObjects?,resetFocusAfterDataBind?): void;
        /**
         * Mark the form dirty
         */
        setFormDirty();
        /**
         * Mark the form pristine and make a copy of the model
         */
        setFormClean();
        /**
         * report an error
         *
         * @param errorMessage : error to report
         * @param isFatal : when true, an error page will be shown
         * @param code, optional: error code to show
         * @param showStackTrace, optional: when true : print stack trace
         * @para= error, optional : error object to use for reporting
         *
         */
        reportError (errorMessage: string, isFatal: boolean,code?:number, showStackTrace?:
boolean, error?: Error): void;

        /**
         * Returns true when form has modifications ( Read-only ).
         */
        IsFormDirty: boolean;


    }
```

IHttpService

This is the interface to the http service for calling the back-end. Maintenance view controllers do implement this interface :

```
/// <reference path="../../../Lib/DefinitelyTyped/angular/angular.d.ts" />
/// <reference path="../../../Lib/DefinitelyTyped/jquery/jquery.d.ts" />
/// <reference path="../../../Lib/kendo/kendo.all.d.ts" />
/// <reference path="../Common/HttpCallController.ts" />


module Qad.Common.Service {
    'use strict';

    export interface IHttpService {
        /**
         * do a http get call and call successCallback or errorCallback.
```

```
         *
         * @param url: url to call.
         * @param successCallback : ng.IHttpPromiseCallback to be called on success.
         * @param errorCallback: ng.IHttpPromiseCallback<any> to be called on error.
         * @param config, optional: ng.IRequestShortcutConfig configuration object
         * @param skipDefaultErrorHandling, optional : skip the default error handling ( default
error handling in http interceptor in index.ts )
         * @param useCache, optional: if true, the get call uses browser caching.
         * @param showLoadingImageFullCoverage, optional: if true, a loading image is shown while
the call is done.
         *
         * @return: HttpCallController : object that can be used to control the http call.
         */
        doHttpGet(url: string, successCallback: ng.IHttpPromiseCallback<{}>,errorCallback: ng.
IHttpPromiseCallback<any>,callCancelledHandler?:(httpCallController: HttpCallController)=>void,
config?: ng.IRequestShortcutConfig,skipDefaultErrorHandling?: boolean,useCache?: boolean,
showLoadingImageFullCoverage?: boolean): HttpCallController;
        /**
         * do a http delete call and call successCallback or errorCallback.
         *
         * @param url: url to call.
         * @param successCallback : ng.IHttpPromiseCallback to be called on success.
         * @param errorCallback: ng.IHttpPromiseCallback<any> to be called on error.
         * @param config, optional: ng.IRequestShortcutConfig configuration object
         * @param skipDefaultErrorHandling, optional : skip the default error handling ( default
error handling in http interceptor in index.ts )
         * @param showLoadingImageFullCoverage, optional: if true, a loading image is shown while
the call is done.
         *
         * @return: HttpCallController : object that can be used to control the http call.
         */
        doHttpDelete(url: string,successCallback: ng.IHttpPromiseCallback<{}>,errorCallback: ng.
IHttpPromiseCallback<any>,callCancelledHandler?:(httpCallController: HttpCallController)=>void,
config?: ng.IRequestShortcutConfig,skipDefaultErrorHandling?: boolean,
showLoadingImageFullCoverage?: boolean): HttpCallController;
        /**
         * do a http delete call and call successCallback or errorCallback.
         *
         * @param url: url to call.
         * @param successCallback : ng.IHttpPromiseCallback to be called on success.
         * @param errorCallback: ng.IHttpPromiseCallback<any> to be called on error.
         * @param config, optional: ng.IRequestShortcutConfig configuration object
         * @param skipDefaultErrorHandling, optional : skip the default error handling ( default
error handling in http interceptor in index.ts )
         * @param showLoadingImageFullCoverage, optional: if true, a loading image is shown while
the call is done.
         *
         * @return: HttpCallController : object that can be used to control the http call.
         */
        doHttpPost(url: string, successCallback: ng.IHttpPromiseCallback<{}>,errorCallback: ng.
IHttpPromiseCallback<any>,callCancelledHandler?:(httpCallController: HttpCallController)=>void,
data?, config?: ng.IRequestShortcutConfig,skipDefaultErrorHandling?: boolean,
showLoadingImageFullCoverage?: boolean): HttpCallController;
        /**
         * Block the UI with a loading image and do a http get call and call successCallback or
errorCallback.
         *
         * @param url: url to call.
         * @param successCallback : ng.IHttpPromiseCallback to be called on success.
         * @param errorCallback: ng.IHttpPromiseCallback<any> to be called on error.
         * @param config, optional: ng.IRequestShortcutConfig configuration object
         * @param skipDefaultErrorHandling, optional : skip the default error handling ( default
error handling in http interceptor in index.ts )
         * @param useCache, optional: if true, the get call uses browser caching.
         *
         * @return: HttpCallController : object that can be used to control the http call.
         */
        blockUIAndDoHttpGet(url: string, successCallback: ng.IHttpPromiseCallback<{}>,
errorCallback: ng.IHttpPromiseCallback<any>,callCancelledHandler?:(httpCallController:
HttpCallController)=>void,config?: ng.IRequestShortcutConfig,skipDefaultErrorHandling?: boolean,
useCache?: boolean): HttpCallController;
        /**
         * Block the UI with a loading image and do a http delete call and call successCallback
or errorCallback.
         *
         * @param url: url to call.
         * @param successCallback : ng.IHttpPromiseCallback to be called on success.
         * @param errorCallback: ng.IHttpPromiseCallback<any> to be called on error.
         * @param config, optional: ng.IRequestShortcutConfig configuration object
```

```
         * @param skipDefaultErrorHandling, optional : skip the default error handling ( default
error handling in http interceptor in index.ts )
         *
         * @return: HttpCallController : object that can be used to control the http call.
         */
        blockUIAndDoHttpDelete(url: string,successCallback: ng.IHttpPromiseCallback<{}>,
errorCallback: ng.IHttpPromiseCallback<any>,callCancelledHandler?:(httpCallController:
HttpCallController)=>void,config?: ng.IRequestShortcutConfig,skipDefaultErrorHandling?: boolean):
HttpCallController;
        /**
         * Block the UI with a loading image and do a http post call and call successCallback or
errorCallback.
         *
         * @param url: url to call.
         * @param successCallback : ng.IHttpPromiseCallback to be called on success.
         * @param errorCallback: ng.IHttpPromiseCallback<any> to be called on error.
         * @param data, optional: data to pass to the call.
         * @param config, optional: ng.IRequestShortcutConfig configuration object
         * @param skipDefaultErrorHandling, optional : skip the default error handling ( default
error handling in http interceptor in index.ts )
         *
         * @return: HttpCallController : object that can be used to control the http call.
         */
        blockUIAnDoHttpPost(url: string, successCallback: ng.IHttpPromiseCallback<{}>,
errorCallback: ng.IHttpPromiseCallback<any>,callCancelledHandler?:(httpCallController:
HttpCallController)=>void,data?, config?: ng.IRequestShortcutConfig,skipDefaultErrorHandling?:
boolean): HttpCallController;
    }
}
```

IErrorGroupPanel

This is the interface of the maintenance view error group panel:

```
    export interface IErrorGroupPanel {
        /**
         * Add errors to the hrid panel.
         *
         * @param errors: array of error dto objects to add.
         */
        addErrorsToErrorGrid(errors: Common.DTO.Error[]);
        /*
         * Removes a list of errors from the error grid.
         *
         * @param errors, list of errors to remove from grid
         */
        removeErrorsFromErrorGrid (errors: Qad.Common.DTO.Error[]);
        /**
         * Removes errors with a given attribute from the error grid.
         *
         * @param fieldName, string representing field ID whose errors to remove
         */
        removeFieldErrorsFromErrorGrid (fieldName: string);
        /**
         * Show the error grid panel.
         */
        showErrorGrid();
        /**
         * Hide the error grid panel.
         */
        hideErrorGrid();
        /**
         * Get the current errors in the error grid.
         */
        getErrorGridErrors (): Qad.Common.DTO.ErrorRow[];

        /*
         * Determines if the error grid contains an error matching the given
         * arguments.
         *
         * @param fieldName, name of field
         * @param code, message code
         * @param gridId, optional, the id of the grid that the error belongs to
         * @param rowUID, optional, the "data-uid" of the row that the error belongs to
```

```
         */
        errorGridHasError (fieldName: string, code: string, gridId?: string, rowUID?): boolean;


    }
```

IGroupPanelNavigator

This is the interface of the maintenance view group panel navigator :

```
    export interface IGroupPanelNavigator {
        /**
         * Get the group panel container element.
         */
        GroupPanelContainer: JQuery;
        /**
         * Get the group panel navigation bar element.
         */
        GroupPanelNavBar: JQuery;
        /**
         * Get the list of group panels.
         */
        GroupPanels: IGroupPanel[];
        /**
         * Get a group panel by it's name.
         *
         * @param name: group panel name
         * @param includeChildGroupPanels: if true, search in child group panels too.
         *
         * @return : group panel object ( null if not found ).
         */
        getGroupPanelByName(name: string,includeChildGroupPanels: boolean): IGroupPanel;
        /**
         * Find the group panel where a certaing grid is on.
         *
         * @param viewGrid: view grid object to find group panel for.
         * @parem onlySearchInSubPanels: optional, if true, search only in child group panels.
         *
         * @return : group panel object ( null if not found ).
         */
        findViewGridGroupPanel(viewGrid: IViewGrid<any,any,kendo.data.ObservableObject>,
onlySearchInSubPanels?: boolean): IGroupPanel;
        /*
         * Hides a group panel.
         *
         * @param panelName, name of the panel (ID in the DOM)
         * @para showInConfig, boolean that determines if the panel should appear in the
         *                     Configuration Panel pop-up. Defaults to true.
         * @param doNotUpdateUserConfig, boolean that determines if this call hides the panel
without influencing the user configuration.
         *                   if true, the panel is hidden, and the user configuration is not
changed. If false, the panel is hidden, but also the user configuration is set to hidden.
         *                   Default value is true
         *
         * Note: Hiding a panel that has sub-panels will also hide those sub-panels and
         *       remove them from the Configuration Panel pop-up (if showInConfig is true).
         *
         */
        hideGroupPanel(panelName, showInConfig, doNotUpdateUserConfig);
        /*
         * Shows a group panel.
         *
         * @param panelName, name of the panel (ID in the DOM)
         * @param showSubPanels, boolean that determines if the panel's children should also
         *                       be shown. Defaults to true.
         * @param showInConfig, boolean that determines if the panel should appear in the
         *                      Configuration Panel pop-up. Defaults to true.
         * @param repositionNav, boolean that determines if the navigator should reposition. If
         *                       true it will reposition to the the current nav or first visible
         *                       nav if the current one is hidden. Defaults to false.
         * @param doNotUpdateUserConfig, boolean that determines if this call shows the panel
without influencing the user configuration.
         *                   if true, the panel is only shown if the user configuration is set to
show it too.
```

```
         *                   if false, the panel is always shown, and the user configuration is
set to show the panel.
         *                   Default value is true
         *
         * Note: Showing a sub-panel will also show its parent panel. And if showInConfig
         *       is true for that sub-panel then the parent's showInConfig will also be set
         *       true.
         */
        showGroupPanel(panelName, showSubPanels, showInConfig, repositionNav,
doNotUpdateUserConfig);

    }
```

IGroupPanel

This is the interface of the maintenance view group panel navigator :

```
    export interface IGroupPanel {
        /**
         * Get the name of the group panel.
         */
        Name: string;
        /**
         * Get the parent group panel.
         */
        ParentGroupPanel: IGroupPanel;
        /**
         * Get the top group panel.
         */
        TopGroupPanel: IGroupPanel;
        /**
         * Get a child group panel by it's name.
         */
        getChildGroupPanelByName(name: string): IGroupPanel;
        /**
         * Retruns true if there is a visible child group panel.
         */
        HasVisibleChildGroupPanel: boolean;
    }
```

IQraViewToolBar

This is the interface of the maintenance view tool bar:

```
    export interface IQraViewToolBar {
        /**
         * Utility to add a button to the bottom tool bar.
         *
         * Note: the button is added to the front of the button list so will appear to the left
         *       of existing buttons.
         *
         * @param name, name of the button
         * @param text, label text, should be already translated
         * @param isDefault, if this is the default button which makes it respond to the enter
key.
         * @param cssClass, CSS class string to control the button color
         */
        addBottomButton(name: string, text: string, isDefault: boolean, cssClass?: string):
ToolbarItem;
        /**
         * Adds an option button to an existing button.
         *
         * @param parentName, name of the parent button
         * @param text, label text for the option button,
         * @param action, string that will get set for the attribute qoptionaction on the button
element
```

```
         */
        addOptionButton(parentName: string, text: string, action?: string): void;
        /**
         * Gets the toolbar button that is the default.
         *
         * @return the default button or null if none exists.
         */
        getDefaultButton(): ToolbarItem;
        /**
         * Utility to set a bottom toolbar button to enabled/disabled.
         *
         * @param name, name of the button
         * @param disabled, true to disable, false to enable
         */
        setToolbarItemDisabled(name: string, disabled: boolean): void;
        /**
         * Utility to set a bottom toolbar button to visible/hidden.
         *
         * @param name, name of the button
         * @param visible, true to show, false to hide
         */
        setToolbarItemVisible(name: string, visible: boolean): void;
         /**
         * Sets the toolbar button isDefault.
         *
         * Note: only one toolbar item can be the default and this will
         *       enforce that by setting isDefault to false for other buttons.
         *
         * @param name, name of the button
         * @param isDefault, if this is the default button which makes it respond to the enter
key.
         */
        setToolbarItemDefault(name: string, isDefault: boolean): void;
        /**
         * Sets the text label of the toolbar button.
         *
         * @param name, name of the button
         * @param text, label text
         */
        setToolbarItemText(name: string, text: string): void;
        /**
         * Sets the button class of the toolbar button.
         *
         * @param name, name of the button
         * @param buttonClass, button class
         */
        setToolbarItemButtonClass(name: string, buttonClass: string): void;
         /**
         * Sets the subType of a toolbar item.
         *
         * @param name, name of the button
         * @param subType, the subType of the button (one of the enum values of
ToolbarItemSubTypeEnum)
         */
        setToolbarItemSubType(name: string, subType: ToolbarItemSubTypeEnum): void;

        /**
         * Finds the button item by name.
         *
         * @param buttonName, name of the button.
         *
         * @return the ToolbarItem or null if not found.
         */
        findItem(buttonName: string): ToolbarItem;



 }
```

IViewField

This is the interface to a field in the maintenance view:

```
    export interface IViewField<TValue> {
        /** Name of the field - Read only */
        Name: string;
        /** Set focus on the field input */
        focus();
        /** Value of the field */
        Value: TValue;
        /**
         * The value the field had when it got focus. - Read only
         */
        ValueOnFocus: TValue;
        /** Base element - Read only */
        Element: JQuery;
        /** Input element - Read only */
        Input: JQuery;
        /**
        * Enable / disable field.
        */
        IsDisabled: boolean;
        /**
        * Set field read only state.
        */
        IsReadOnly: boolean;
        /** Label element - Read only */
        IsKeyField: boolean;
        /** Label element - Read only */
        Label: JQuery;
        /** IsVisible */
        IsVisible: boolean;
        /** Lookup code - Read only */
        LookupCode: string;
        /** Name of the bound field - Read only */
        FieldName: string;

        /**
         * Name of the view field that displays the description.
         */
        DescriptionField.Name: string;
        /**
         * Url to fetch the description from the server.
         * The url can have placeholders for the field name and field value ( {fieldName} ,
{fieldValue} ).
         */
        DescriptionField.GetDescriptionUrl: string;
        /**
         * Function te be called to get the url to fetch the description from the server.
         * The value retuned by this function overrides GetDescriptionUrl.
         * The url can have placeholders for the field name and field value ( {fieldName} ,
{fieldValue} ).
         */
        DescriptionField.GetDescriptionUrlFunction:()=>string;
        /**
         * String to be used to read description value from response data ( e.g. "data.
description" ).
         */
        DescriptionField.GetDescriptionResponseDataValue: string;
        /**
         * Function te be called to get the string to read the description from the response data
( e.g. "data.description" ).
         * The value retuned by this function overrides GetDescriptionResponseDataValue.
         */
        DescriptionField.GetDescriptionResponseDataValueFunction: (data:any )=>string;
        /**
         * Determines whether to use async http call or not ( default=true ).
         */
        DescriptionField.Async: boolean;


    }
```

IViewLabel

This is the interface to a label in the maintenance view:

```
    export interface IViewLabel {
```

```
        /** Name of the field - Read only */
        Name: string;
        /** Base element - Read only */
        Element: JQuery;
        /** IsVisible */
        IsVisible: boolean;
        /** Text */
        Text: string;
    }
```

IViewButton

This is the interface to a button in the maintenance view:

```
    export interface IViewButton {
        /** Name of the field - Read only */
        Name: string;
        /** Base element - Read only */
        Element: JQuery;
        /**
        * Enable / disable field.
        */
        IsDisabled: boolean;
        /** IsVisible */
        IsVisible: boolean;
    }
```

IViewGrid

This is the interface to a grid in the maintenance view:

```
    export interface IViewGrid<TNgData, TRecord,TObservableRecord extends kendo.data.
ObservableObject> {
        /**
         * Id of the grid ( gridwiring.gridname ) - read only
         */
        GridID: string;
        /**
         * Get the data of the currently selected row.
         */
        getSelectedRowData(): TObservableRecord;
        /**
         * Returns the data currently in the data source of the grid - read only.
         */
        Data: TObservableRecord[];
        /**
         * Set a description column for a certain column.
         *
         * @param fieldName = name of the column to add a description field for
         * @param fieldDescriptionFieldProperties : FieldDescriptionFieldProperties object.
         */
        setDescriptionField(fieldName: string,fieldDescriptionFieldProperties:
FieldDescriptionFieldProperties);
        /**
         * Add a function that wil run once when the grid binds it's data.
         *
         * functionToRun: (grid: JQuery)=>void : function that will run when the grid it's data
is bound.
         */
        addRunOnceAfterGridDataBoundFunction(functionToRun: (grid: JQuery)=>void);
        /**
         * Get the element of the currently selected row.
         */
        SelectedRow : JQuery;
        /**
         * Search for a grid row by it's ( partial ) data.
         *
         * data : data to search for
         * onlyCheckKeyFields : compare only the key fields when searching
```

```
         * onlyCheckSchemaFields : compare only fields that are in the grid schema
         * convertDataForComparison : convert the data to the correct type before comparing
         *
         */
        getGridRowByData (data: any,onlyCheckKeyFields?: boolean,onlyCheckSchemaFields?,
convertDataForComparison?):JQuery;
        /**
         * Get the row that is being edited
         */
        EditRow: JQuery;
        /**
         * Put a row in edit mode.
         *
         * row : row element of the row to put in edit mode.
         */
        editGridRow(row: Element | JQuery );
        /**
        * Refreshes the data in the grid.
        * Grids that get data from external entities will make HTTP calls to refresh data from
the server.
        * Grids connected to the form data (ngData; i.e. internal entities) are simply refreshed
from this existing data in memory
        * without getting new data from the server. If new server data is desired, the form data
should be refreshed first by
        * calling $scope.requery($scope.ngData) before calling $scope.refreshDataGrid(id) or
$scope.refreshDataGrids() to refresh all grids.
        *
        * Note: This is likely used by the Application plugins so should be treated as a public
API.
        * Note: A single-row-edit grid will not be refreshed if the screen action is "CREATE". In
other words,
        *       a single-row-edit grid will not be refreshed if the user is in the process of
creating or
        *       saving a new header record.
        *
        * @param "refreshSingleRowGrid", determines if single-row-edit grids should be refreshed,
        *                                when false, this can be overridden by the
"alwaysRefresh"
        *                                grid setting.
        */
        refreshDataGrid(refreshSingleRowGrid: boolean);
        /**
         * Cancel the edit mode of the current row.
         */
        cancelEditRow();
        /**
         * Select a grid row.
         *
         * rowUID : kendo UID to search for the row.
         */
        selectGridRow(rowUID: string);
        /**
         * Kendo Grid Object - read only
         */
        KendoGrid: kendo.ui.Grid;
         /*
         * Set button visibility depending on autogrid state
         *
         * publishSetState : puplish the state setting event.
         *
         */
        autoGridToolBarState(publishSetState?: boolean);
         /*
         * get the Kendo data item of a grid row
         *
         * row element to find row with.
         *
         */
        dataItem(row: string | Element | JQuery): TObservableRecord;
        /*
         * setRowFieldValue: sets value in a specified grid field in the current row
         *
         * @param rowData: data row to set field value on
         * @param fieldId: id of the control within the grid
         * @param value: value to set the field to
         *
         * @return : true if change was successfull.
         *
         */
```

```
        setRowFieldValue (rowData: kendo.data.ObservableObject,fieldId: string, value: any):
boolean;

        /**
         * set the value of a field in the current selected row.
         *
         * fieldId: id of the field
         * value: value to set.
         */
        setCurrentRowFieldValue (fieldId: string, value: any): boolean;
        /*
         * getCurrentRowFieldValue: gets value of a specified grid field in the current row
         *
         * @param fieldId: id of the control within the grid
         *
         * @return: field value
         *
         */
        getCurrentRowFieldValue (fieldId: string): any;

        /**
         * Sets the value of a grid field for multiple rows in the grid.
         *
         * @param fieldId, id of the field
         * @param value, value to set the field to
         * @param selectionFunction, function to determine if given row should be set. This
function will be called for each
         *                           row in the grid and must return true or false. If true then
the row will be set. This
         *                           function will be passed one argument which is the row of
data. If no function is given
         *                           then all rows will be set.
         */
        setFieldValueInRows (fieldId: string, value: any, selectionFunction?: (rowData: kendo.
data.Model) => boolean): void;
        /*
         * setGridColumnDisabled: disables/ enables the specified column in a grid based on field
         *
         * fieldId: id of the column field
         * isDisabled: boolean indicating whether column field should be set to disabled or made
editable.
         *            isDisabled is true --> disable
         *            isDisabled is false --> enable or editable
         *
         */
        setGridColumnDisabled (fieldId: string, isDisabled: boolean);
        /*
         * setGridControlDisabled: disables/ enables the specified field in a grid within a grid
         *
         * fieldId: id of the control within the grid
         * isDisabled: boolean indicating whether field should be set to disabled or made
editable.
         *            isDisabled should be true --> disable
         *            isDisabled should be false --> enable or editable
         *
         */
        setGridControlDisabled (fieldId: string, isDisabled: boolean);
          /*
         * Visually simulates disabling a Kendo Grid by disabling mouse clicks
         * in the grid and setting the grid's opacity.
         *
         * Note: this does not disable any controls in the grid but prevents
         * access to them.
         *
         * @param cover: boolean indicating whether to cover it or uncover it
         * @param opacity: (optional) decimal between 0 and 1 inclusive which
         *                 specifies how opaque it should appear. 1 is the total
         *                 opaqueness and the default, if not specified, is .5.
         */
        setGridCovered (cover: boolean, opacity?: any);
        /**
         * set row actions that are not allowed.
         *
         * @param dataRow: datarow to set disallowed action on
         * @param disallowedActions: string with the disallowed actions ( e.g. "EDITDELETE" );
         */
        setRowDisallowedActions(dataRow: TRecord,disallowedActions: string): void;
        /**
         * get row actions that are not allowed.
```

```
         *
         * @param dataRow: datarow to set disallowed action on
         *
         * @return : string with the disallowed actions ( e.g. "EDITDELETE" );
         */
        getRowDisallowedActions(dataRow: any): string;
        /*
         * Sets the focus to a field in a grid.
         *
         * @param rowUID: the uid of the row
         * @param fieldId: id of the column field
         */
        setFocusToGridField(rowUID: string, fieldId: string);
        /**
         * get / set Visible state
         */
        IsVisible: boolean;
        /**
         * Hide / show grid column
         *
         * @param fieldId: id of the field to hide
         * @param isHidden: if true, hide the field
         */
        hideColumn(fieldId: string,isHidden: boolean);
        /**
         * set column title
         *
         * @param fieldId: id of the field to hide
         * @param title : text to display in the column header
         */
        setColumnTitle(fieldId: string,title: string): void;
        /**
         * Get an array of key field names
         */
        KeyFields: string[];

        /**
         * Details Link ( read only )
         */
        DetailsLink: JQuery;

        /**
         * Get / set details link visibility
         */
        DetailsLinkVisible: boolean;

        /**
         * Get / set details link enabled flag
         */
        DetailsLinkEnabled: boolean;
        /**
         * Add a TS handler
         *
         * @param key: identifier of the tsHandler
         * @param tsHandler : ts handler to add
         */
        addTSHandler(key: string,value: IViewGridTSHandlerEvents<any,TNgData, TRecord,
TObservableRecord>);

                /**
         * get a grid schema field
         *
         * @param fldName: field name
         *
         * @return QraGridViewSchemaField object.
         */
        getGridViewSchemaField (fldName:string): QraGridViewSchemaField;

        /**
         * New button ( read only)
         */
        NewButton : JQuery;

        /**
         * property for enabling or disabling new button
         */
        NewButtonEnabled: boolean;

        /**
```

```
         * property for showing/hiding new button
         */
        NewButtonVisible: boolean;

        /**
         * Edit button ( read only)
         */
        EditButton : JQuery;

        /**
         * property for enabling or disabling edit button
         */
        EditButtonEnabled: boolean;

        /**
         * property for showing/hiding edit button
         */
        EditButtonVisible: boolean;

        /**
         * Delete button ( read only)
         */
        DeleteButton : JQuery;

        /**
         * property for enabling or disabling delete button
         */
        DeleteButtonEnabled: boolean;

        /**
         * property for showing/hiding delete button
         */
        DeleteButtonVisible: boolean;

        /**
         * Set the confirmation data that needs to be send with the next grid http call to the
controller.
         * @param confirmationData : data to send on next call.
         */
        setConfirmationData(confirmationData: any): void;

        /**
         * commit the grid changes to ngData
         */
        commitGridData(): void;


        /**
         * Adds a button to the view grid tool bar, appends in the div area of
kAutoGridCustomToolbar
         * Note: the button is added to the end of the kAutoGridCustomToolbar so will appear to
the right
         *       of existing buttons.
         * @param buttonName, name of the button
         * @param text, label text, should be already translated
         * @param cssClass (optional)- CSS class string to control the button color
         * @param htmlText (optional)- html text e.g. '<span class="fa fa-edit"></span>' - insert
icon
         * @param insertHtmlTextAfter (optional)- true - inserts html text after button text.
         */
        addCustomButton(buttonName: string, text: string, cssClass: string, htmlText: string,
insertHtmlTextAfter: boolean);

        /*
         * setDataGridModified: set data grid dirty flag, also add/remove class ng-dirty to
indicate form is modifed.
         */
        setDataGridModified (dataModified): void;

        /**
         * Whether to save the form before launching the details popup
         */
               SaveFormBeforeLaunchingDetails: boolean;

        /**
         * Add a field validator
         */
        addFieldValidator(fieldName: string,fieldValidator: IFieldValidator,configData:
FieldValidatorConfigData);
```

```
        /**
         * Insert a field validator
         */
        insertFieldValidator(fieldName: string,index,fieldValidator: IFieldValidator,configData:
FieldValidatorConfigData);

        /**
         * Remove a field validator
         */
        removeFieldValidator(fieldName: string,fieldValidator: IFieldValidator,configData:
FieldValidatorConfigData);

        /**
         * Resize grid
         */
        resizeGrid();
}
```

QraBrowseTSHandler

This is the base class for browse TS handlers, it implements BaseTSHandler and in addition the following:

```
    export interface IQraBrowseTSHandler<TRow> extends IQraBrowseTSHandlerEvents<TRow>,
IBaseTSHandler {
        /**
         * Get the view controller
         *
         * @return: view controller object.
         */
        ViewController: IQraBrowseController<TRow>;
    }
```

It also implements empty overridable methods for QraBrowseTSHandler events .

QraBrowseTSHandlerV2

This is the V2 base class for browse TS handlers, it implements QraBrowseTSHandler and in addition the following:

```
    export interface IQraBrowseTSHandlerV2<TRow,TRowObservable extends kendo.data.
ObservableObject> extends IQraBrowseTSHandler<TRow> {
        /**
         * Get the view controller
         *
         * @return: view controller object.
         */
        ViewController: IQraBrowseControllerV2<TRow,TRowObservable>;
    }
```

IQraBrowseController

This is the interface to the browse view controller :

```
    export interface IQraBrowseController<TRow> {
        /**
         * Hide the advanced search lookup panel.
         */
        hideGridAdvanceSearchLookup (fieldName: string, isHidden: boolean);
        /**
         * Publish a vie refresh event.
         */
        publishViewRefeshEvent(eventData: Qad.Common.Service.Communication.EventData.QraBrowse.
ViewRefeshEventData);
```

```
        /**
         * Compile html for Angular.
         *
         * @param htmlToCompile: html to compile
         * @param ngScope: optional Angular scope object to compiule with
         */
        compileHtml(htmlToCompile: string,ngScope?: ng.IScope) : JQuery;
        /**
         * Get event service
         */
        QraEventService: QraEventService;
        /**
         * Get a label translation
         *
         * @param key: key of the string to translate.
         * @param pluginname: optional, name of the plugin to get the translation from.
         *
         * @return: translation for key.
         */
        getLabel(key: string, pluginName?: string): string;
        /**
         * Get a set of label translations
         *
         * @param terms: array of keys of strings to translate.
         * @param pluginname: optional, name of the plugin to get the translation from.
         *
         * @return: key/value set of translations.
         */
        getLabels(terms: string[], plugin?: string):{ [key:string]:string; };
        /**
         * Shows the loading image. It will be a modal image, with rotating icon.
         *
         * To cancel this call: hideLoadingImage().
         * It is crucial to cancel this image in all logic paths after it is shown, since it is a
modal overlay which
         * will otherwise prevent the user from further actions on the current view screen.
         *
         * @param onlyDisplayInDialog: optional, true if only the current dialog should have a
waiting cursos, default is false.
         *
         **/
        showLoadingImageFullCoverage(onlyDisplayInDialog?):void
        /**
         * Hides the loading image that was shown by showLoadingImage() call.
         *
         **/
        hideLoadingImageFullCoverage();
        /**
         * Shows the loading image. It will be a modal image, with rotating icon.
         *
         * To cancel this call: hideLoadingImage().
         * It is crucial to cancel this image in all logic paths after it is shown, since it is a
modal overlay which
         * will otherwise prevent the user from further actions on the current view screen.
         *
         * @param opacity: optional, default is 0
         *
         **/
        showLoadingImage(opacity?):void;
        /**
         * Hides the loading image that was shown by showLoadingImage() call.
         *
         **/
        hideLoadingImage():void;
        /*
         * refreshBrowseSearch: Refresh Browse
         *
         * @param startValues:  key/value pair of initial filter values. Can also be undefined or
null.
         *                      It can also be an actions string with one of the following values
:
         *                      "saveNext": go to next row after refresh ( when save and next was
clicked ).
         * @param publishRowChange : boolean, when true, a row change event is published
         * @param focusOnRow : boolean, when true, focus is set on the first browse grid row
after refreshing
         *
         */
        refreshBrowseSearch(startValues: {[id: string]: string} | string, publishRowChange:
```

```
boolean, focusOnRow?: boolean);
        /*
         * The currently selected row of the browse - read-only
         *
         * @return: selected row data
         */
        SelectedRow: TRow;
        /*
         * select a row in the browse grid.
         *
         * @param rowToSelect: row to select in the browse grid
         */
        selectRow(rowToSelect: JQuery): void;


    }
```

IQraBrowseControllerV2

This is the V2 browse controller interface, it implements IQraBrowseController and in addition the following:

```
    export interface IQraBrowseTSHandlerV2<TRow,TRowObservable extends kendo.data.
ObservableObject> extends IQraBrowseTSHandler<TRow> {
        /**
         * Get the view controller
         *
         * @return: view controller object.
         */
        ViewController: IQraBrowseControllerV2<TRow,TRowObservable>;
    }
```

TypeScript Best Practices

TypeScript best practices include the following:

Use === for Equality

Bad	Good

`if (myVar1 == myVar2)`
 	
`if (error != null)`
	`if (myVar1 === myVar2)`
	
	`if (error !== null)`

Exception Case (Null and Undefined):

- Never use '===' to check to see if a value is undefined or null. Always use ==
- or use if (!errorVar) will always be true when errorVar is undefined or null
- or use if (errorVar) will always be false when errorVar is undefined or null

Never use Null, always use Undefined

- Always use ***undefined*** to set variable or object to non value.

Never use ***null***. Further reading <<link>>

Bad	Good

`let myVar = null;`
`function barFunc( ) {`
`return null;`
`}`
`if (error === null)`
`if (error === undefined)`
`if (error !== null)`
	`let mYVar = undefined;`
	`function barFunc( ) {`
	`return undefined;`
	`}`
	`if (error)`
	`if (error)`
	`if (error != undefined)`

Arrays

- Always use the [] when creating Arrays and not the Array constructor.

Bad	Good

var names:number[] = new Array(4); var names:string[] = new Array("Mary","Tom","Jack","Jill")	var names:number[4]; var names:string[] = ["Mary","Tom","Jack","Jill"];

Use Types as much as possible

Bad	Good

var myVar: any = undefined;
 	
var myVar;
 	
`var names:any[];`
 	
var eventData: FieldChangeEventData<any>;
 	
private confirmViewNavigation(buttonId: any, buttonProperties: any) {
 	
.....
 	
}
	var myVar: string = undefined;
	
	var myVar = undefined;
	
	`var names:string[];`
	
	var eventData: FieldChangeEventData<TGridRecordObservableRecord>;
	
	private confirmViewNavigation(buttonId: string, buttonProperties: ButtonProperties) {
	
	.....
	
	}

Exception Handling

Bad	Good
Ignore error handling Do not use jquery ajax methods. They run outside the control of the framework.	Use try/catch where necessary, and put the UI in an error state by calling the following framework method on the view controller : * * report an error * * @param errorMessage : error to report * @param isFatal : when true, an error page will be shown * @param code, optional: error code to show * @param showStackTrace, optional: when true : print stack trace * @param error, optional : error object to use for reporting * */ reportError (errorMessage: string, isFatal: boolean,code?:number, showStackTrace?: boolean, error?: Error): void;

Ajax Requests

Bad	Good
$.ajax Do not use jquery ajax methods. They run outside the control of the framework.	Use the framwork's IHttpService. see How to do http calls to data controllers

Memory Cleanup & Management

Bad	Good
Ignore memory management. It's really easy to write memory leaks in javascript or TypeScript. Please be sure to take memory management in account.	Take memory management in account : see Avoiding memory leaks in TS handlers

Platform Development in YAB

This page covers the following topics:

Exporting your platform app

Once you have completed the setup of your App in the Platform, you can export it to continue development in OOABL.

To export your App, run:

```
> yab app-export -dir:<directory to export to> -appuri:<app uri>
```

This export will:

1. Create the necessary directory structure including src/impl directory
2. Export the metadata for your app
3. Export the schema for your app
4. Populate the lib directory with your dependencies
5. Create and populate the config/module-config.xml and release-metadata.xml files with your App data

You can run app-export multiple times to export new changes to your platform extension. The following needs to be taken into account:

- Subsequent app-exports will overwrite the exported metadata files, but will not delete any new files added to the platform extension.
- Also the config/module-config.xml file will be overwritten, so it is a good practice to make a backup of that file before you execute the "yab app-export" for a second time.
- Dependencies will be added or removed as necessary, but dependency versions will not be changed.

Configuring your platform app in YAB

To enable the YAB commands to support Platform development, you need to configure your exported Platform App in YAB.

To do this, edit build/config/configuration.properties and add a platform-extension configuration.

```
platform-extension.<instance>.dir=<export directory>
```

For example:

```
platform-extension.acme-extension.dir=/dr01/qadapps/sytest/acmeext
```

To see the available commands, run:

```
> yab help dev-<name>-
```

Where <name> is the name of the directory the platform extension is located in by default.

For example:

```
> yab help dev-acmeext-
```

Optional configuration

The following can be optionally configured:

```
# Determines the command instance's name, e.g.
# dev-<name>-update
# Defaults to the platform extension's directory name
platform-extension.<instance>.name=<name>


# Vendor information, which is included in module-config.xml
platform-extension.<instance>.vendor=<vendor>
platform-extension.<instance>.vendorurl=<vendor url>
```

For example:

```
platform-extension.acme-extension.name=acme-extension
platform-extension.acme-extension.vendor=QAD
platform-extension.acme-extension.vendorurl=https://www.qad.com
```


Many of the examples below will use this acme-extension example. If you pick another platform-extension instance name, replace the commands with your instance name.

API documentation

The API documentation in your environment is available as a webapp in the default tomcat instance.

The default location is

http://hostname:22000/platform-api/

This can be confirmed by running the following and looking for "Platform API"

```
> yab env-info
```

Or alternatively by checking the configuration

```
> yab config webapp.platform-api.url
```

Compiling Code

Your platform app is compiled against the code, dependencies, and schema defined in the platform-extension directory. This compilation also includes the creation of the related procedure library for your platform-extension.

Two core commands are available for compiling code in your Platform App.

`dev-<name>-update`

This creates/updates isolated databases for the schema defined in the platform extension, compiles the code, and creates a procedure library.

For example:

```
> yab dev-acme-extension-update
```

dev-<instance>-compile

This compiles the code (using the same isolated databases) and creates the procedure library.

If your schema hasn't changed, you can use this to perform the compile without going through any database updates.

For example:

```
> yab dev-acme-extension-compile
```

Testing your code

To test your code changes, they need to be deployed to your runtime environment. The following YAB command will update the bootstrap file with a reference to your module's procedure library and trim the appservers.

```
> yab dev-acme-extension-code-register
```

You can then test your code in the runtime.

To remove your code and revert to the default, run:

```
> yab dev-acme-extension-code-unregister
```

Remember to trim the appservers after any code changes. Registering and unregistering the code in the environment will invoke appserver-trim automatically.

```
> yab appserver-trim
```

Updating dependencies

By default, once a dependency on another App is set, it is not updated to a new version. However, you can explicitly force the update of a dependency, which updates the lib directory with the new API contents, and updates the runtime dependency version.

To force your App to use a new version of an Apps API, you need to do two steps:

1. Add the new version of the dependent app into your environment.
2. Force the update.

```
> yab dev-acme-extension-dependency-update <dependent app name>
```

For example:

```
> yab dev-acme-extension-dependency-update sales-app
```

Packaging your App

To promote your App to another environment (for example, a test environment), you need to create a package and install it.

To create the package, run:

```
yab dev-acme-extension-package-create
```

This will create a package in the current working directory based on the Platform App.

Versions

The version of your Platform App is controlled by the .version file in the root of the platform-extension. This file is created by the initial package creation and contains the version sequence information used for the app.

This is the default document, which allocates versions (1.0.0.0, 1.0.0.1, ...)

```
<version-specification>
<version>1.0.0.0</version>
</version-specification>
```

This can be changed to allocate, e.g., versions (1.0.1.0, 1.0.1.1, ...)

```
<version-specification>
<version>1.0.1.0</version>
<revision-max>1</subrevision-max>
</version-specification>
```

Alternatively, you can specify the exact version to use (useful for creating a once-off package with a particular version).

```
> yab dev-acme-extension-package-create -version:1.2.3.4
```

Using the version sequence is the recommended approach to managing your app's version.

Attributes

You can specify package attributes to be included when creating the package.

```
> yab dev-acme-extension-package-create -attribute:key=value -attribute:key2=value2
```

For example, the following would add the variant package attribute to indicate the app is a variant of the app foobar.

```
> yab dev-acme-extension-package-create -attribute:variant=foobar
```

Installing into a new environment

To install the new App into a second environment, you need to copy the created package into a location where it can be referenced and run yab install.

For example:

```
> yab install acme-extension-1.0.0.0.zip
```

This will install the App into your environment and run a yab update.

Archiving workspace

As the final package created by this process does not contain non-runtime artifacts which may be used for further development, you have the option of archiving these changes to a backup location.

This is controlled by the directorybackup configuration, which is automatically generated for platform extensions but can be further adjusted.

The default backup location is the backups/ folder in the root of the platform extension directory, as configured by directorybackup.<instance>.target.dir

```
> yab dev-acme-extension-backup
```

You can then restore your backup.

The backup will be restored into the backup source location, as configured by directorybackup.<instance>.source.dir

```
> yab dev-acme-extension-restore
```

See the Custom Directory Backups section of the *QAD Configuration and Administration Guide YAB 1.8* for background and additional information.

Business Component API Access with Swagger

Introduction

This section explains the use of Swagger for exposing the APIs for business components in Apps.

Developing a Custom App - Step by Step

The steps for developing a custom app are as follows:

0. Database and Data Stores Configuration

Extension database

When you install Channel Islands for the first time, a new extension database is introduced. This is configured in YAB using the db.extension instance.

This extension database is created as an empty database and is intended to be used for any Platform Business Components that are created. It is connected to all standard YAB runtime sessions by default and should be added to any custom sessions that may execute these platform extensions.

The detailed configuration of this database can be reviewed by executing the following:

```
> yab config db.extension*
```

Data stores and environment type

When installing Channel Islands, a database or databases may be marked as being available as a development data store for platform development. Development data stores can only be configured when the environment is identified as a development environment and should not be configured in a test or production environment.

To configure an environment as a development environment, update the environment type:

```
environment.type=development
```

To configure a database as a development data store, ensure it is configured as a data store:

```
db.extension.datastore=true
```

Next, configure the associated data store as type development:

```
datastore.extension.type=development
```

To support platform development, a database must have an SQL Broker enabled. Refer to "Configuring a SQL-92 Secondary Login Broker" in the *QAD Configuration and Administration Guide YAB 1.9* for additional details of how to enable a SQL Broker.

Standard QAD databases such as db.qaddb should not be configured as development data stores. The new db.extension is provided for this purpose or a custom database can be used.

If these settings have been changed in the proper `build/config/configuration.properties` file after the initial installation, they are applied by running:

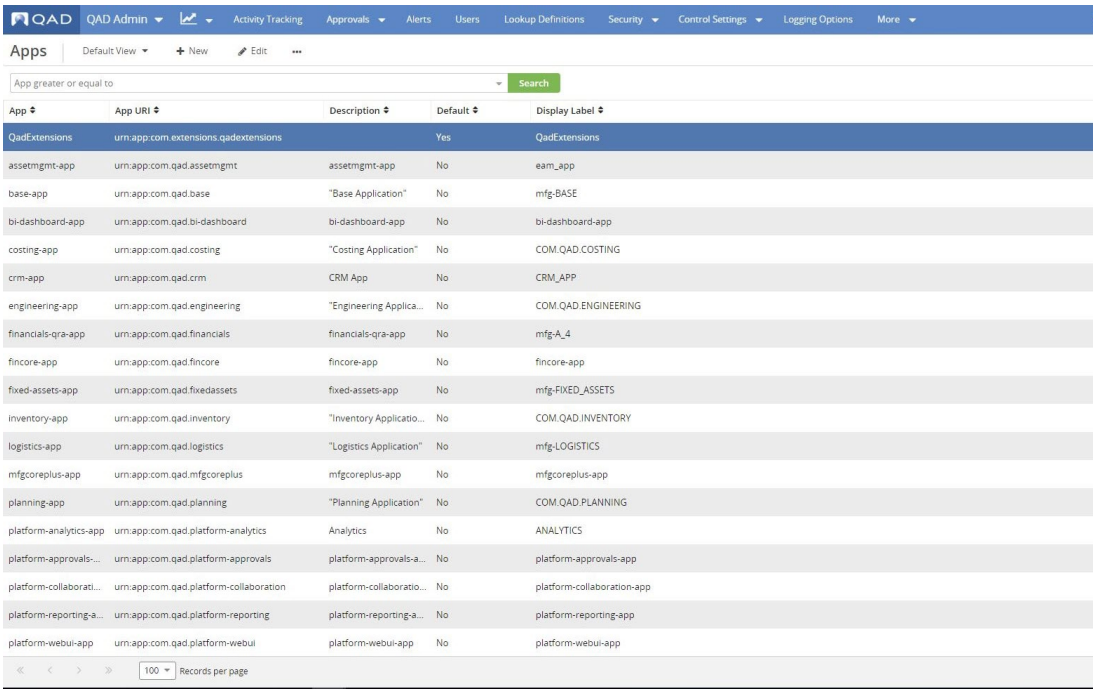
```
> yab update
```

1. Create an App and make it active

1- View the Default App

The system always has one "Default App" that is defined automatically depending on the environment namespace.

You can navigate to **Apps** and be sure the environment already has one "Default App".

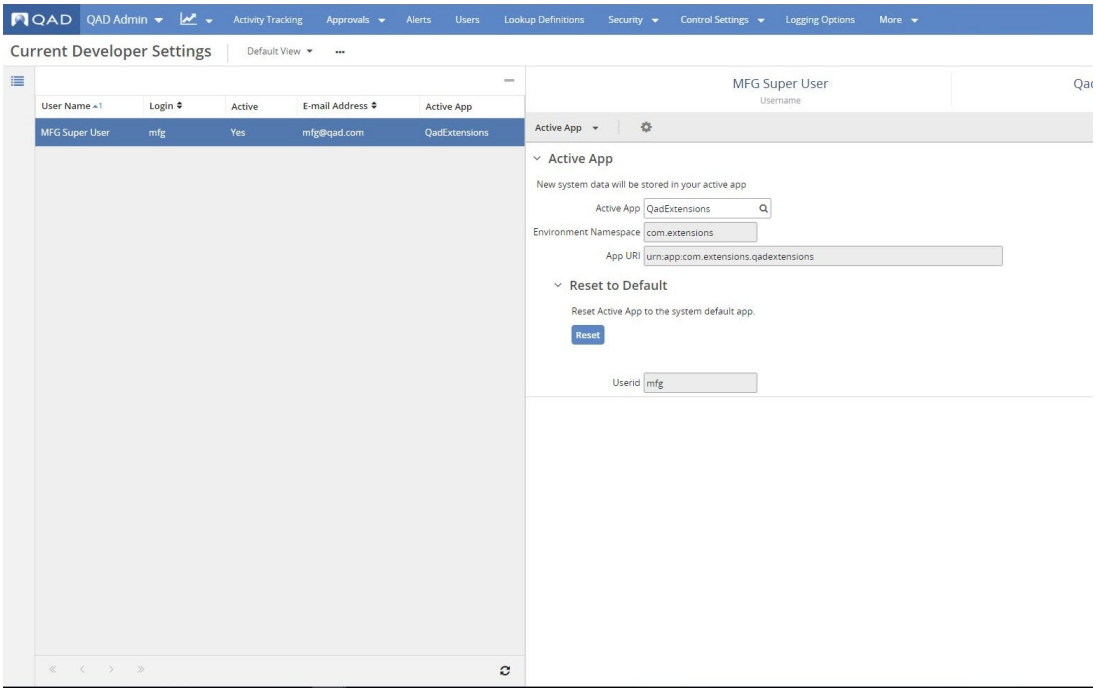


In this example, the system has default App with name "QadExtensions", and with App URI "urn:app:com.extensions.qadextensions".

App URI consists of environment namespace "com.extensions" and the name of App "qadextensions". Having predefined environment namespace, all the further new Apps will be created in this environment namespace "com.extensions".

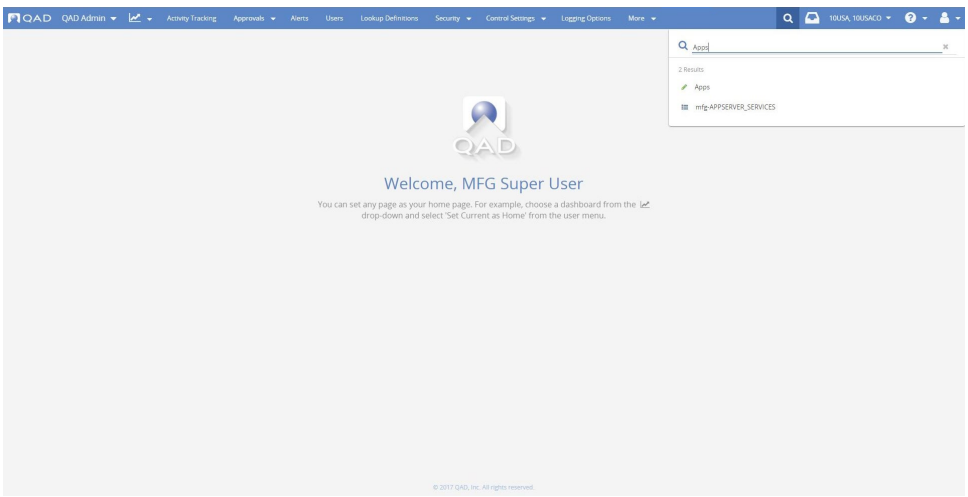
2- View the Active App

By default, the Active App in the system is set up as the Default App. This can be seen in the **Current Developer Settings** for the current user.

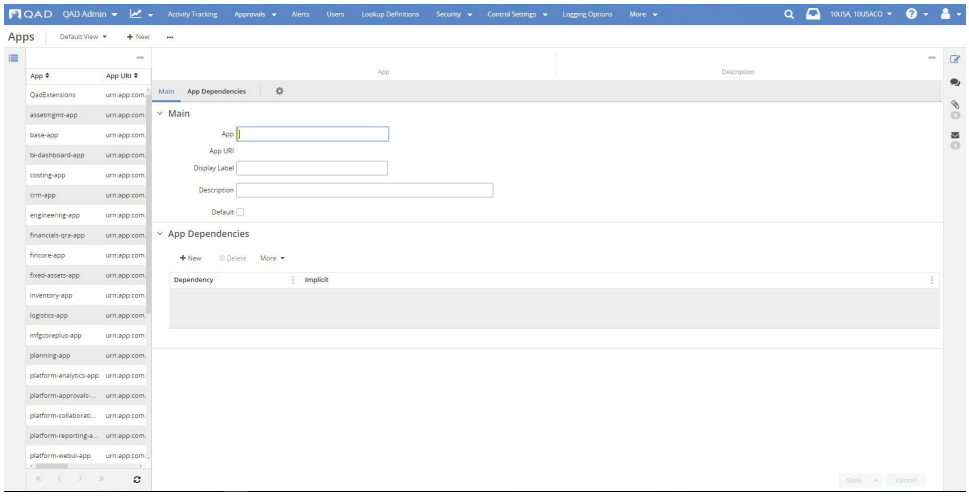


3- Create a custom App

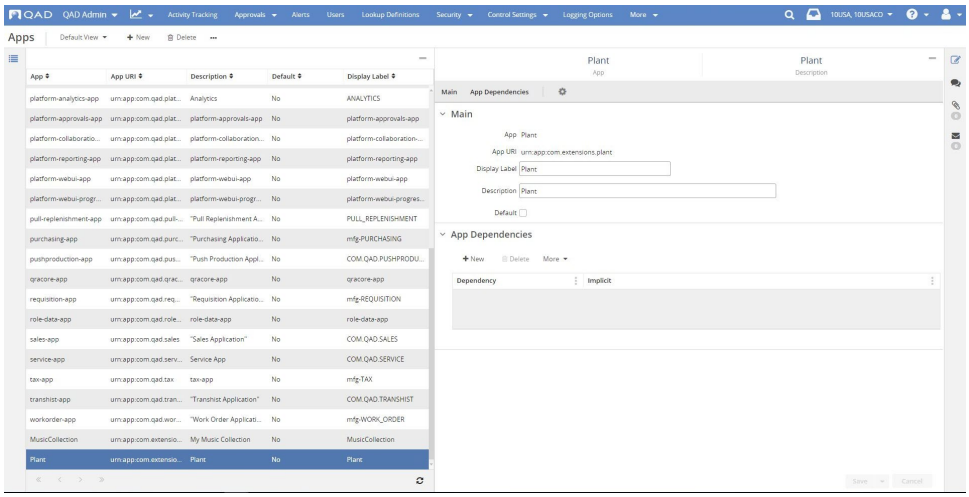
1. Navigate to **Apps** from the menu search.



2. Click the **New** toolbar button.



3. Define the "App" name, "Display Label", "Description", and "Default", and then save App.

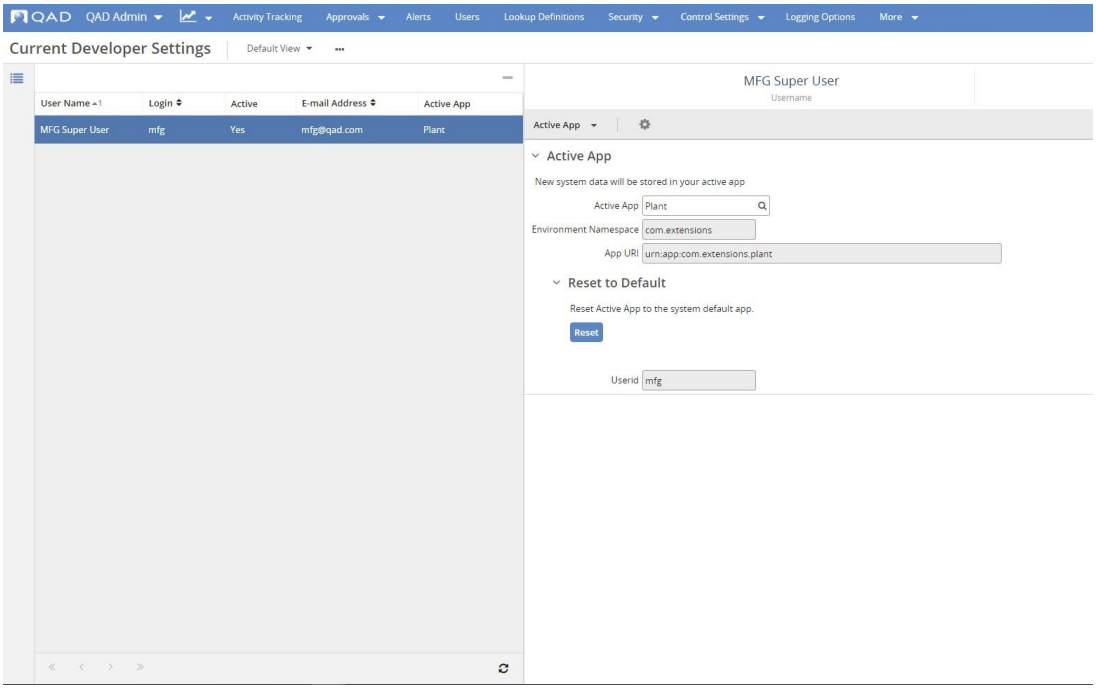


Note: New App is NOT defined as Default App, this means that a new App is still not active in the system. Otherwise, if a new App previously was defined as Default, the system would set default App as Active automatically.

4- Make a new App active

To make a new App active:

1. Navigate to **Current Developer Settings**.
2. Open lookup in the "Active App" field and choose a new App as active.
3. Save.

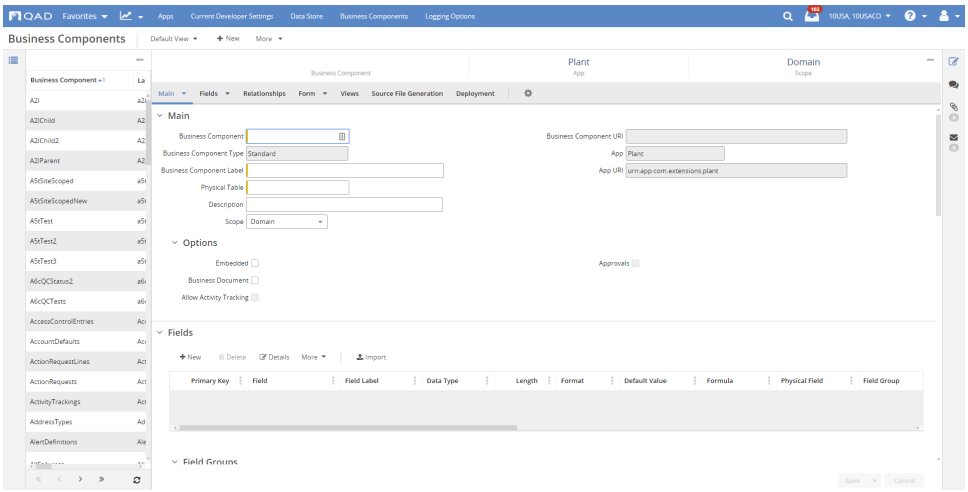


==> Now a new App is set as active and this means that all the further secured resources will be created in this App.

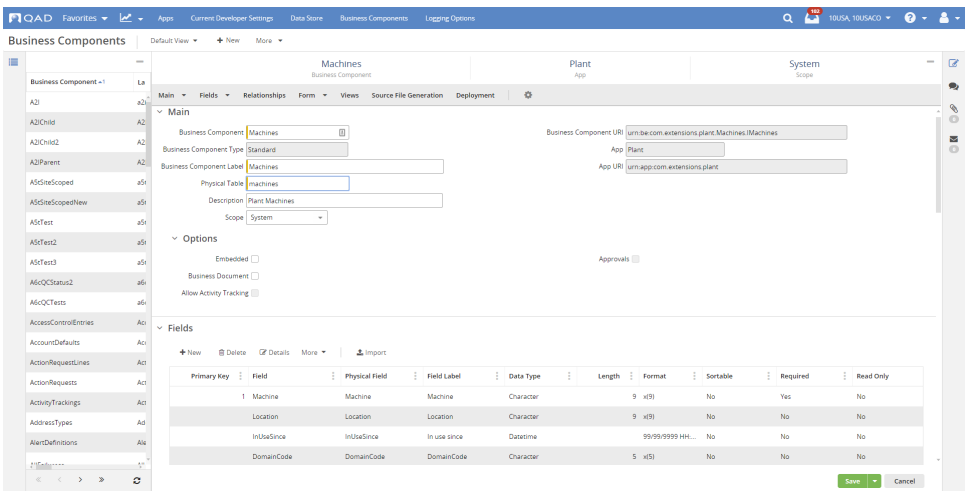
2. Create a business component

1- Create a Business Component

1. Navigate to **Business Components** from the menu search.
2. Click the **New** toolbar button.



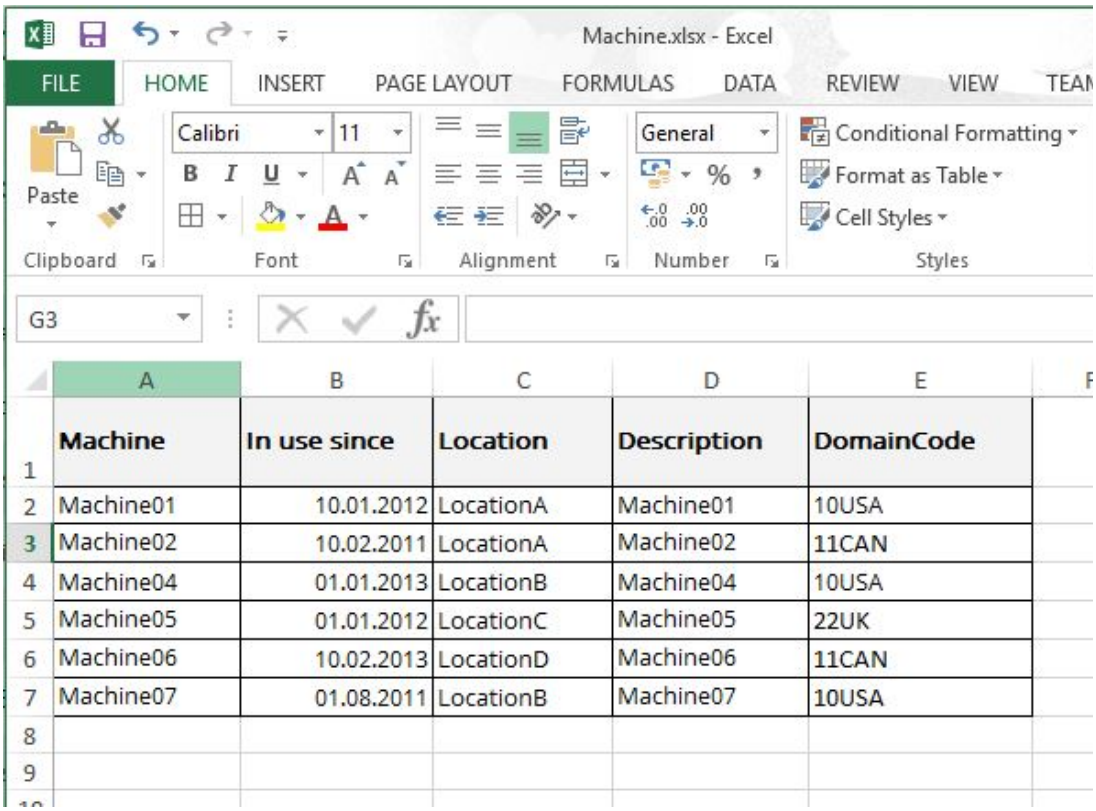
3. Navigate to the **Main** panel.
4. Define the "Business Component" name, "Business Component Label", "Physical Table", "Description", and "Scope". The Business Component URI will be initialized automatically depending on the "Business Component" name.
5. Navigate to the **Fields** panel.
6. Create the fields for Business Component. At least one field should be marked as "Primary Key" (set value "1" to "Primary Key"). The Business Component can have a Composite Key that may consist of several Primary Key fields (in this case, in the "Primary Key" column, the first primary key field should be set as "1", the second - should be set as "2", the third - should be set as "3", etc.).



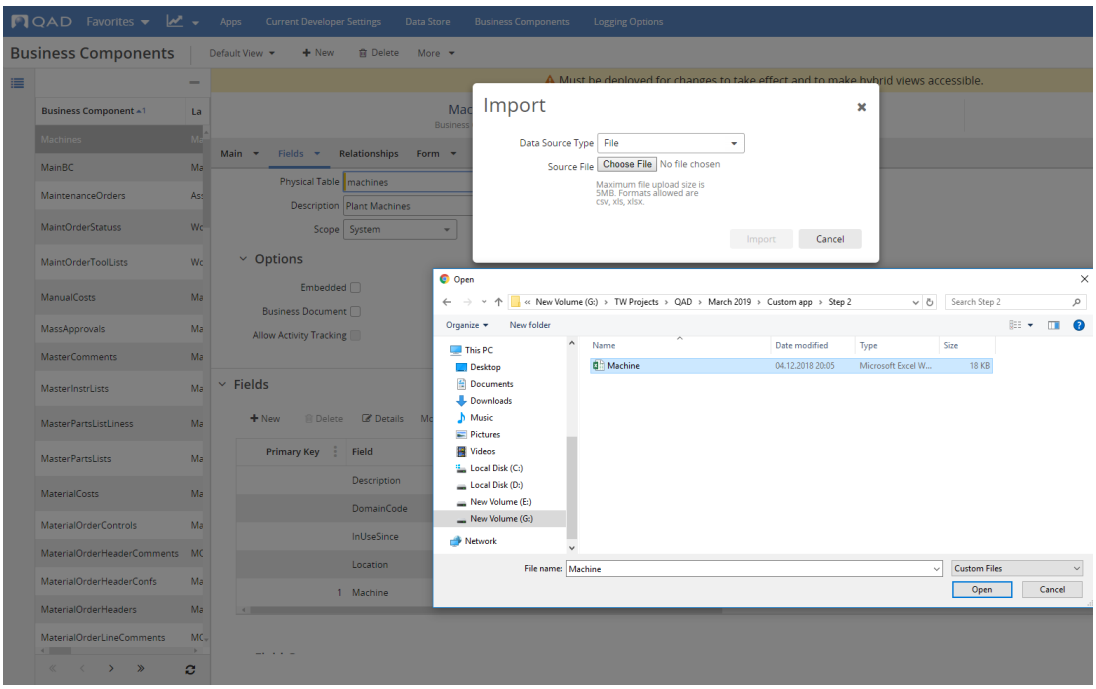
Note: The fields for Business Component can be created in three ways:

- Manually, using the **New** grid button.
- Can be imported from a file (xls, xlsx, or csv), using the **Import** grid button.
- Can be imported from an existing database, using the **Import** grid button.

The example above used import from previously prepared excel file:



	A	B	C	D	E
1	**Machine**	**In use since**	**Location**	**Description**	**DomainCode**
2	Machine01	10.01.2012	LocationA	Machine01	10USA
3	Machine02	10.02.2011	LocationA	Machine02	11CAN
4	Machine04	01.01.2013	LocationB	Machine04	10USA
5	Machine05	01.01.2012	LocationC	Machine05	22UK
6	Machine06	10.02.2013	LocationD	Machine06	11CAN
7	Machine07	01.08.2011	LocationB	Machine07	10USA
8					
9					



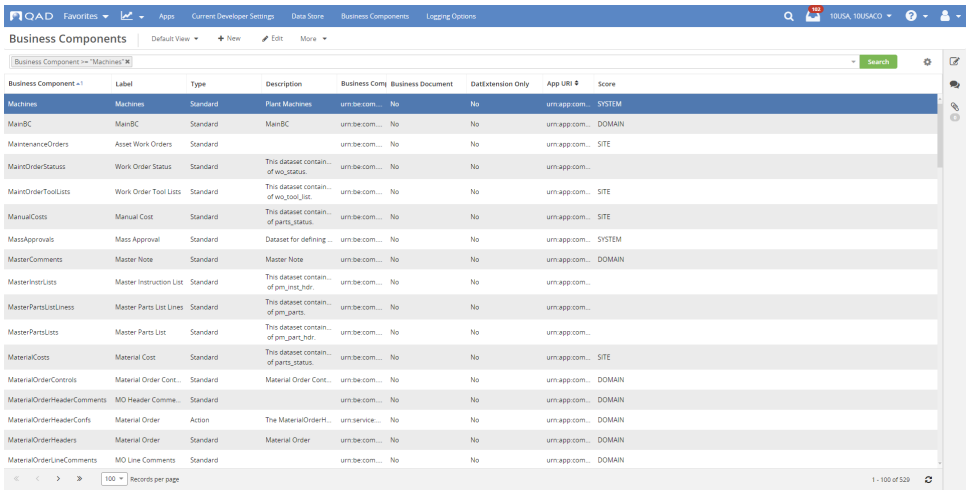
7. Save Business Component.

8. Business Component is saved, the next steps should be creating Form and Hybrid View/Browse for this Business Component.

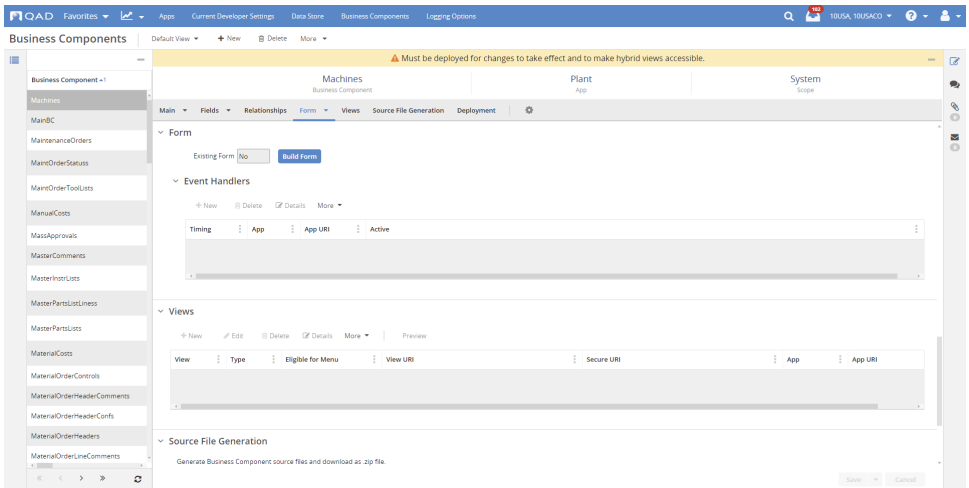
3. Create a view (Form and Hybrid Browse) for a business component

1- Form Builder

1. Navigate to **Business Components** from the menu search.
2. Using the quick search, find the created Business Component.

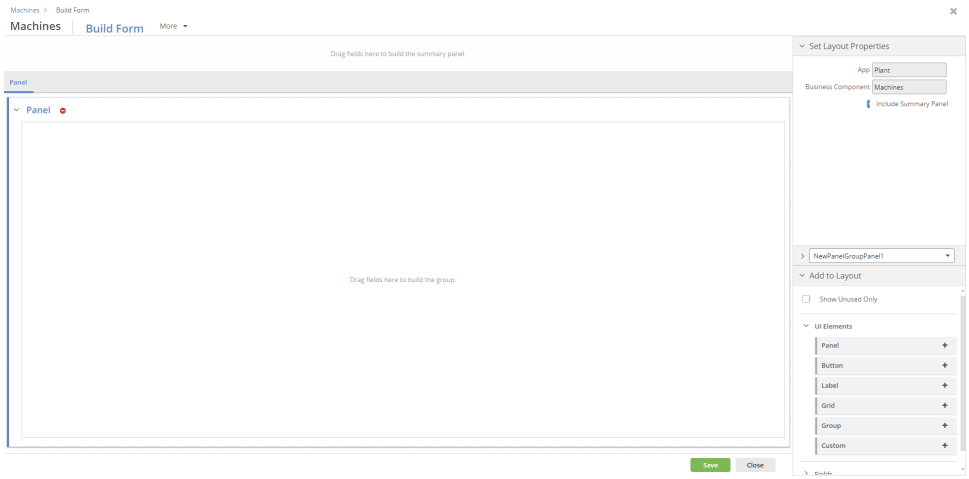


3. Click the **Edit** toolbar button.
4. In the opened screen, there are the **Form** and **Views** panels that give a possibility to create a Form and Hybrid View for this Business Component.

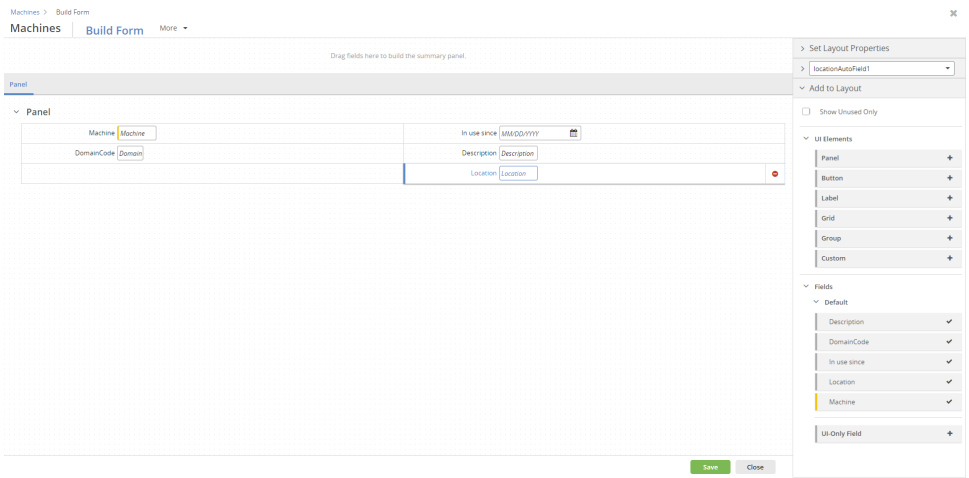


2- Create a Form

1. Navigate to the **Form** panel, and then click the **Build Form** button.
2. Expand the **UI Elements** sub-section in the **Add to Layout** section, and then drag and drop new Panel.



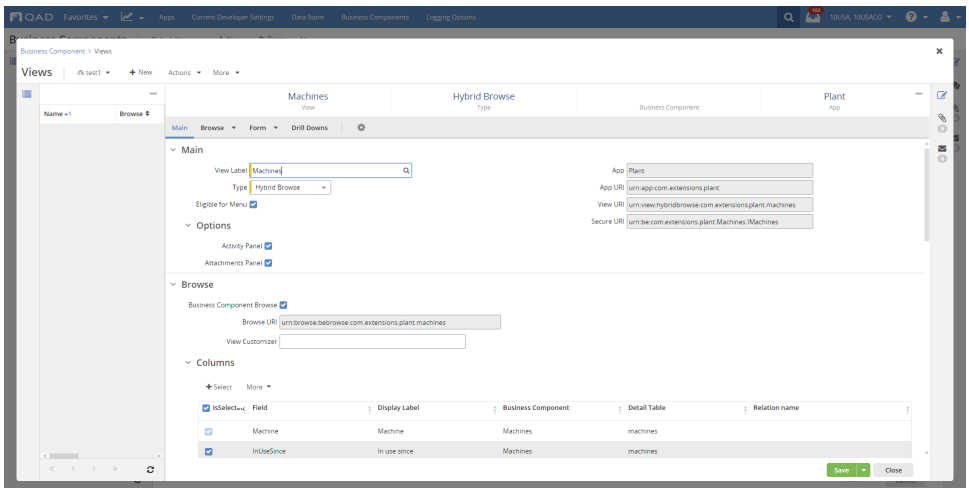
3. Expand the **Fields** sub-section in the **Add to Layout** section, and then drag and drop all the fields on the new Panel.



4. Save the form.

3- Create a Hybrid Browse

1. Navigate to the **Views** panel, and then click the **New** grid button.
2. In the opened screen, manually define the View Label and click **Save**.

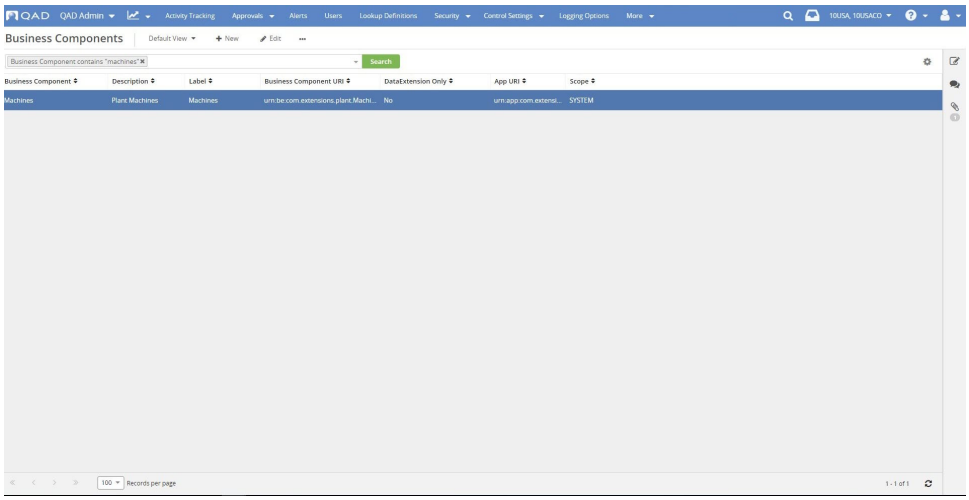


==> The Form and Hybrid Browse are now created for the virtual Business Component.

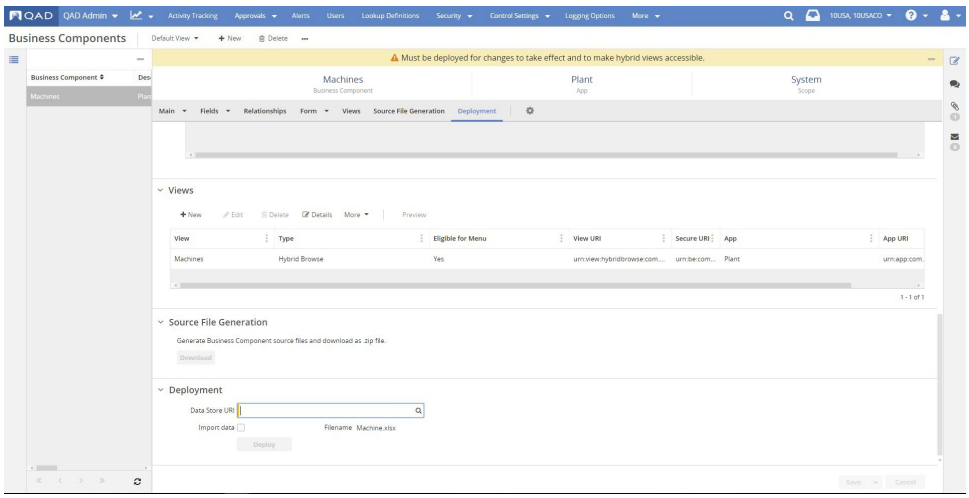
4. Deploy a business component

1- Deploy a virtual Business Component

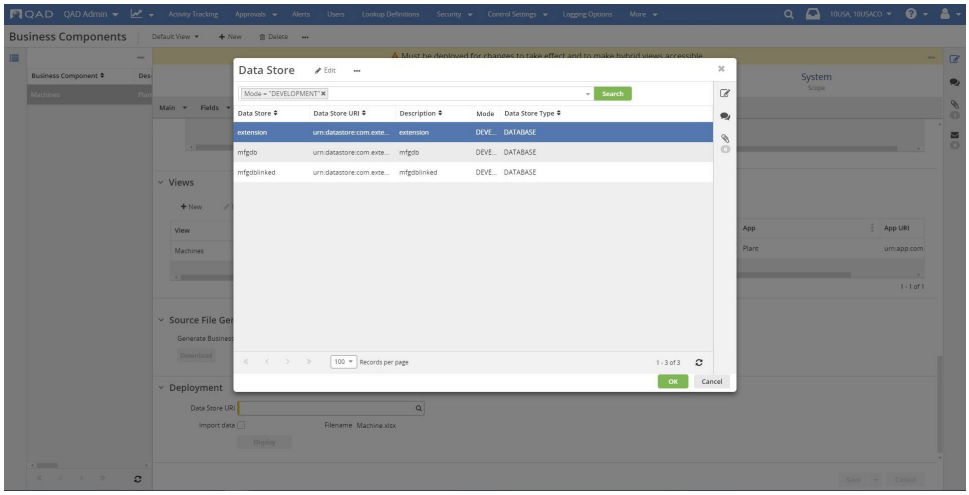
1. Navigate to **Business Components** from the menu search.
2. Using the quick search, find the created Business Component.

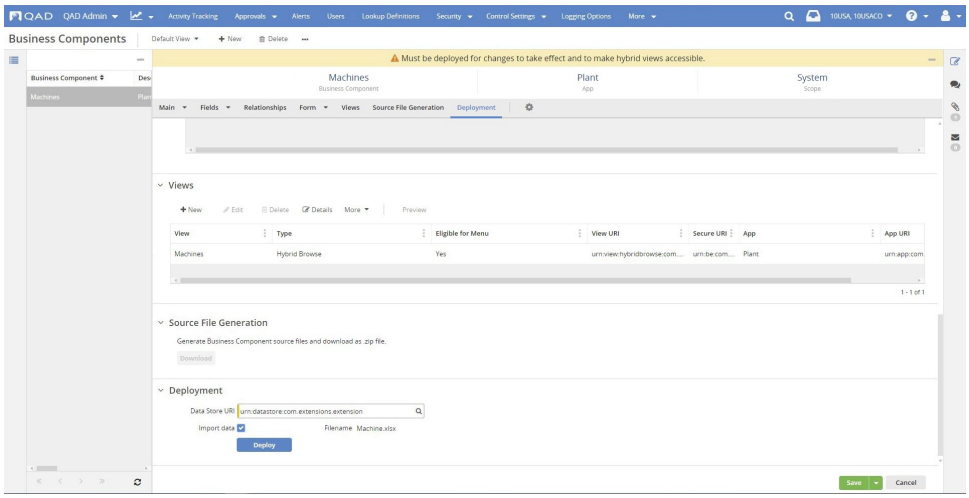


3. Click the **Edit** toolbar button.
4. Navigate to the **Deployment** panel.



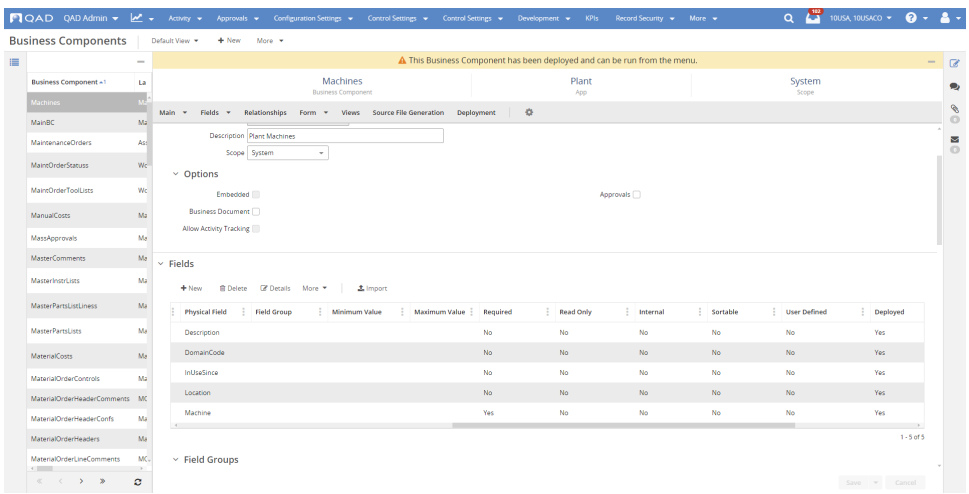
5. Open the lookup for the "Data Store URI" field, choose the data store where the business component should be deployed, and click **OK**.





Note: The current example used a file for import, and that's why there is an ability to select or clear the **Import data** checkbox. Also "Filename" displays the name of the file that was used for importing.

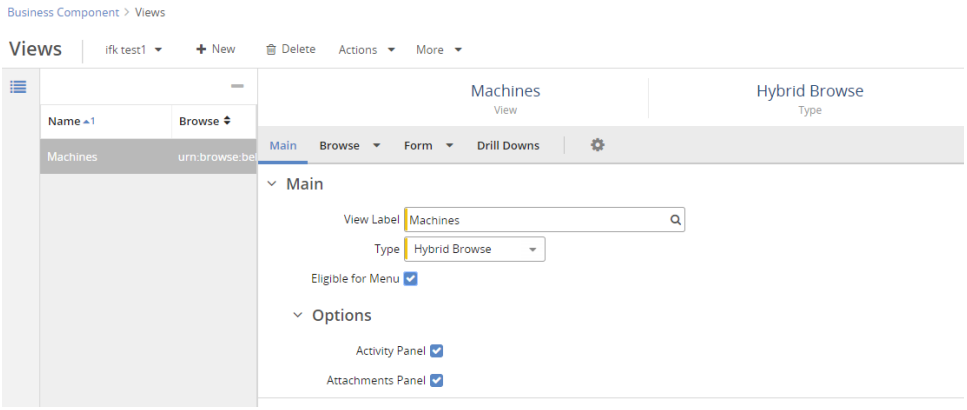
6. Click the **Deploy** button.
   Business Component was successfully deployed, that is represented by the yellow warning message on the top of the screen and all the fields are marked as "Deployed" now.



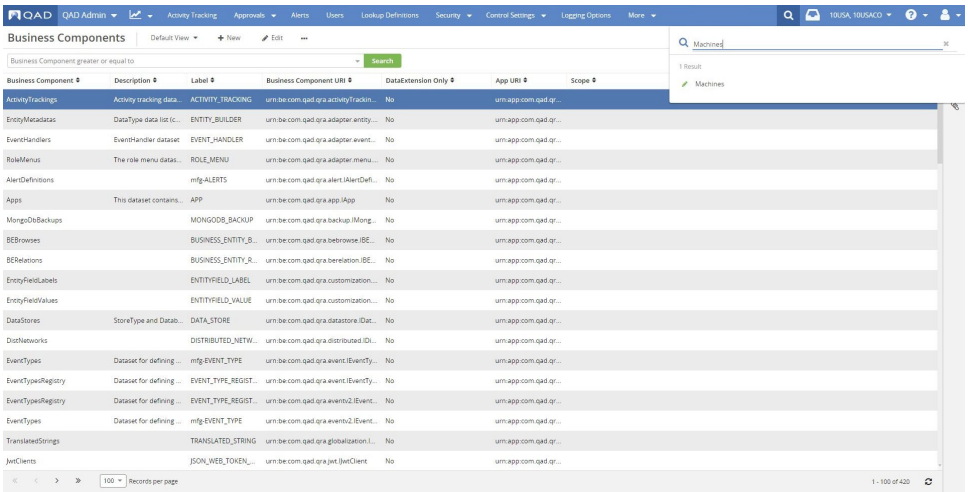
2- Run a view of a Business Component

Once the Business Component was successfully deployed, it is possible to run a view of the Business Component from the menu search using the View Label that was defined for this Business Component (if the **Eligible for Menu** checkbox is selected).

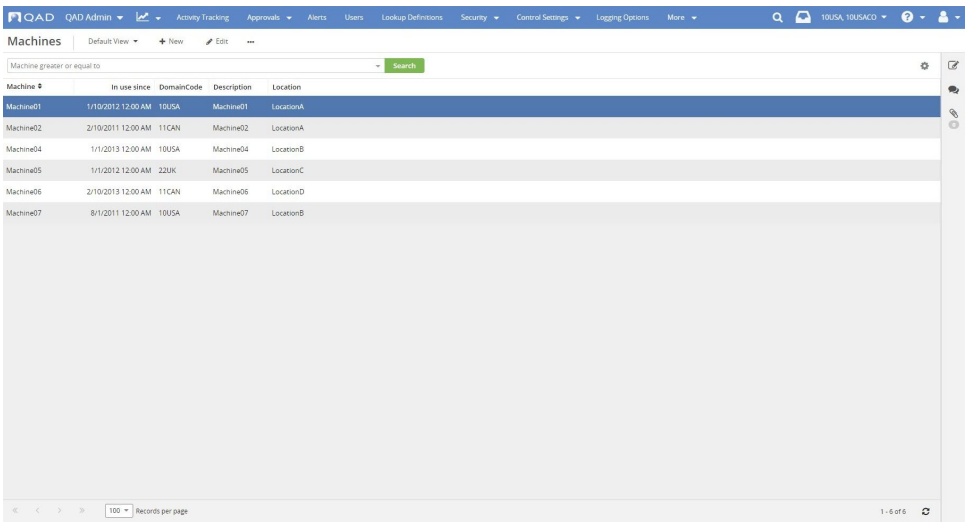
1. Get the label of the view, in this example, it is **Machines**.
2. Be sure the **Eligible for Menu** checkbox is selected.



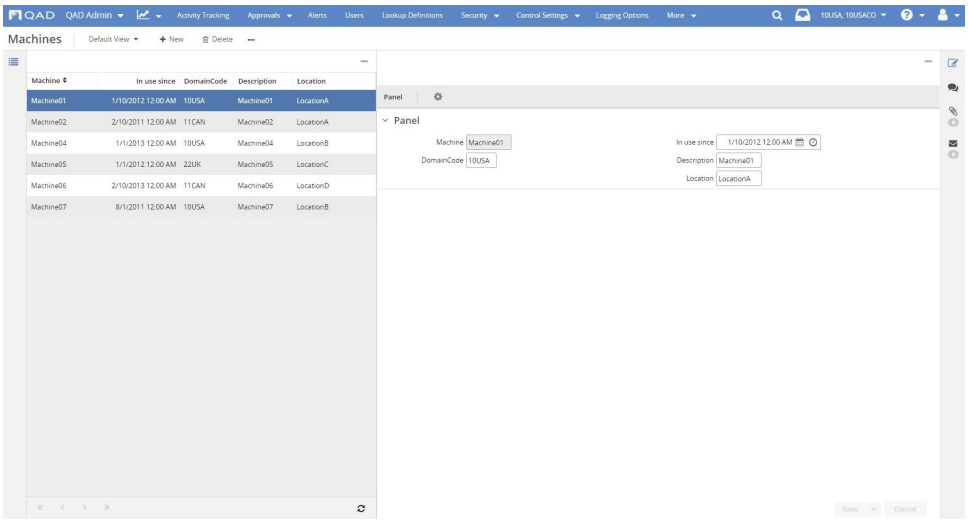
3. Use the menu search.



4. After running from the menu search, the view is opened, including all imported data from the file.



5. Choose any record and click the **Edit** toolbar button.



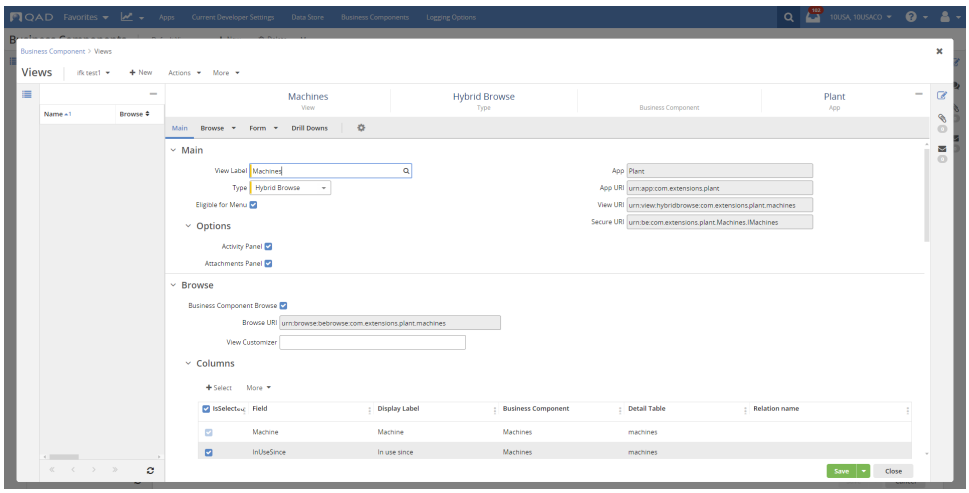
5. Add a business component browse

The Business Component Browse can be created while you build/edit the View.

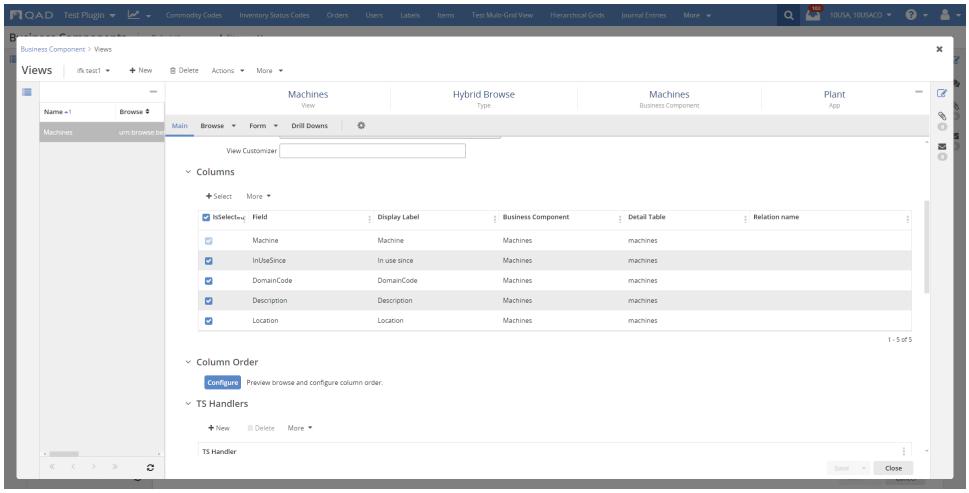
Table of Contents:

1- BC Browse for a single Virtual Business Component without relations

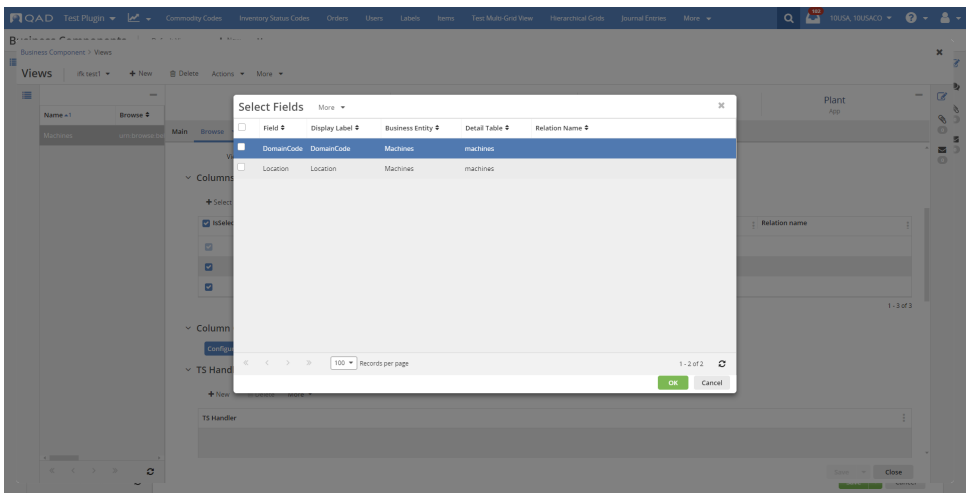
1. Navigate to **Business Components** from the menu search.
2. Using the quick search, find the existing Business Component. In our case, we will use the previously created **Machines** BC.
3. Click the **Edit** toolbar button.
4. Navigate to the **Views** panel and click the **New** grid button. The Form should be already built, otherwise the **New** grid button stays disabled.
5. The **Views** screen should be opened with the Hybrid Browse Type by default. Pay attention that the **Business Component Browse** checkbox is selected automatically on the **Browse** panel. This means that the View will be created with the BC Browse.



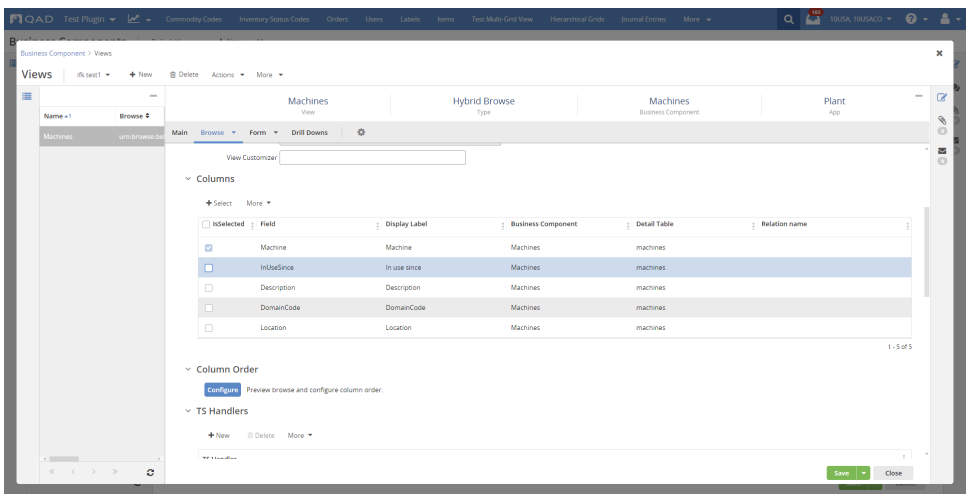
6. Define the View Label.
7. Pay attention that the **Columns** grid consists of those fields that were dragged to the Panel in the Form Builder.



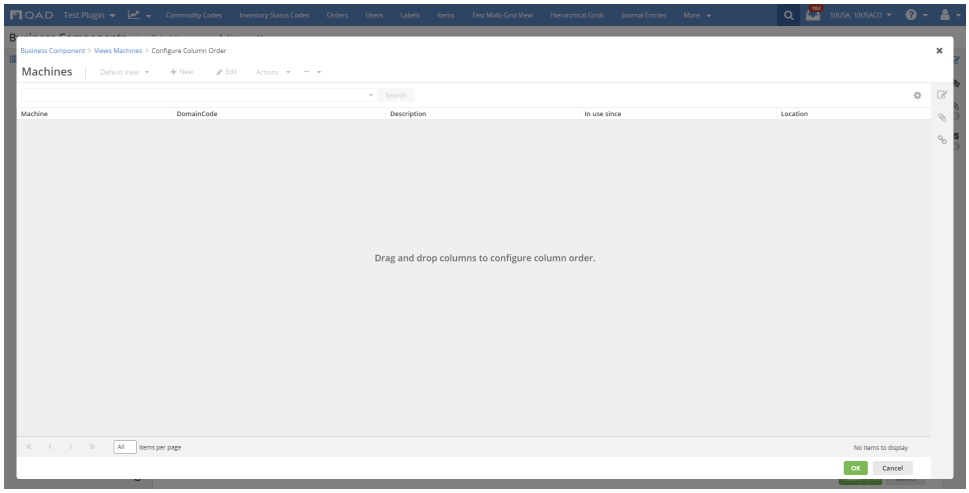
8. Fields that were not dragged can be found in the Selector and added to the **Columns** grid. To do this, click the **+Select** button and select the fields you need in the **Select Fields** pop-up window that opens.



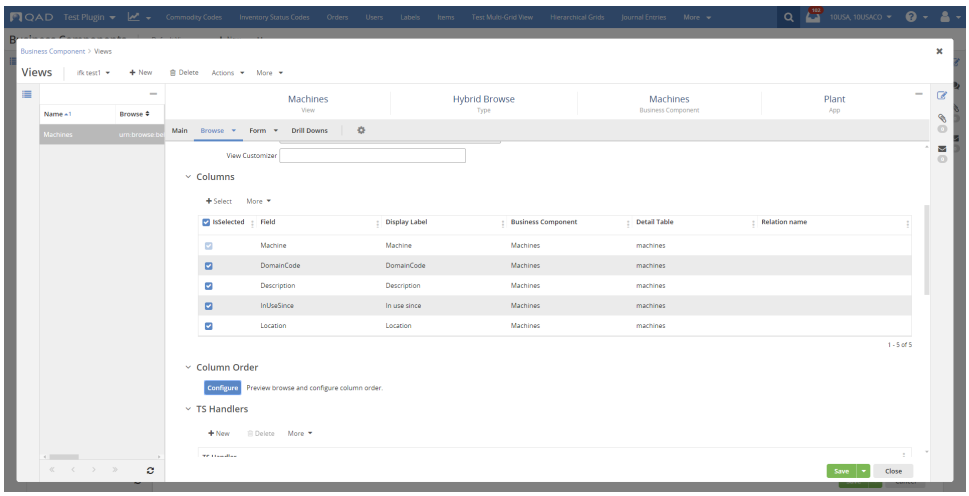
9. Click **OK**.
10. The fields can be unselected (except the "Primary Key" field, it is marked and disabled by default).
    Important: The "Primary Key" field is always present in the **Columns** grid even if it was not dragged to the Panel in the Form Builder.



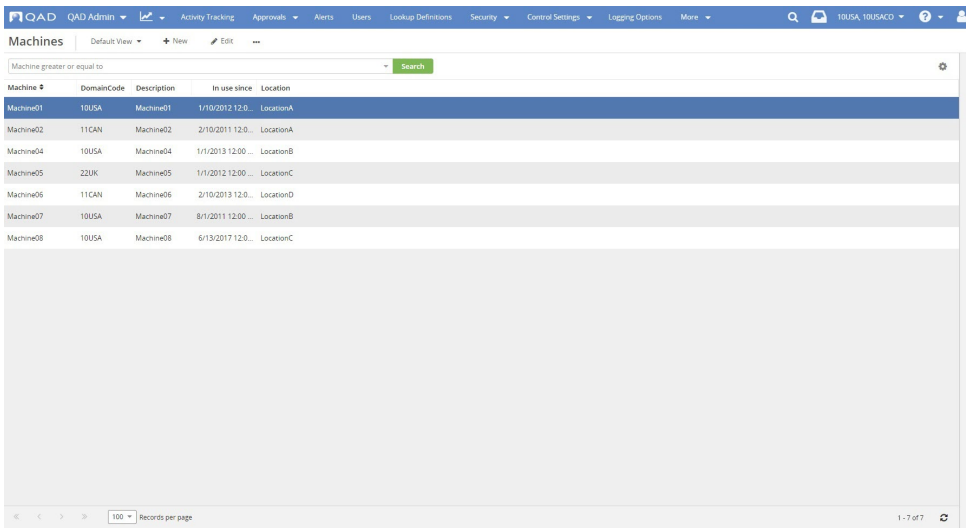
11. Mark the fields you need in BC Browse as selected.
12. Click the **Configure** button on the **Column Order** panel. The Preview for Business Component Browse opens.
13. Change the fields' order if you want and save the changes.



14. See that the fields' order changed in the **Columns** grid.

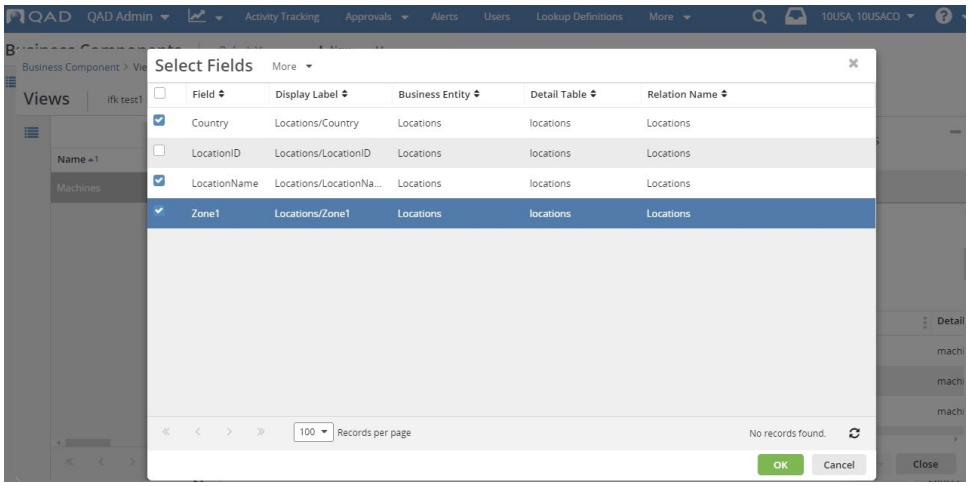


15. Click the **Save** button and close the **Views** screen.
16. The Business Component Browse can be opened from menu search (only for already deployed Business Components); it is ready for CRUD operations.

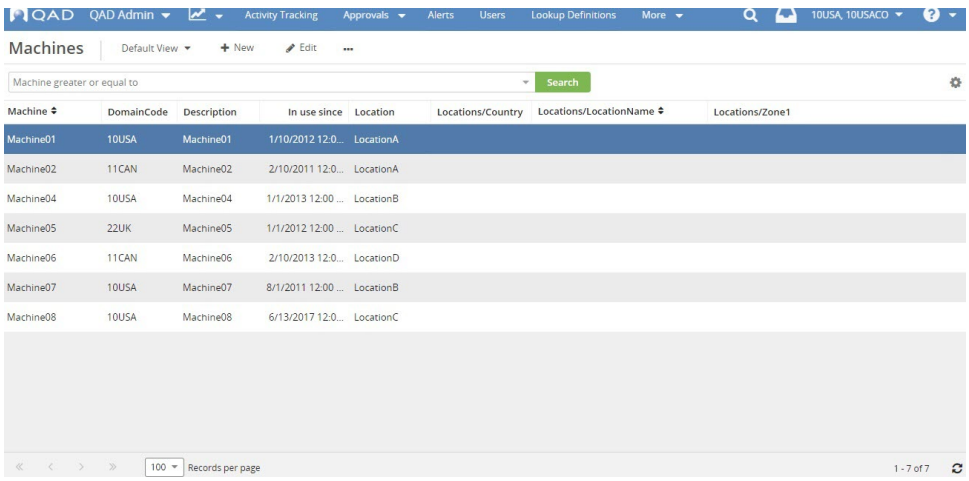


2- BC Browse for Virtual Business Component related to another Virtual Business Component (Traditional (non-extension) Relationship with 1-1 cardinality)

1. See 6 p., there is a non-extension Relationship between the Main BC **Machines** and related BC **Locations**.
2. You can create a Hybrid Browse with BC Browse and all fields from the Main and related Business Components can be included.
3. Related fields can be selected only in the Selector and after that can be added to the **Columns** grid. To do this, click the **+Select** button and select fields you need in the **Select Fields** pop-up window that opens.
4. For non-extension Relationship, the "Primary Key" field from the related BC can be selected and added to the BC Browse.

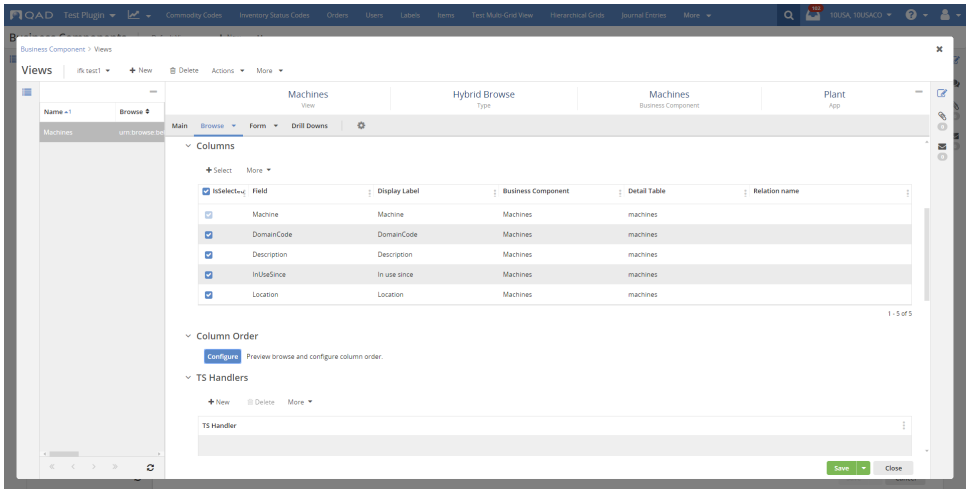


5. Confirm the saving.
6. Only fields that belong to the deployed Business Component will be displayed in the Preview of BC Browse, so you need to deploy both the Main and related BCs if you want to observe all the fields you have included.
7. Here you can see fields from the Main BC and related fields which are displayed with the appropriate Relation Name ("Locations/Country", "Locations/LocationName", and "Locations/Zone1").

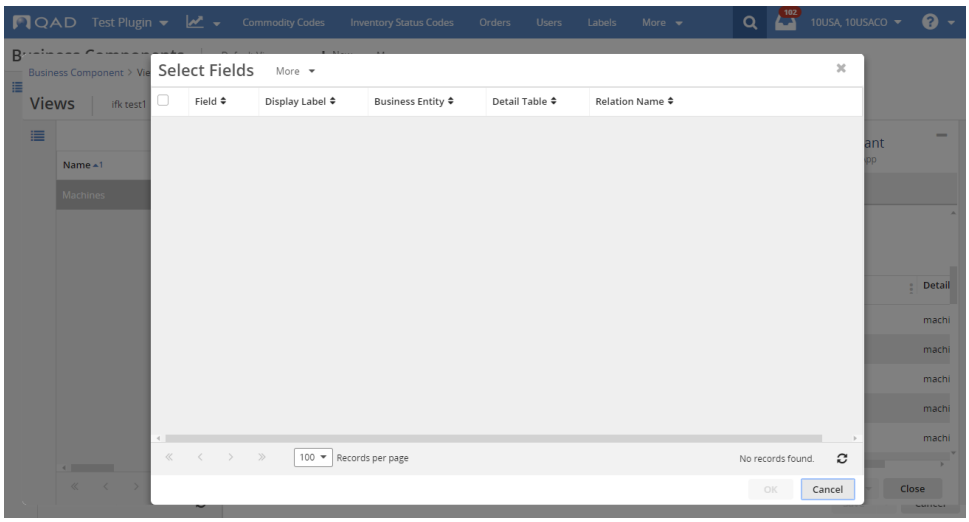


3- BC Browse for Virtual Business Component related to another Virtual Business Component (Traditional (non-extension) Relationship with 1-N cardinality)

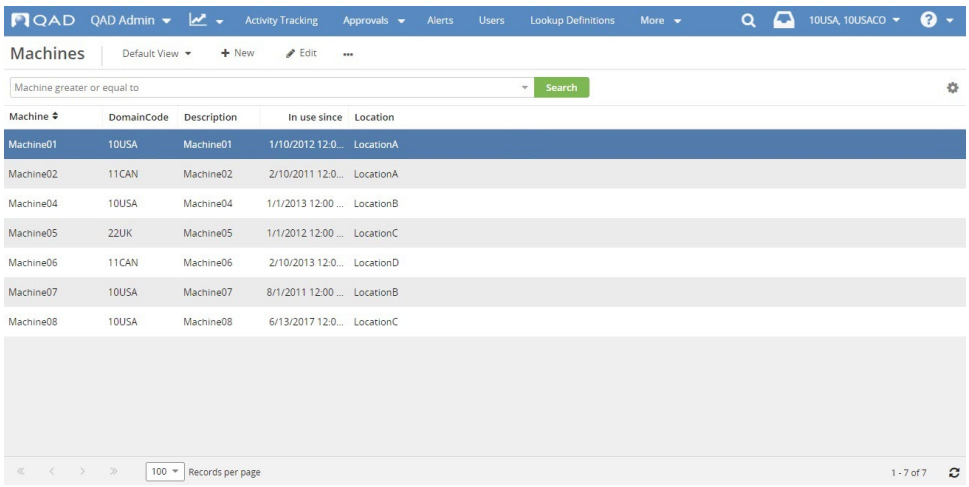
1. See 6 p., there is a non-extension Relationship between the Main BC **Machines** and related BC **Departments**.
2. You can create a Hybrid Browse with BC Browse **for the Machines BC:** only its fields can be included, but without related fields from **Departments**. That means when BC Browse is created on "1" side (Main BC), it can define only its own fields.



3. Related fields are not displayed in the Selector and cannot be added to BC Browse.

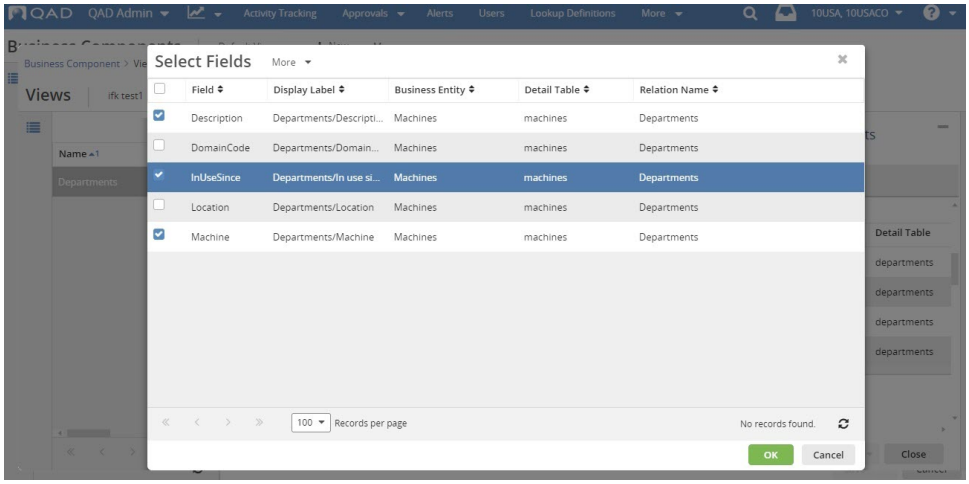


4. Here you can see the **Machines** BC Browse and its fields without related fields.

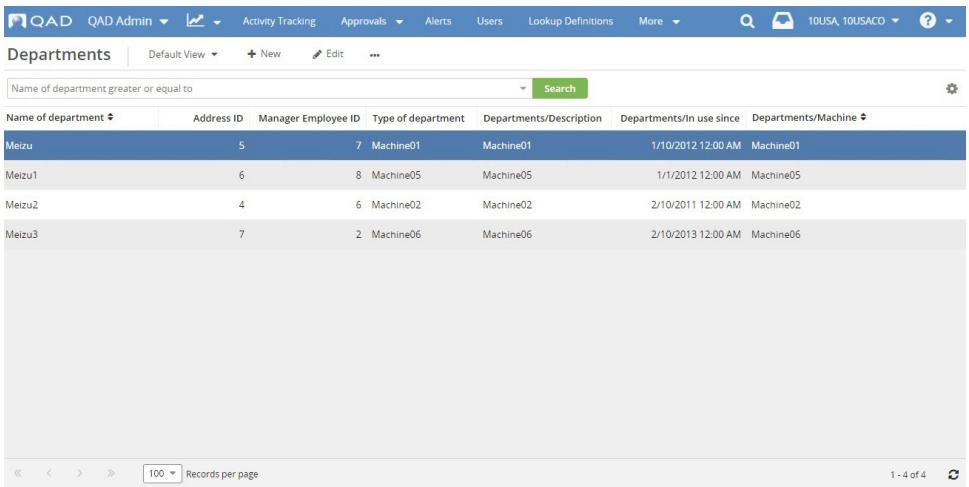


4- BC Browse for Virtual Business Component related to another Virtual Business Component (Traditional (non-extension) Relationship with N-1 cardinality)

1. For the previous relation, it can be considered that it is N-1 cardinality from the **Departments** BC as "N" side.
2. You can create a Hybrid Browse with BC Browse **for the Departments BC:** all fields from the Main and related Business Components can be included. That means when BC Browse is created on "N" side, it can define its own and related fields.
3. Related fields can be selected only in the Selector and after that can be added to the **Columns** grid. To do this, click the **+Select** button and select fields you need in the **Select Fields** pop-up window that opens.
4. For non-extension Relationship, the "Primary Key" field from the related BC can be selected and added to the BC Browse.



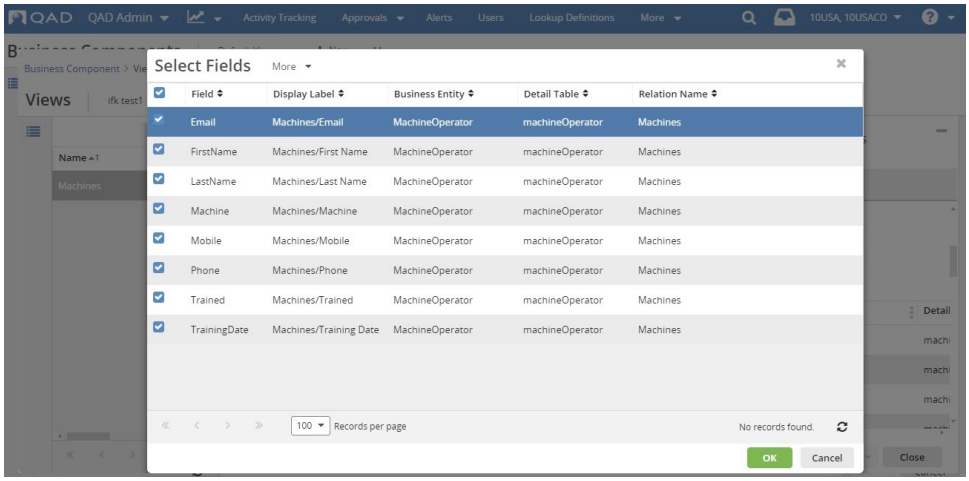
5. Confirm the saving.
6. Keep in mind that only fields that belong to the deployed Business Component will be displayed in the Preview of BC Browse, so you need to deploy both the Main and related BC if you want to observe all the fields you have included.
7. Here you can see fields from the **Departments** BC and related fields which are displayed with their appropriate Relation Name. The related Primary Key "Departments/Machine" can be seen.



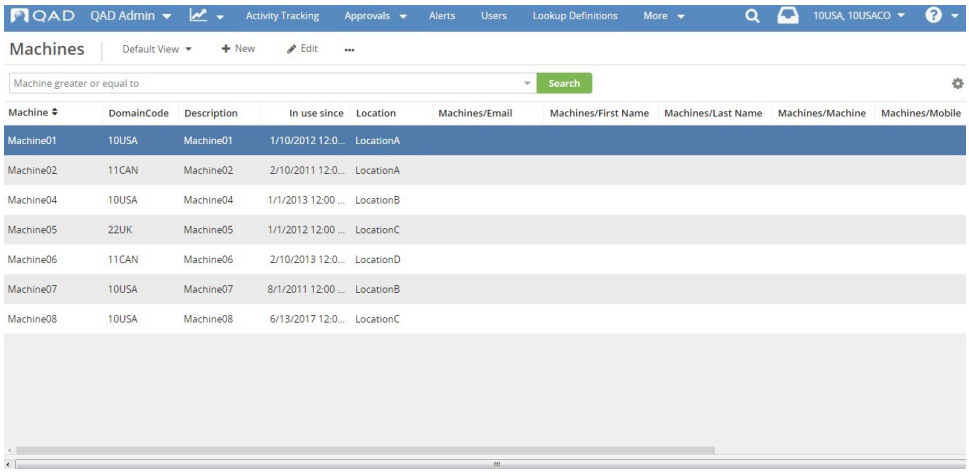
5- BC Browse for Extended Virtual Business Component (Extension Relationship with 1-1 cardinality)

It is forbidden to create a Hybrid View with BC Browse for the embedded Business Component, but it is allowed for the extended Business Component.

1. See 7 p., there is an Extension Relationship between the BC **Machines** and embedded BC **MachineOperator**.
2. You can create a Hybrid View with BC Browse for extended BC **Machines** and all fields from the Main and related Business Components can be included (except the related "Primary Key" field). **For an Extension Relationship, the "Primary Key" field from the related BC cannot be selected and added to the BC Browse.**
3. Related fields can be selected only in the Selector and after that can be added to the **Columns** grid. To do this, click the **+Select** button and select fields you need in the **Select Fields** pop-up window that opens.
4. You can see that the related "Primary Key" field "Employee" is not present and cannot be selected as it was mentioned above.

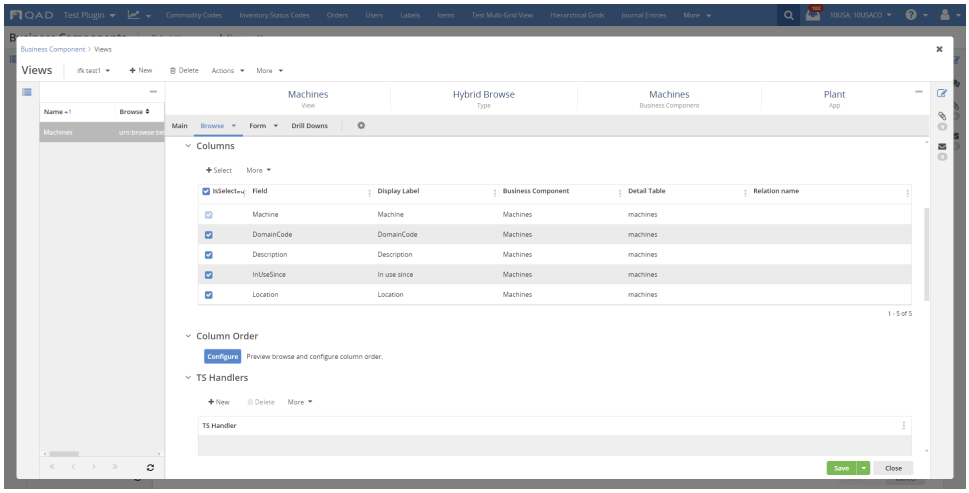


5. Confirm the saving.
6. Only fields that belong to the deployed Business Component will be displayed in the Preview of BC Browse, so you need to deploy both Business Components if you want to observe all the fields you have included.
7. Here you can see fields from the extended BC **Machines** and its related fields which are displayed with the appropriate Relation Name.

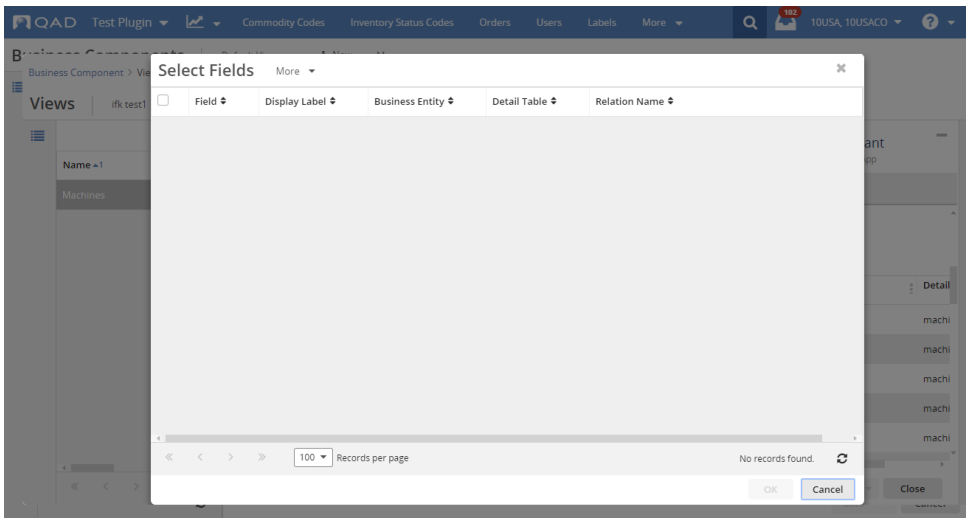


6- BC Browse for Extended Virtual Business Component (Extension Relationship with N-1 cardinality)

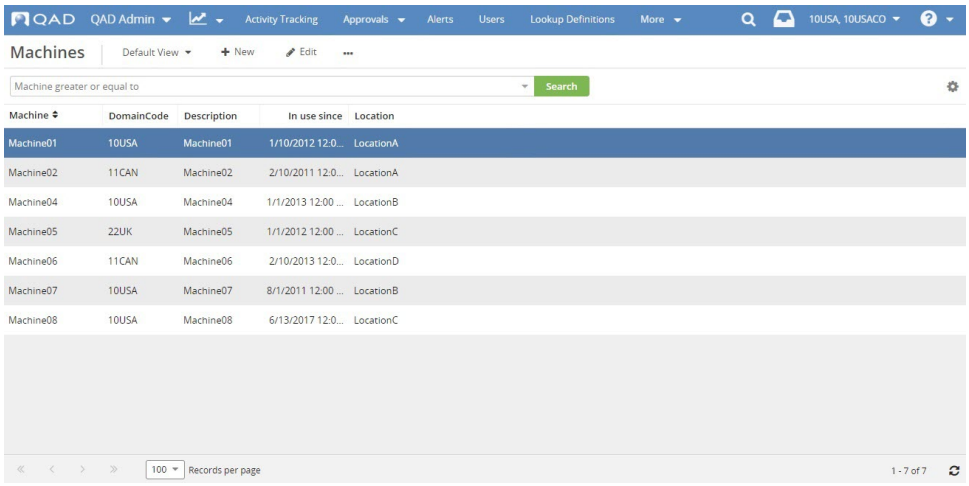
1. See 7 p., there is an Extension Relationship between the BC **Machines** and embedded BC **MachineOperator** wit h cardinality N-1, where the embedded BC is "N" side.
2. As **it is forbidden to create a Hybrid View with BC Browse for the embedded Business Component,** you can create a Hybrid View with BC Browse only for the **Machines** BC, so only its fields can be defined and included to the BC Browse, but without related fields from **MachineOperator**.



3. Related fields are not displayed in the Selector and cannot be added to the BC Browse.



4. Confirm the saving.
5. Here you can see the **Machines** BC Browse and its own fields, but without related extending fields.



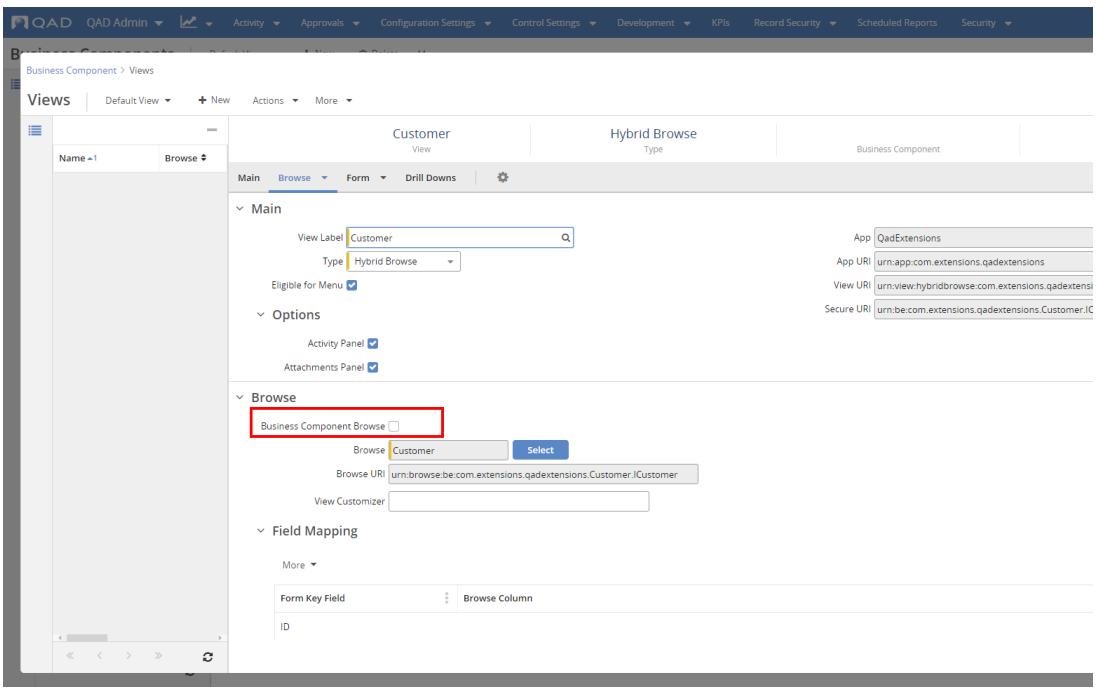
5.1. Create Standard Browse

Introduction

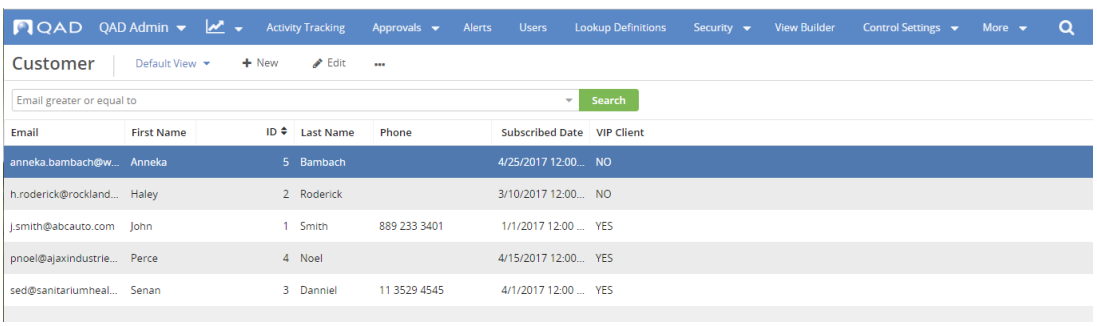
Standard Browse is a browse that shows all fields of Business Component.

Screen Shots

In the Form Builder, you can define a View with the Standard Browse.



Browse page for a Business Component with Standard Browse:



UI Item Details

The following describes the different items on the UI:

1. **Business Component Browse**. Switch between the browse types; for the Standard Browse, this checkbox must be unselected.
2. **View Label**. Displays the label for a View.

3. **Browse**. Click the **Select** button to select the browse resource.

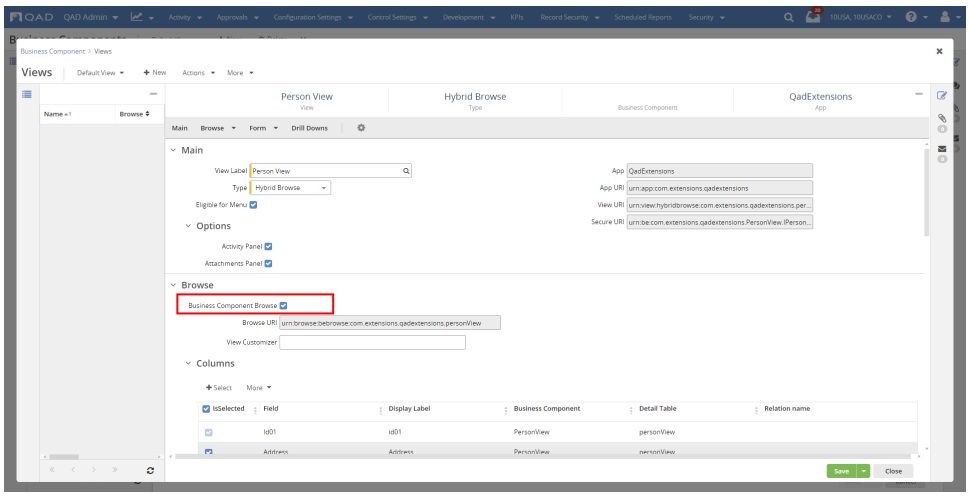
5.2. Create Custom Browse

Introduction

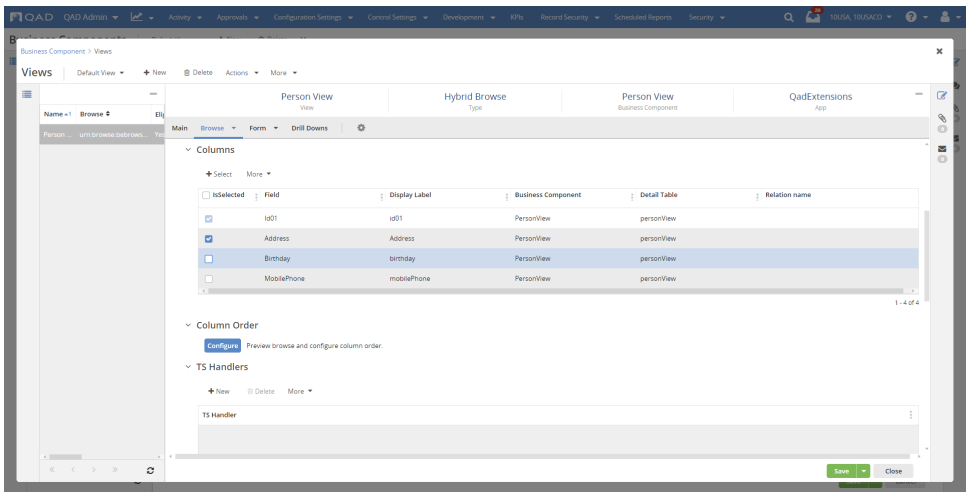
Custom Browse is a definition of a browse and contains information about which fields from the BC and relations can be shown on the browse page.

Screen Shots

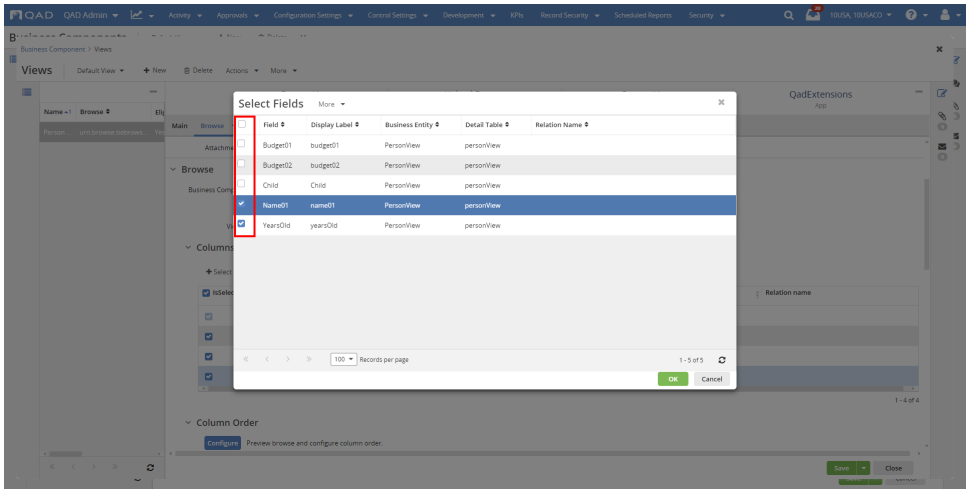
1. In the Form Builder, you can define a View with the Custom Browse support. For a newly created View, the list of fields is taken from the Business Component Form.



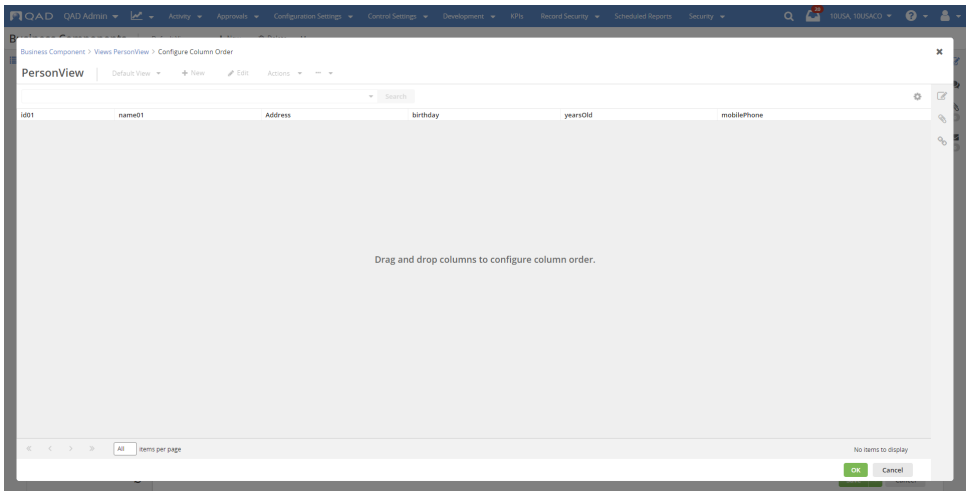
2. To define or modify the list of fields, select the fields in the grid.



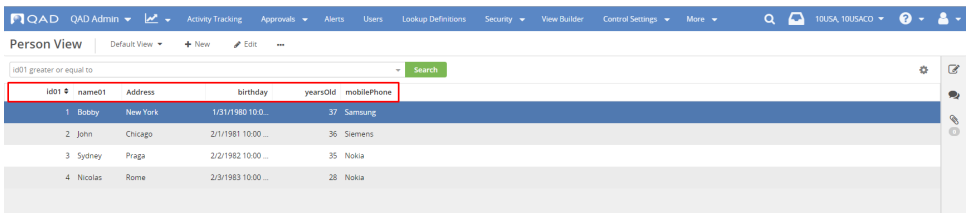
3. To add new fields, click the **+Select** button, select the fields, and then click **OK**.



4. To reorder the fields, click the **Configure** button, drag-and-drop the fields as needed, and then click **OK**.



After these changes, we will get the browse page:



UI Item Details

The following describes the different items on the UI:

1. **Business Component Browse**. Switch between the browse types; for the Custom Browse, this checkbox must be selected.
2. **View Label**. Displays the label for a View.
3. **+Select**. Click the button to add unselected fields to the browse.
4. **Configure**. Click the button to reorder the fields of a browse.

6. Create a relation between two business components

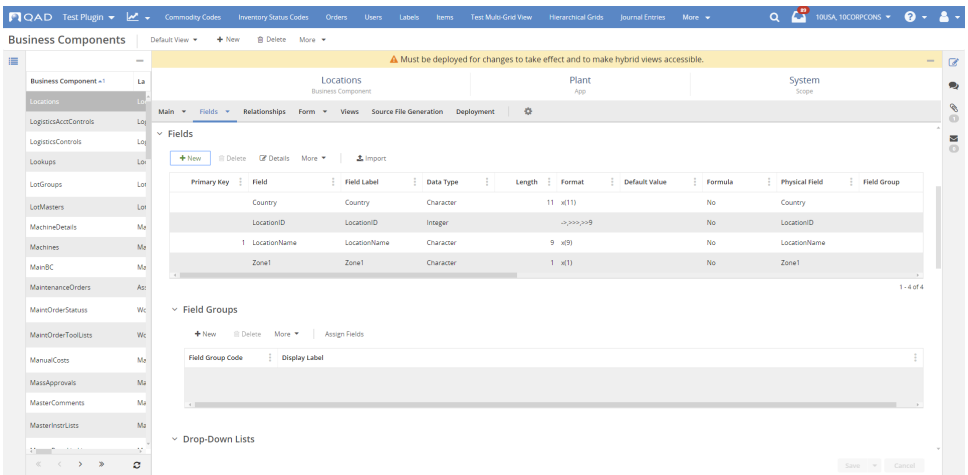
There are 3 variants of cardinality for traditional (non-extension) Relationships between Business Components:

1- Cardinality: one-to-one

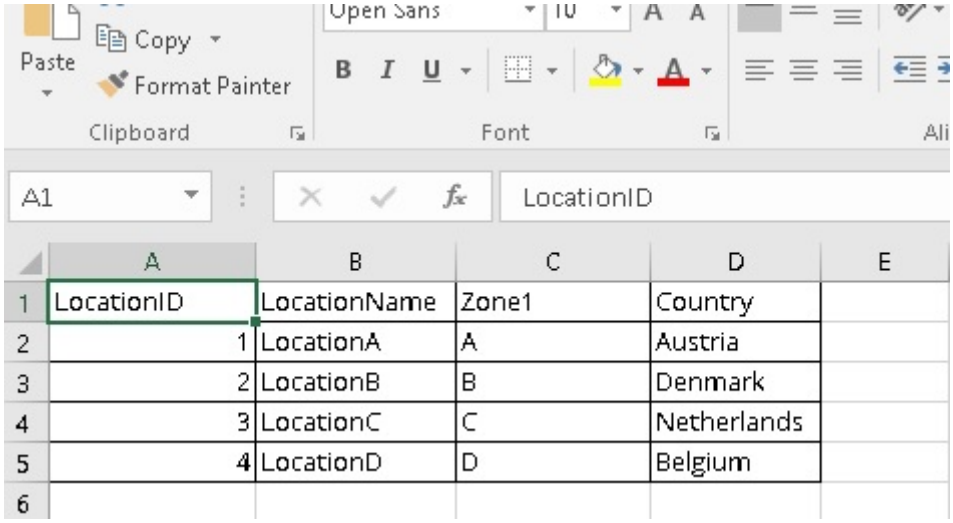
Let's create a Business Component named **Locations** and relate it to the **Machines** BC that was created in our previous examples.
One Machine is located in one appropriate Location. So it is the one-to-one cardinality.

Step 1 - Create the virtual BC which should be related to the Main Virtual BC

1. Navigate to **Business Components** from the menu search.
2. Click the **New** toolbar button. The "App URI" is already initialized and equals the active App in the system.
3. Navigate to the **Main** panel and define the "Business Component" name, "Business Component Label", "Physical Table", "Description", and "Scope". The "Business Component URI" will be initialized automatically depending on the "Business Component" name.
4. Navigate to the **Fields** panel and create the fields for Business Component. At least one field should be marked as "Primary Key" (set value "1" to "Primary Key").
   In 2 p., a few ways for Fields creation were mentioned.
5. The "LocationName" field will be a Primary Key.



6. The example below uses import from the previously prepared excel file:

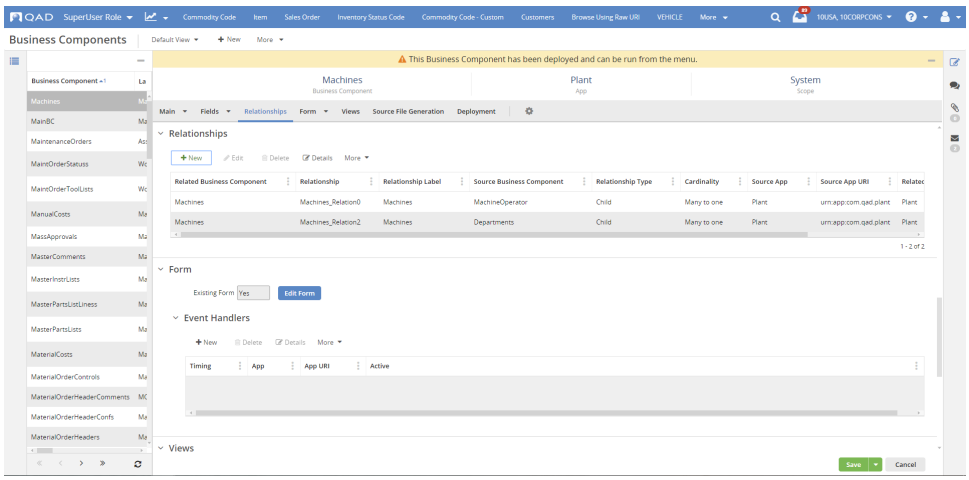


	A	B	C	D	E
1	LocationID	LocationName	Zone1	Country	
2	1	LocationA	A	Austria	
3	2	LocationB	B	Denmark	
4	3	LocationC	C	Netherlands	
5	4	LocationD	D	Belgium	
6					

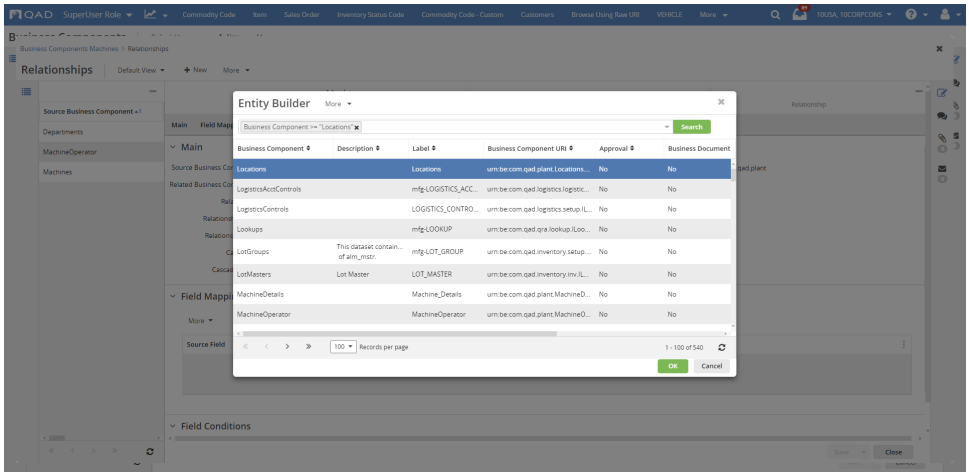
7. If you want to create Relation with one-to-one cardinality, the Primary Key field in the related BC should match the Primary Key field in Main BC **by quantity/order and datatype.** In other case, the cardinality many-to-one will be set.
8. Save the Business Component.

Step 2 - Create the Relation between the Main Virtual BC Machines and virtual BC Locations with one-to-one cardinality

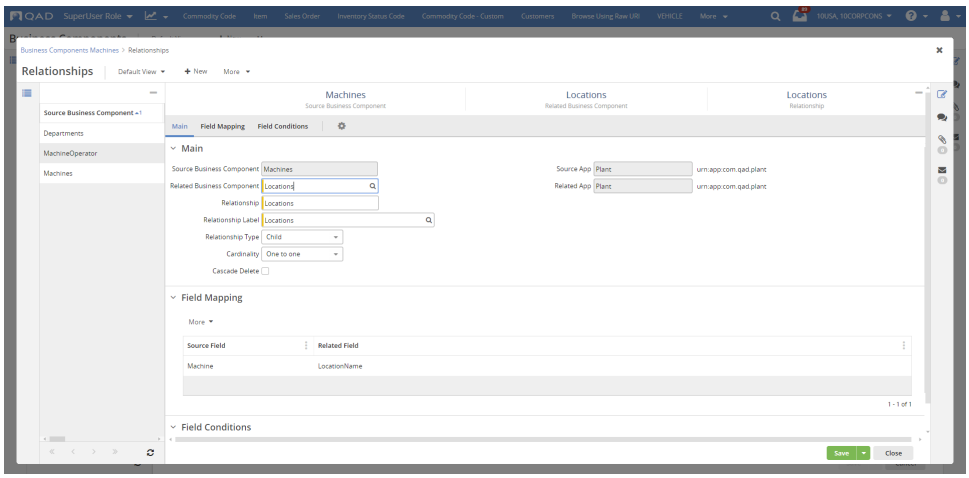
1. Find the Main Virtual BC **Machines** using the quick search.
2. Click the **Edit** toolbar button. Maintenance View for the **Machines** BC will open.



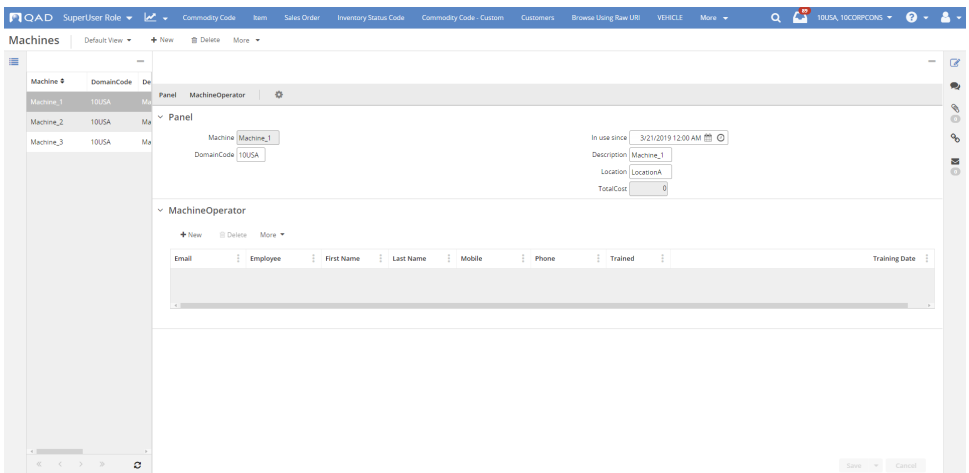
3. Expand the **Relationships** panel in the **Business Components** screen and click the **New** button.
4. Click the lookup in the "Related Business Component" field and in browse set the criteria for the virtual BC that is aimed to be related with the Main Virtual BC **Machines**. In our case, it is a newly created BC **Locations**. Then click the **Search** button.



5. Confirm the selected BC by clicking **OK**.
6. If the "Primary Key" field in the related BC matches the "Primary Key" field in the Main BC **by quantity/order and datatype**, the cardinality is set automatically as one-to-one (with **Machines** BC Primary Key in the Source Field and **Locations** BC Primary Key in the Related Field).
7. Save the Relationship.



8. Confirm the new created Relation by clicking the **Save** button.
9. After that, build Form and Hybrid View for the related BC **Locations** before its deploy.
10. You can create a Hybrid View with BC Browse and all fields from the Main and related Business Components can be included (case with one-to-one cardinality).
11. Deploy the related Business Component **Locations**. You can use data import for deploy to fill it with data.
12. The opened Preview for the Main or related BCs can display all included fields if they were set into BC Browse.
13. You can see only the Main Business Component Panel with fields, but not the related one.

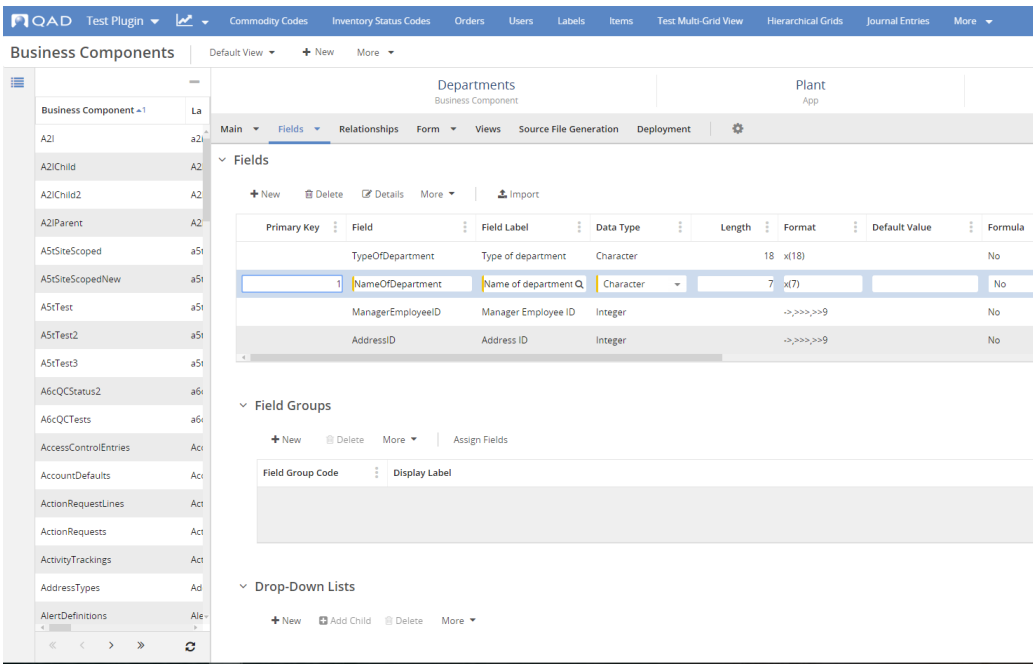


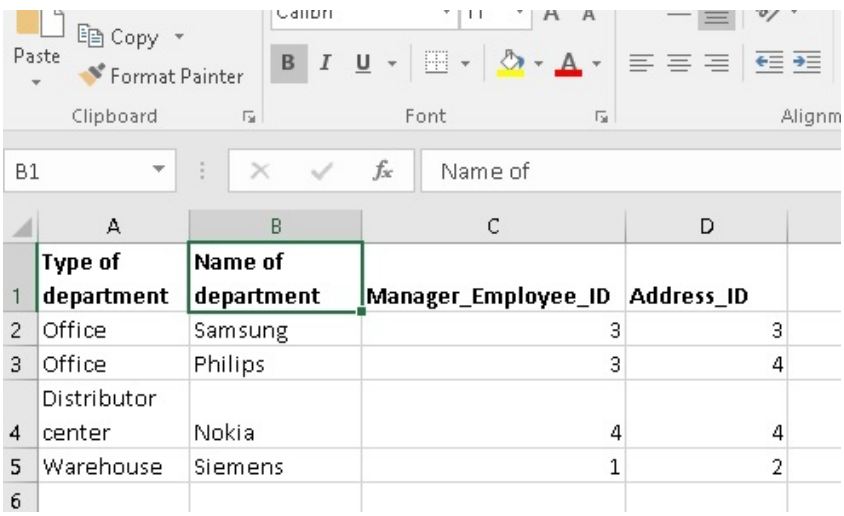
2- Cardinality: one-to-many

Let's look at the situation when one-to-many cardinality is used: one machine serves for several Departments.
At start, we need to create the **Departments** Business Component and repeat the steps mentioned above.

Step 1 - Create the virtual BC which should be related to the Main Virtual BC

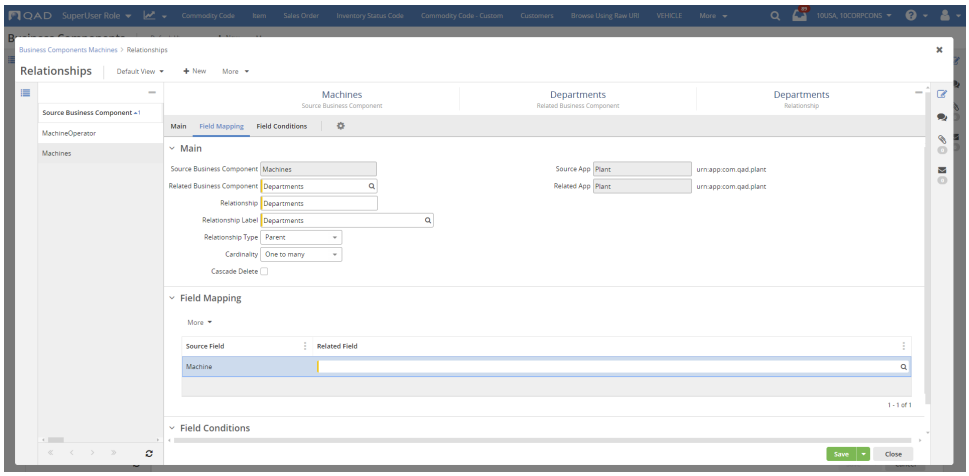
1. Navigate to **Business Components** from the menu search.
2. Click the **New** toolbar button. The "App URI" is already initialized and equals the active App in the system.
3. Navigate to the **Main** panel and define the "Business Component" name, "Business Component Label", "Physical Table", "Description", and "Scope". The "Business Component URI" will be initialized automatically depending on the "Business Component" name.
4. Navigate to the **Fields** panel and create the fields for Business Component. At least one field should be marked as "Primary Key" (set value "1" to "Primary Key").
5. Here we have a unique field "Name of department"- it is our Primary Key, and the "Type of department" will be a Foreign Key.
6. In 2 p., a few ways for Fields creation were mentioned. The example on the screen shot below uses import from previously prepared excel file.
7. Save Business Component.



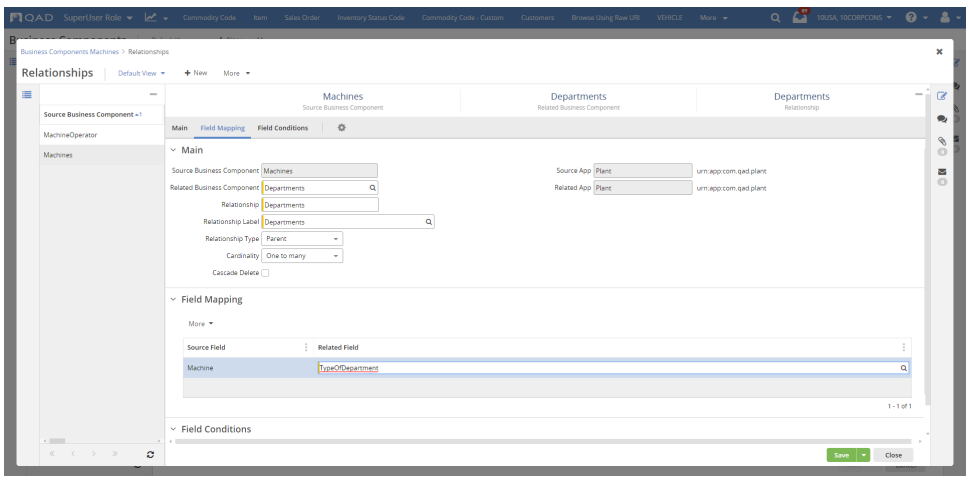


Step 2 - Create the Relation between the Main Virtual BC Machines and virtual BC Departments with 1-N cardinality

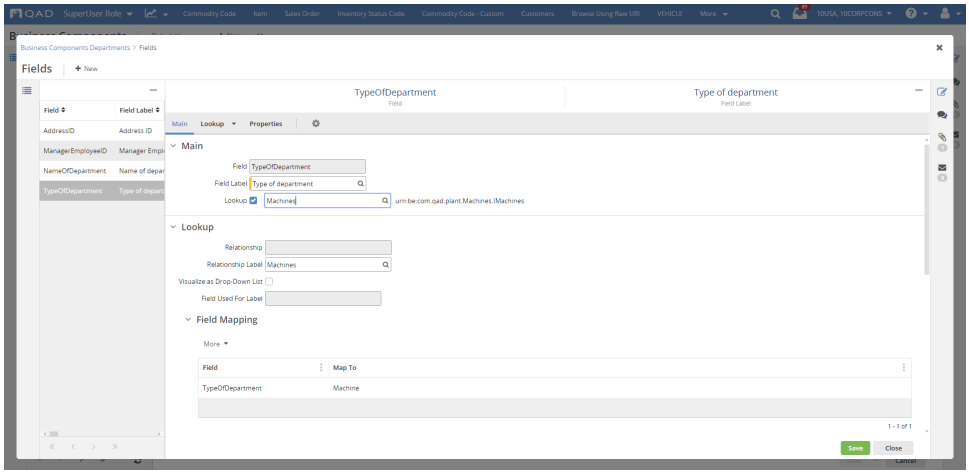
1. Find the Main Virtual BC **Machines** using the quick search.
2. Click the **Edit** toolbar button. Maintenance View for the BC **Machines** will open.
3. Expand the **Relationships** panel in the **Business Components** screen and click the **New** button.
4. Click the lookup in the "Related Business Component" field and in browse set the criteria for the virtual BC that is aimed to be related with the Main Virtual BC **Machines**. In our case, it is a newly created BC **Departments**. Then click the **Search** button.
5. Confirm the selected BC by clicking **OK**.
6. If the "Primary Key" field in the related BC matches the "Primary Key" field in Main BC **by quantity/order and datatype,** the cardinality is set automatically as one-to-one. If you want to create the Relationship with cardinality one-to-many, change it by selecting it in the "Cardinality" drop-down list.



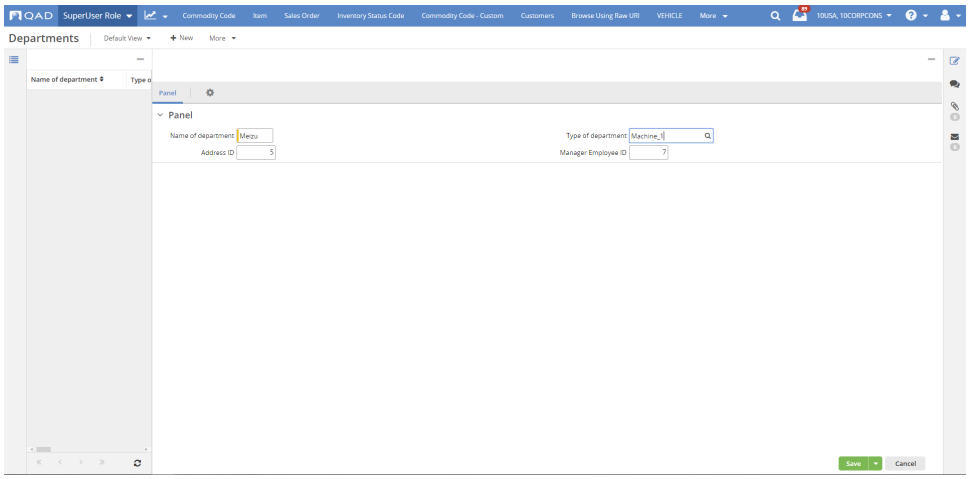
7. Click the lookup in Related Field and select the "Type of department" field as Foreign Key in this Relation.

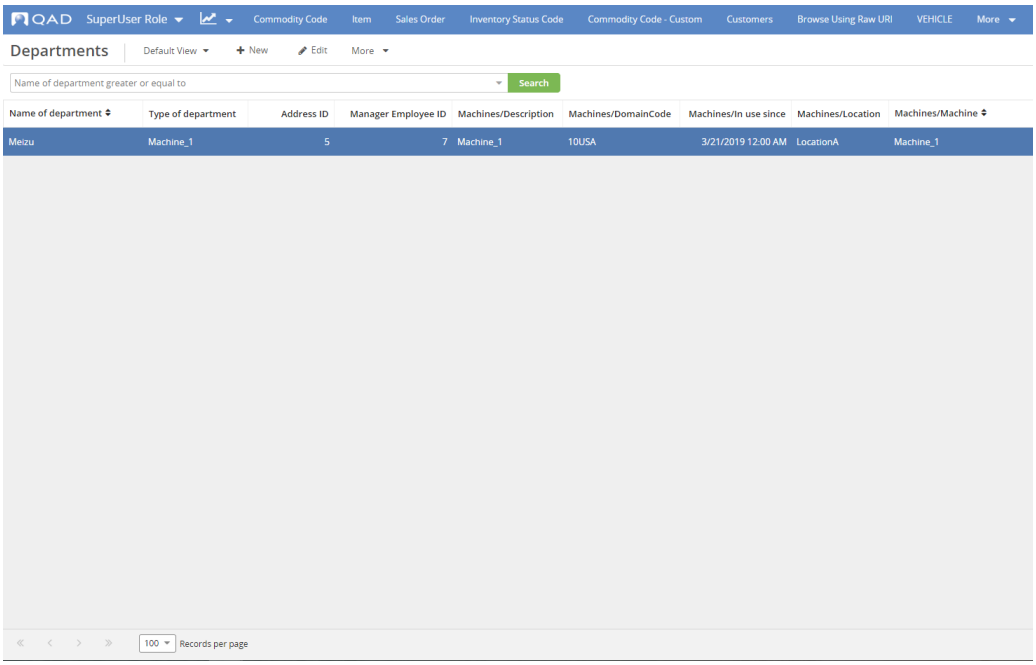


8. After that, save the Relationship.
9. Confirm the new created Relation by clicking the **Save** button.
10. After that, build Form and Hybrid View for the related BC **Departments** before its deploy.
11. You can create Hybrid View with BC Browse **for the related Departments BC:** all related fields can be included (case with one-to-many cardinality where "N" side is BC with BC Browse).
12. You can set the lookup for a specific field on the **Fields** panel by clicking the **Details** button for the field. Then select the Lookup checkbox and click the icon to choose the BC.



13. You can create Hybrid View with BC Browse **for the Machines BC:** only its fields can be included, but without related fields (case with one-to-many cardinality where "1" side is BC with BC Browse).
14. Deploy the related Business Component **Departments**.
15. Open the Preview for the Main or related BCs. Their BC Browses display included fields if they were set into BC Browse.
16. Open Maintenance View for the **Departments** BC by clicking the **New** button. You can see only the Main Business Component panel with fields, but not the related Main **Machines** BC panel.
17. Foreign Key "Type of department" has a lookup; you can fill the data and select the linked record from the **Machines** BC through their Relationship.

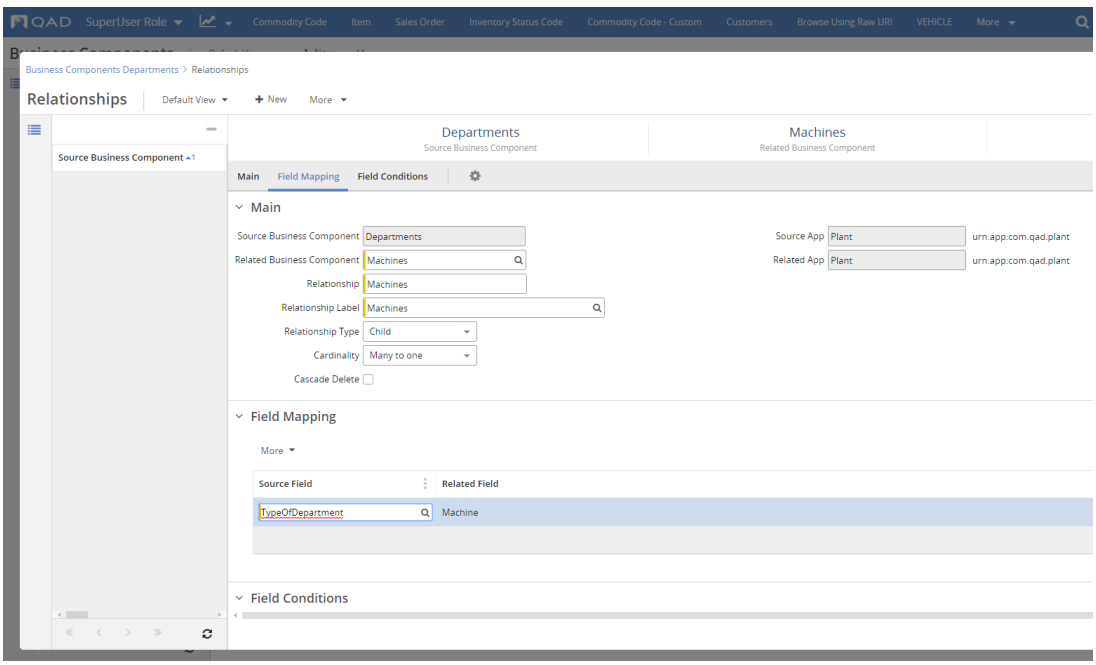




3- Cardinality: many-to-one

The same principle can be used to create a Traditional Relationship with cardinality many-to-one (reverse case when some Types of Departments are served by the one concrete Machine).

The steps are the same for the **Departments** BC as Main BC, only in the **"Cardinality" drop-down list you should select many-to-one and Source Field should be set from lookup as Foreign Key.**

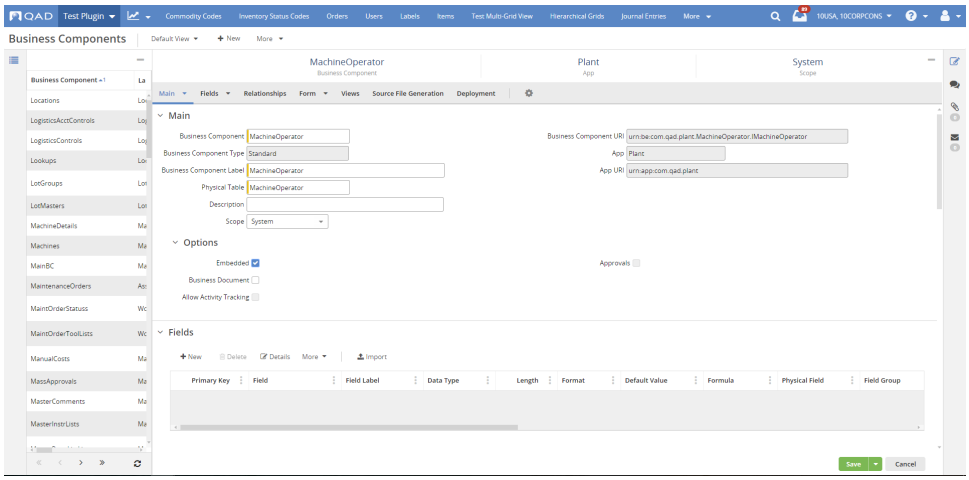


Then, the same scenario with Form, Hybrid View, and Business Component deployment can be used. After that the related BC is ready to be opened and to be filled with records according to its relation.

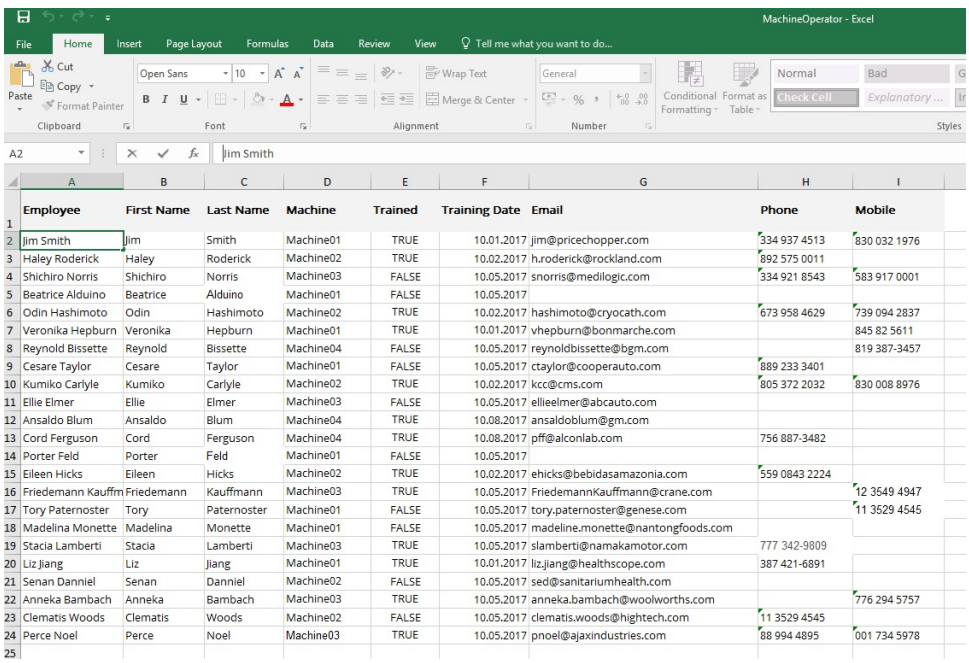
7. Extend a business component with an embedded BC

Step 1 - Create the virtual embedded BC which is aimed to extend the Main Virtual BC

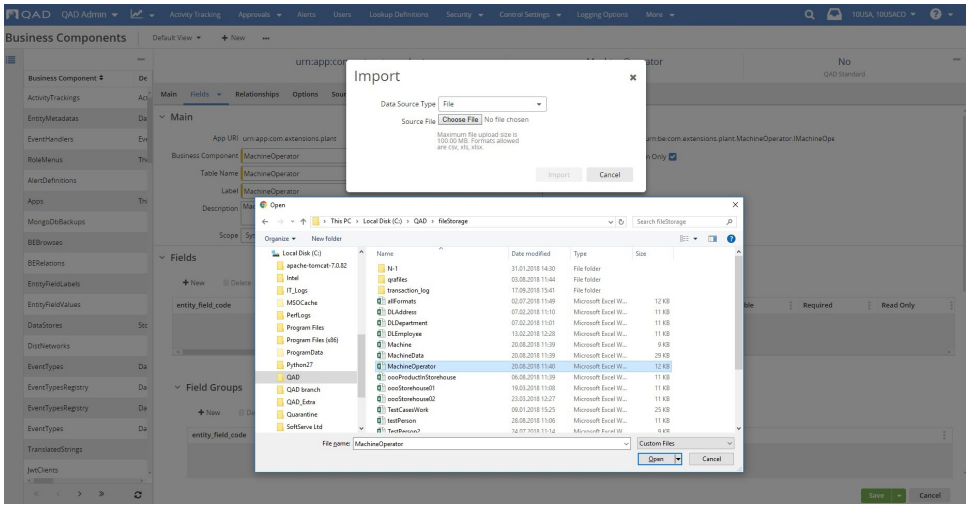
1. Navigate to **Business Components** from the menu search.
2. Click the **New** toolbar button. The "App URI" is already initialized and equals the active App in the system.
3. Navigate to the **Main** panel and define the "Business Component" name, "Business Component Label", "Physical Table", "Description", "Scope", and **select the Embedded checkbox.** The "Business Component URI" will be initialized automatically depending on the "Business Component" name.



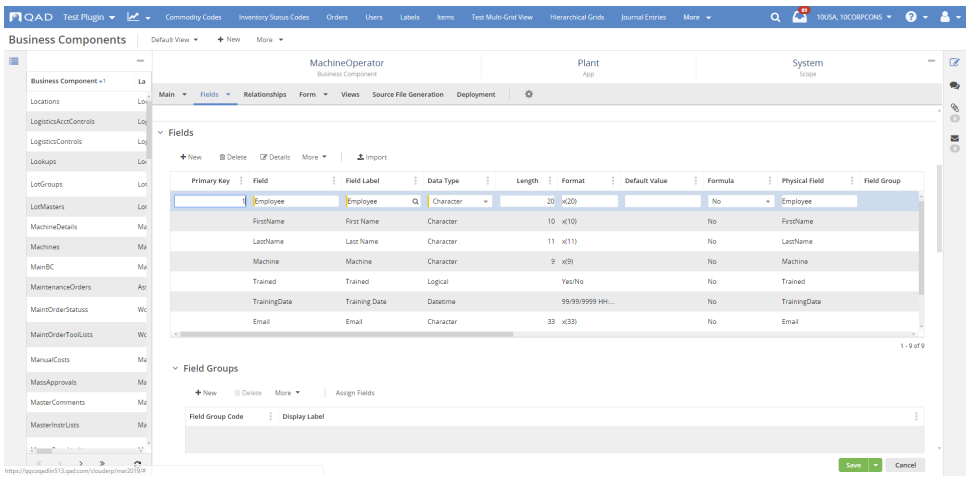
4. Navigate to the **Fields** panel and create the fields for the Business Component. At least one field should be marked as "Primary Key" (set value "1" to "Primary Key").
5. In 2 p., a few ways for Fields creation were mentioned. The example below uses import from previously prepared excel file:



	Employee	First Name	Last Name	Machine	Trained	Training Date	Email	Phone	Mobile
2	Jim Smith	Jim	Smith	Machine01	TRUE	10.01.2017	jim@pricechopper.com	334 937 4513	830 032 1976
3	Haley Roderick	Haley	Roderick	Machine02	TRUE	10.02.2017	h.roderick@rockland.com	892 575 0011	
4	Shichiro Norris	Shichiro	Norris	Machine03	FALSE	10.05.2017	snorris@medilogic.com	334 921 8543	583 917 0001
5	Beatrice Alduino	Beatrice	Alduino	Machine01	FALSE	10.05.2017			
6	Odin Hashimoto	Odin	Hashimoto	Machine02	TRUE	10.02.2017	hashimoto@cryocath.com	673 958 4629	739 094 2837
7	Veronika Hepburn	Veronika	Hepburn	Machine01	TRUE	10.01.2017	vhepburn@bonmarche.com		845 82 5611
8	Reynold Bissette	Reynold	Bissette	Machine04	FALSE	10.05.2017	reynoldbissette@bgm.com		819 387-3457
9	Cesare Taylor	Cesare	Taylor	Machine01	FALSE	10.05.2017	ctaylor@cooperauto.com	889 233 3401	
10	Kumiko Carlyle	Kumiko	Carlyle	Machine02	TRUE	10.02.2017	kcc@cms.com	805 372 2032	830 008 8976
11	Ellie Elmer	Ellie	Elmer	Machine03	FALSE	10.05.2017	ellieelmer@abcauto.com		
12	Ansaldo Blum	Ansaldo	Blum	Machine04	TRUE	10.08.2017	ansaldoblum@gm.com		
13	Cord Ferguson	Cord	Ferguson	Machine04	TRUE	10.08.2017	pff@alconlab.com	756 887-3482	
14	Porter Feld	Porter	Feld	Machine01	FALSE	10.05.2017			
15	Eileen Hicks	Eileen	Hicks	Machine02	TRUE	10.02.2017	ehicks@bebidasamazonia.com	559 0843 2224	
16	Friedemann Kauffm	Friedemann	Kauffmann	Machine03	TRUE	10.05.2017	FriedemannKauffmann@crane.com		12 3549 4947
17	Tory Paternoster	Tory	Paternoster	Machine01	FALSE	10.05.2017	tory.paternoster@genese.com		11 3529 4545
18	Madelina Monette	Madelina	Monette	Machine01	FALSE	10.05.2017	madeline.monette@nantongfoods.com		
19	Stacia Lamberti	Stacia	Lamberti	Machine03	TRUE	10.05.2017	slamberti@namakamotor.com	777 342-9809	
20	Liz Jiang	Liz	Jiang	Machine01	TRUE	10.01.2017	liz.jiang@healthscope.com	387 421-6891	
21	Senan Danniel	Senan	Danniel	Machine02	FALSE	10.05.2017	sed@sanitariumhealth.com		
22	Anneka Bambach	Anneka	Bambach	Machine03	TRUE	10.05.2017	anneka.bambach@woolworths.com		776 294 5757
23	Clematis Woods	Clematis	Woods	Machine02	FALSE	10.05.2017	clematis.woods@hightech.com	11 3529 4545	
24	Perce Noel	Perce	Noel	Machine03	TRUE	10.05.2017	pnoel@ajaxindustries.com	88 994 4895	001 734 5978
25									



6. Pay attention that two types of cardinality are allowed in Extension Relations:
   - 1-1 or N-1
   - If you want to create Extension Relation with cardinality 1-1, the "Primary Key" field in the embedded BC should match the "Primary Key" field in the Main BC **by quantity/order and datatype.** In other case, the cardinality N-1 will be set. Also, remember that length (or format) for the character "Primary Key" field in the embedded BC should be equal/more than the "Primary Key" field in the Main BC.

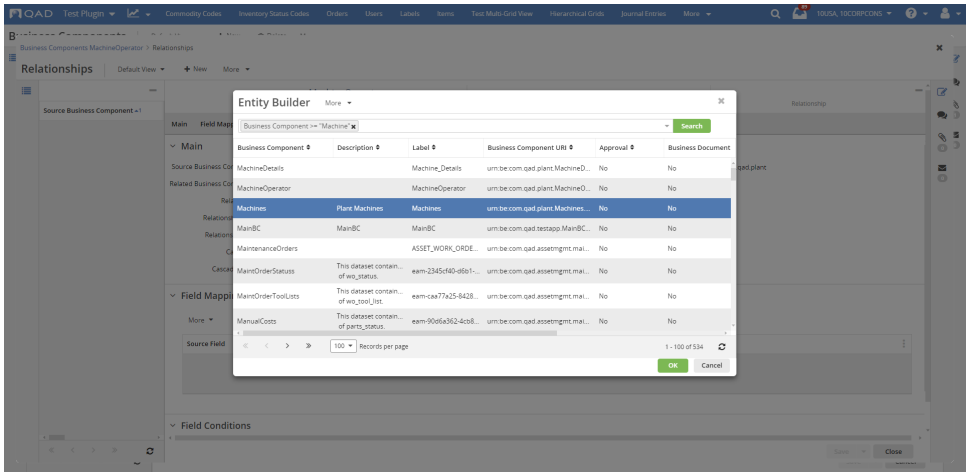


7. Save Business Component.

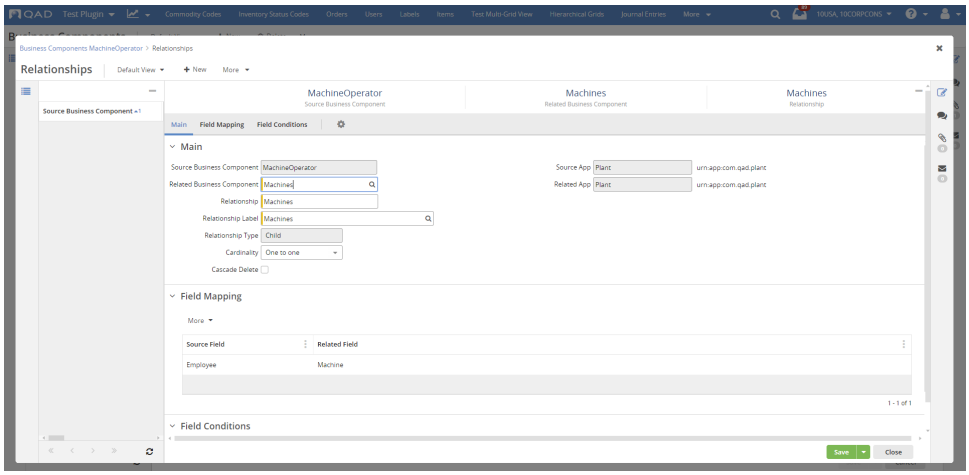
Step 2 - Create the Extension Relation between the embedded BC and the Main Virtual BC

Note: The Extension Relation can be created or edited only on the embedded Business Component side.

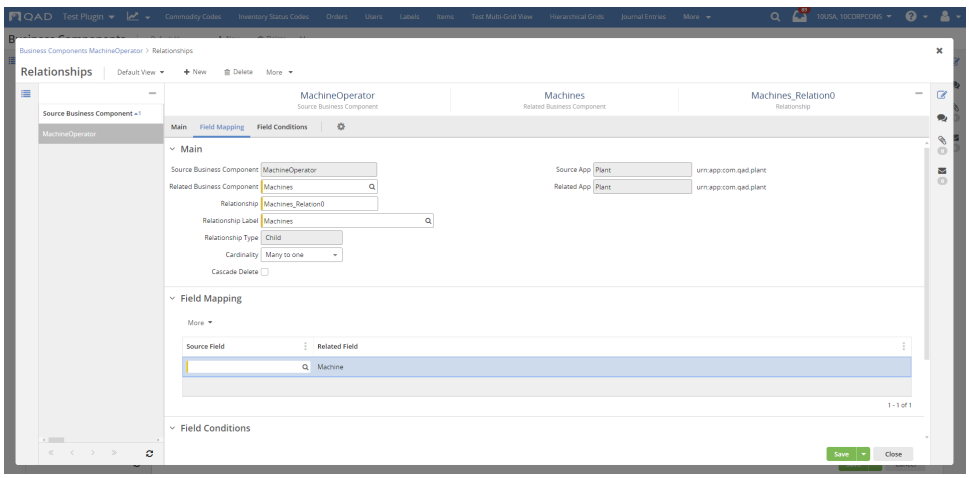
1. Expand the **Relationships** panel in the **Business Components** screen and click the **New** button.
2. Click the lookup in the "Related Business Component" field and in browse set the criteria for the virtual Main BC that is aimed to be extended. Then click the **Search** button.
   Remember that you can't make a Relationship between two embedded Business Components.



3. Confirm the selected BC by clicking **OK**.
4. If the "Primary Key" field in the embedded BC matches the "Primary Key" field in the Main BC **by quantity/order and datatype,** the cardinality is set automatically as one-to-one with the embedded BC Primary Key in the Source Field and the Main BC Primary Key in the Related Field.
5. Save the Extension Relationship.

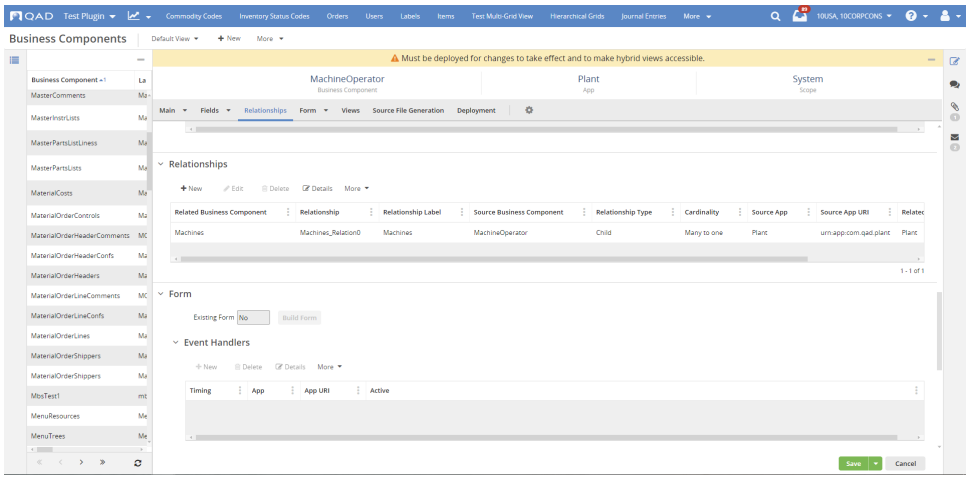


6. If you want to create an Extension Relationship with cardinality many-to-one, change it by selecting it in the "Cardinality" drop-down list.



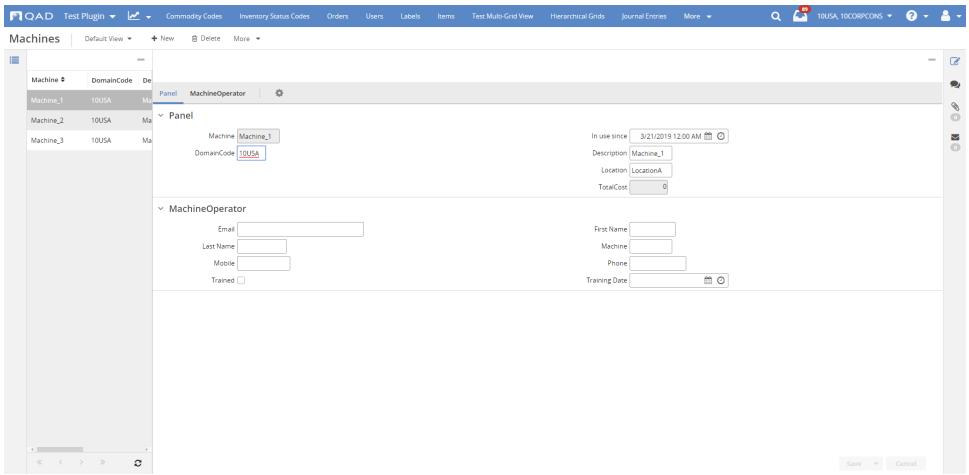
7. Click the lookup in the Source Field and select the field as Foreign Key in this Relation.
   Remember that you can't use the "Primary Key" field as Foreign Key for Extension Relation.
8. After that save the Extension Relationship.

9. Confirm the new created Extension Relation by clicking the **Save** button.

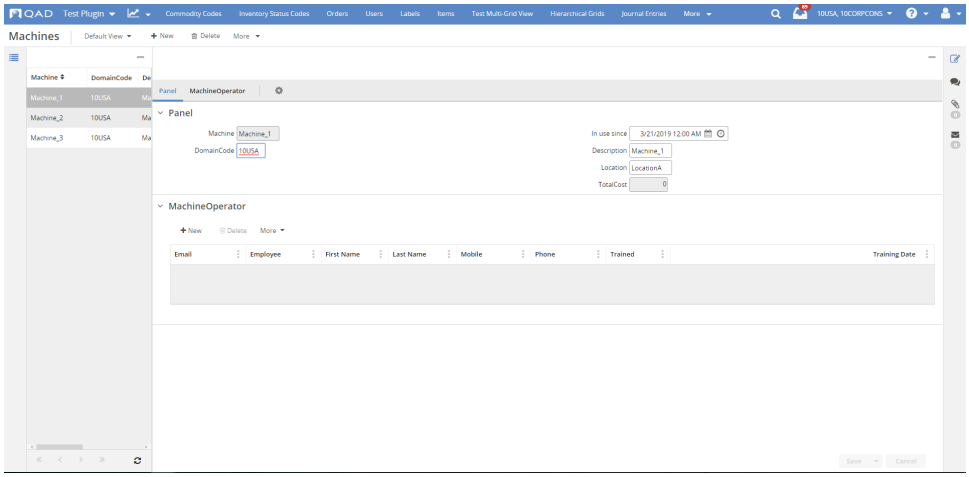


Step 3 - Deploy the embedded BC which is aimed to extend the Main Virtual BC

1. Deploy the embedded Business Component. You can't use data import for deployment of the embedded Busines s Component. The **Import data** checkbox is disabled.
   Remember that you don't need to build a Form and Hybrid View for it because ViewMetadata and ViewResourceMetadata are auto-generated during the embedded Business Component deployment. However, only the embedded Business Component with existing Relationship can be deployed.
2. Open the Preview and Maintenance View for the Main Business Component that was extended.
3. In case of one-to-one cardinality, you can see the Main Business Component Panel and the embedded Business Component Panel with their fields. The "Primary Key" field for the embedded BC shouldn't be seen.



4. In case of many-to-one cardinality, the embedded Business Component Panel's fields are displayed as a grid. The "Foreign Key" field for the embedded BC shouldn't be seen.

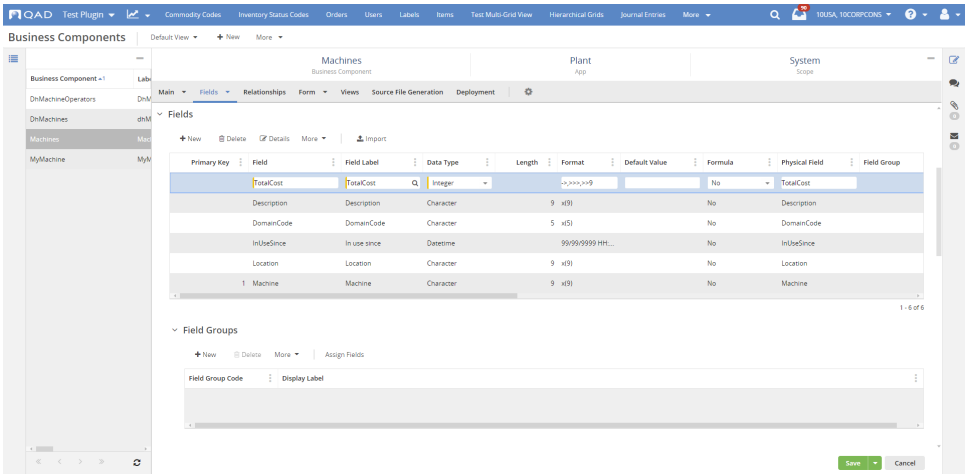


5. The embedded BC is ready for filling with data.

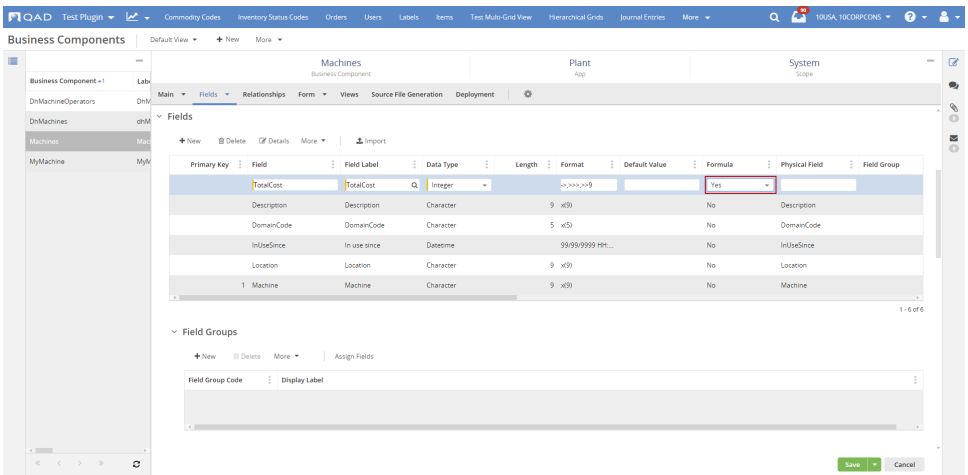
8. Extend a business component with formulas

In order to extend a business component with formulas, you need to do the following:

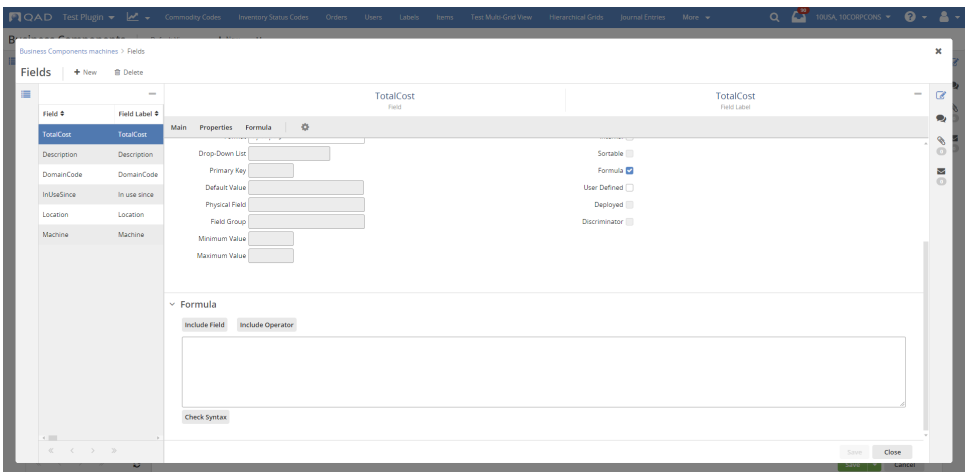
1. Navigate to **Business Components** from the menu search and find the Business Component you want to extend with formula fields.
2. Click the **Edit** toolbar button.
3. On the **Fields** panel, click the **New** button, and then fill all necessary characteristic of the field.



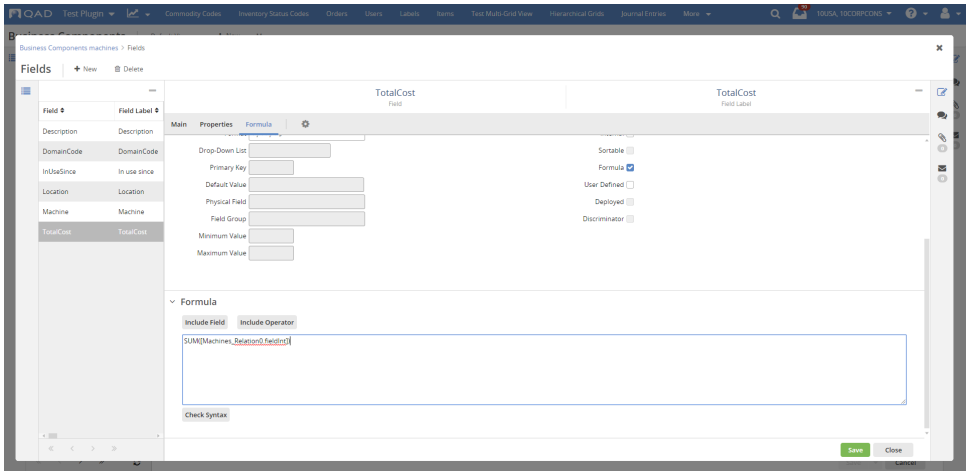
4. Change the value of the **Formula** column to "Yes".



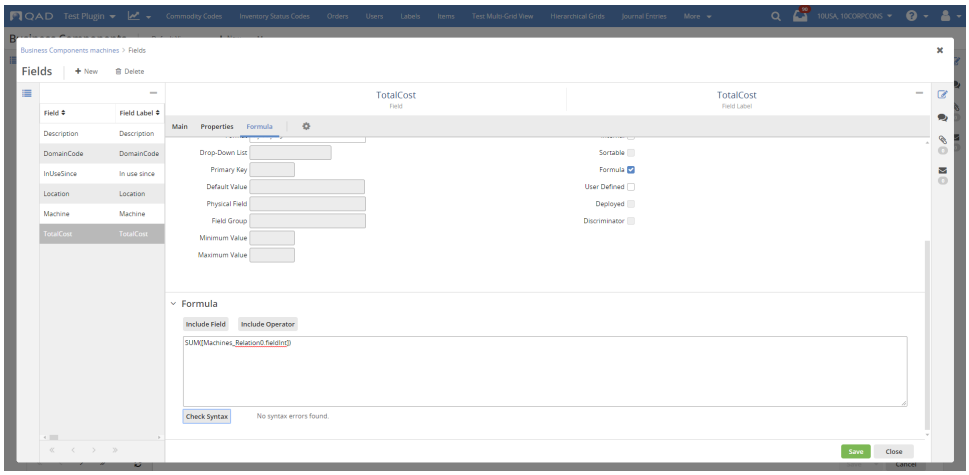
5. On the **Fields** panel, click the **Details** button, and then scroll down to the **Formula** panel.



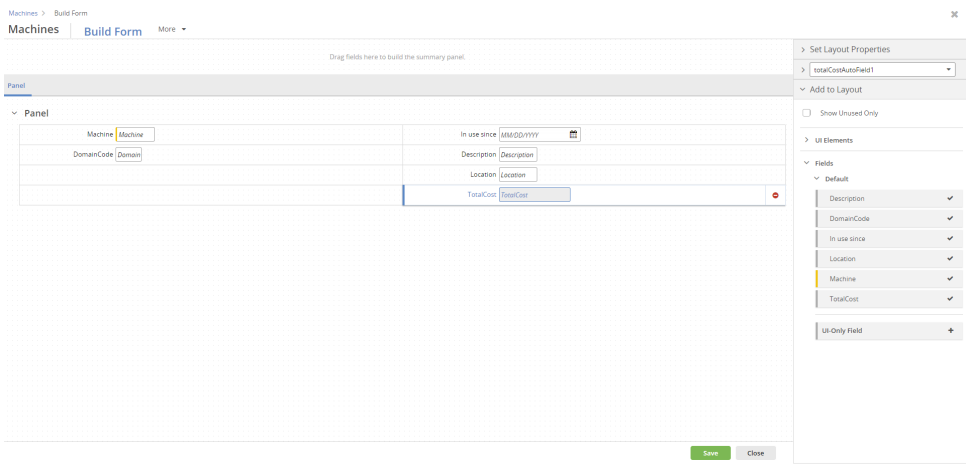
6. Change the formula definition. If you need some other field to be involved in the calculation, use the **Include Field** button. Use the **Include Operator** button to choose operators needed for the calculation.



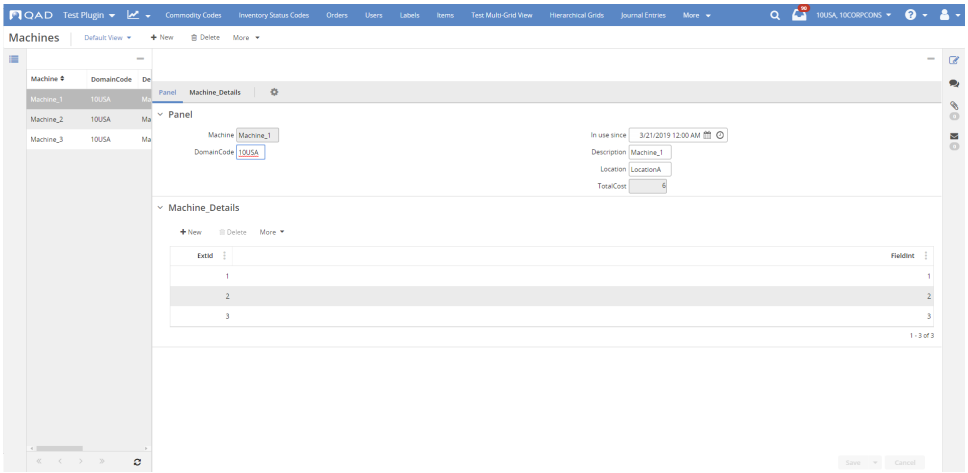
7. Click the **Check Syntax** button to verify if the formula definition is correct. You can save the formula definition only after checking the formula syntax.



8. Click **Save**.
9. Click **Save** again to save the Business Component with the newly added field.
10. Add the field to the Business Component View using the Form Builder.



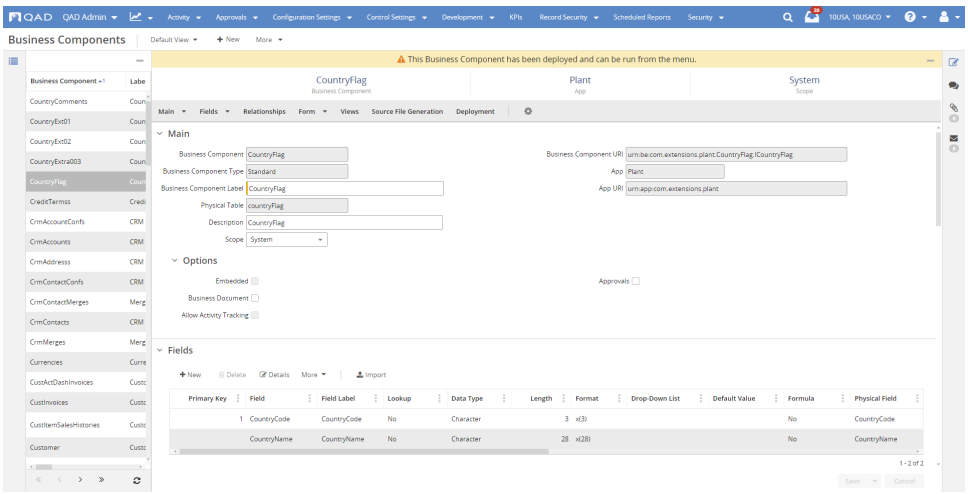
11. Open a Hybrid Browse of the Business Component using the menu search to see the result.


QAD
Test Plugin
Commodity Codes
Inventory Status Codes
Orders
Users
Labels
Items
Test Multi-Grid View
Hierarchical Grids
Journal Entries
More
10USA, 10CORPCONS
Machines
Default View
New
Delete
More
Machine
DomainCode
Machine_1
10USA
Machine_2
10USA
Machine_3
10USA
Panel
Machine_Details
Panel
Machine
Machine_1
DomainCode
10USA
In use since
3/21/2019 12:00 AM
Description
Machine_1
Location
LocationA
TotalCost
6
Machine_Details
New
Delete
More
ExtId
FieldInt
1
1
2
2
3
3
1 - 3 of 3
Save
Cancel

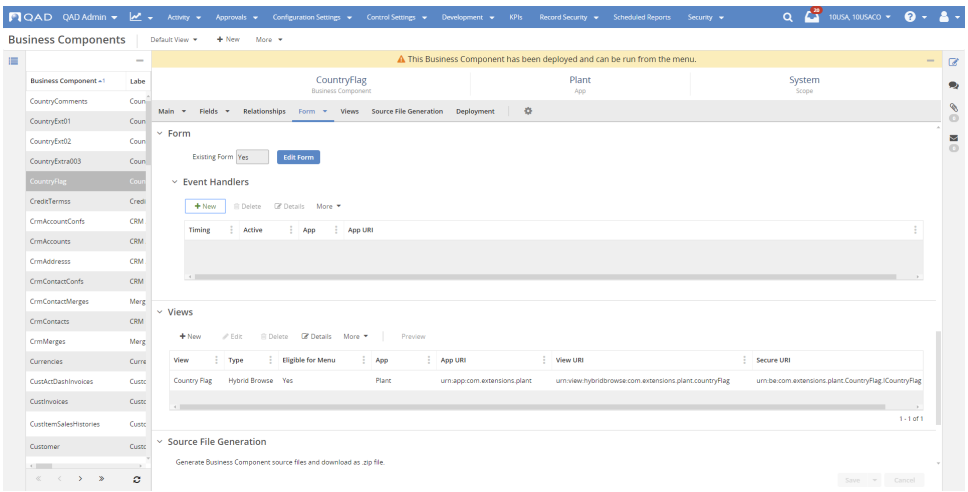
9. Extend a business component UI with event handlers

In order to extend a business component UI with event handlers, you need to do the following:

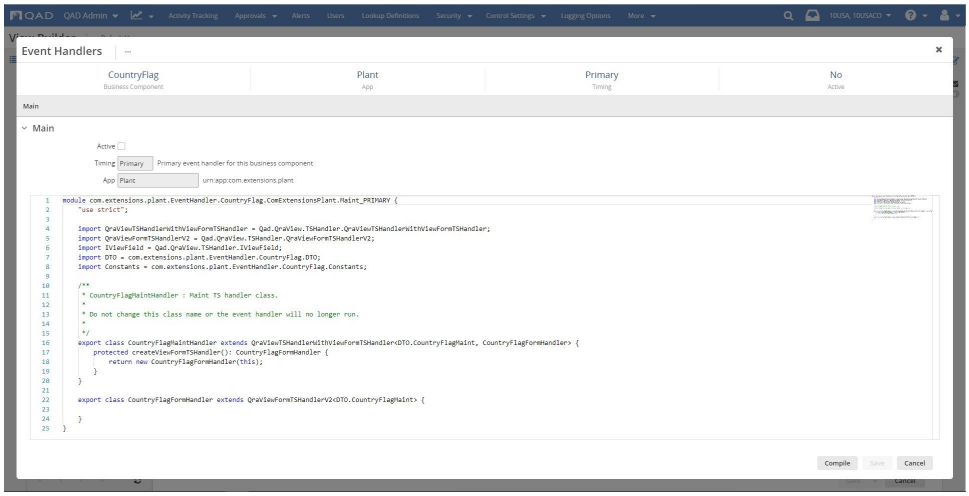
1. Create a Business Component, create a Form and a View/Hybrid Browse for it, and then deploy the Business Component.



2. Go to the **Form** panel of the Business Component created in the 1st step.

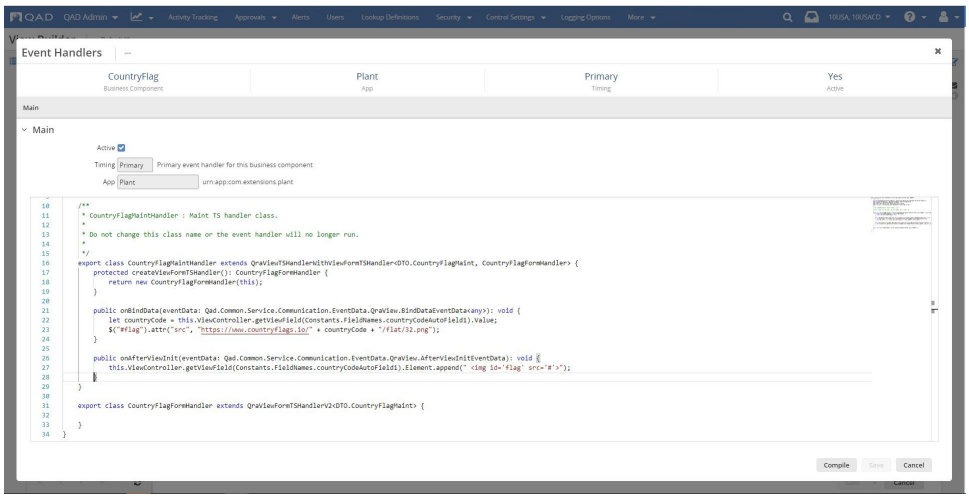


3. Click the **New** button on the **Event Handlers** panel. You will see the following pop-up window with prepared TypeScript template.

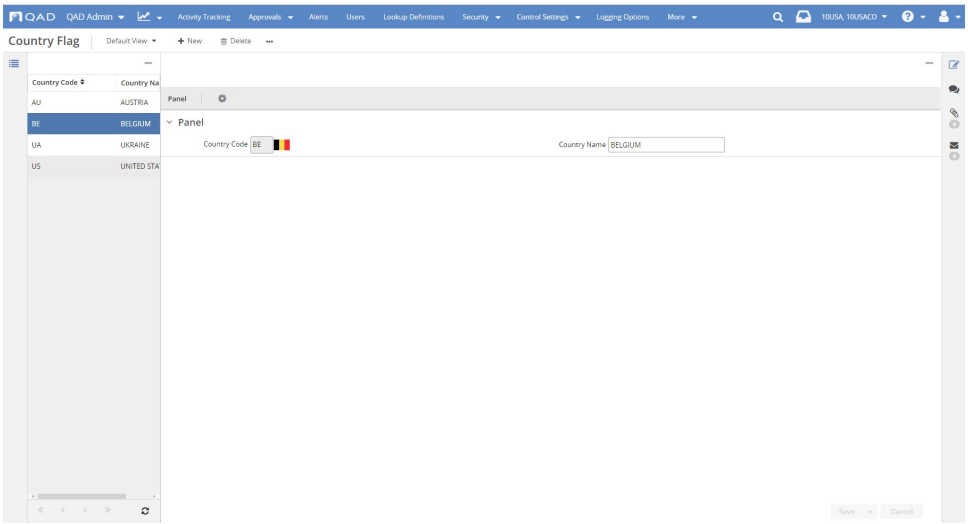


4. Select the **Active** checkbox.
5. Change TS Code according to the expected behavior. For example, paste the following code on line 21:

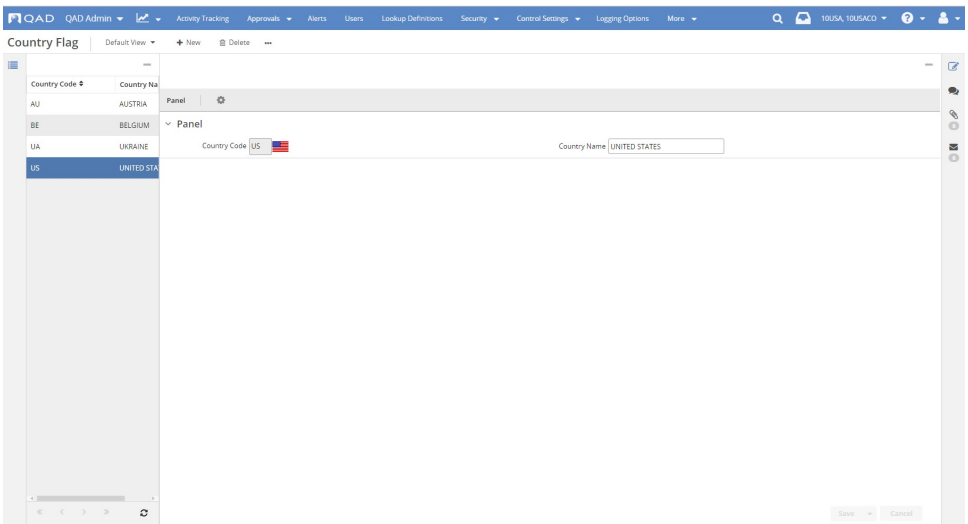
```
public onBindData(eventData: Qad.Common.Service.Communication.EventData.QraView.
BindDataEventData<any>): void {
        let countryCode = this.ViewController.getViewField(Constants.FieldNames.
countryCodeAutoField1).Value;
    $("#flag").attr("src", "https://www.countryflags.io/" + countryCode + "/flat/32.png");
}

public onAfterViewInit(eventData: Qad.Common.Service.Communication.EventData.QraView.
AfterViewInitEventData): void {
        this.ViewController.getViewField(Constants.FieldNames.countryCodeAutoField1).
Element.append(" <img id='flag' src='#'>");
}
```



6. Click the **Compile** button.
7. After the **Compile Successful** notification message is shown, click **Save**.
8. Open a Hybrid Browse of the Business Component.



When you choose another record, the flag changes.



10. Extend a business component with OOABL code

Introduction

The framework supports extending the business logic flow for coded BCs and virtual platform BCs.

The following page explains the mechanism that support all capabilities: Extending the Business Logic

Steps to take for the OOABL coding

Platform Development in YAB

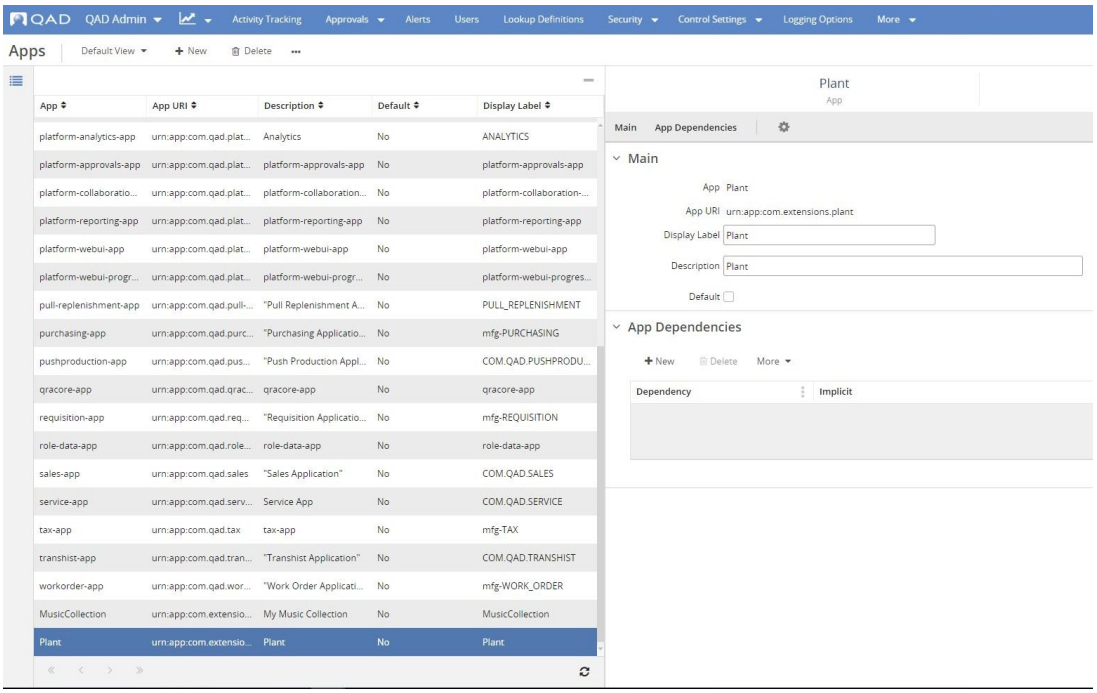
Use cases for extending OOABL code

Service based inheritance

Hook implementations

11. Export app and load it in other environment

All secured resources, created earlier in items 1-9, were created in one App that was set as active in the system.



Export App

To export all resources, do the following:

1. Connect by SSH to the current environment.
2. Navigate to the working folder in the current environment.
3. Run Yab command:
   yab app-export -dir:<*directory to export to*> **-appuri:**<*app uri*>

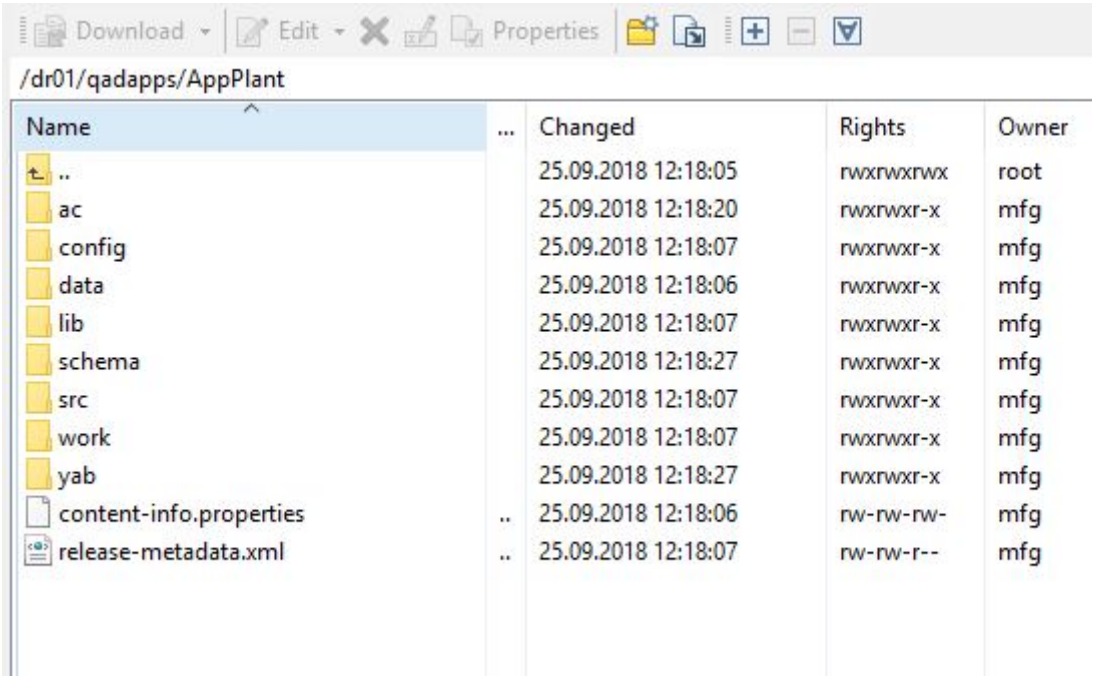
   In our example, that is:
   yab app-export -dir:/dr01/qadapps/AppPlant -appuri:urn:app:com.extension.plant

mfg@public-platform-branch:/dr01/qadapps

```
login as: mfg
mfg@public-platform-branch.qad.com's password:
Last login: Tue Sep 25 02:08:53 2018 from public-platform-branch.qad.com
[mfg@public-platform-branch ~]$ cd /dr01/qadapps/
[mfg@public-platform-branch qadapps]$ ls
sm2  systest  yab
[mfg@public-platform-branch qadapps]$ cd sm2
[mfg@public-platform-branch sm2]$ yab app-export -appuri:urn:app:com.extensions.plant -di

                    app-export (8 tasks)                                [APPLY]
-------------------------------------------------------------------------------
1/8 app-export                                                 OK (0.007 s)
2/8 metadata-all-export                                        OK (1.935 s)
3/8 app-package-metadata-create                                OK (0.098 s)
4/8 app-export-dependency-update                               OK (0.012 s)
5/8 action-center-dashboard-extract                            OK (2.845 s)
6/8 action-center-kpi-files-extract                            OK (10.501 s)
7/8 action-center-kpi-extract                                  OK (6.447 s)
8/8 app-schema-dump                                            OK (0.349 s)
-------------------------------------------------------------------------------

BUILD SUCCESSFUL (22.917 s)
```

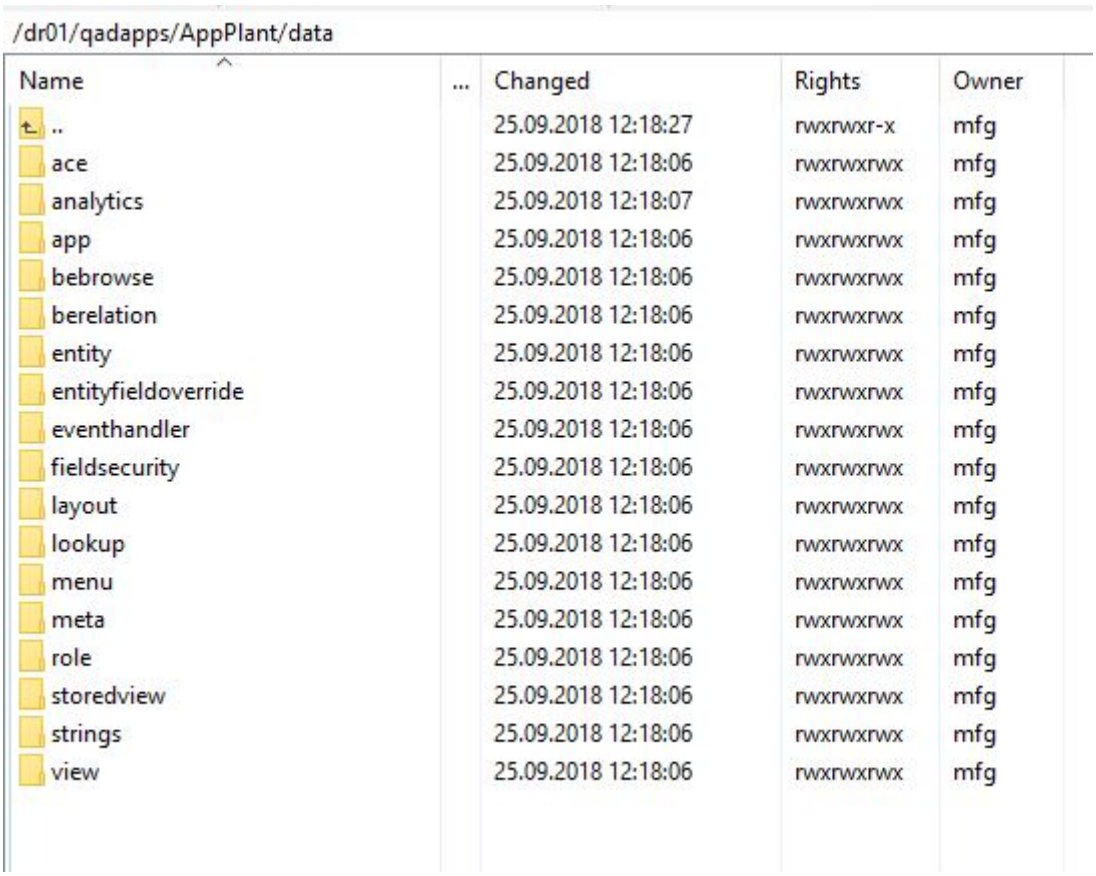
Note: For detailed information, go to the QDN page Platform Development in YAB.

The target folder contains all related information (described in xml files) about the exported App, including earlier created secured resources, schema dump, and other.



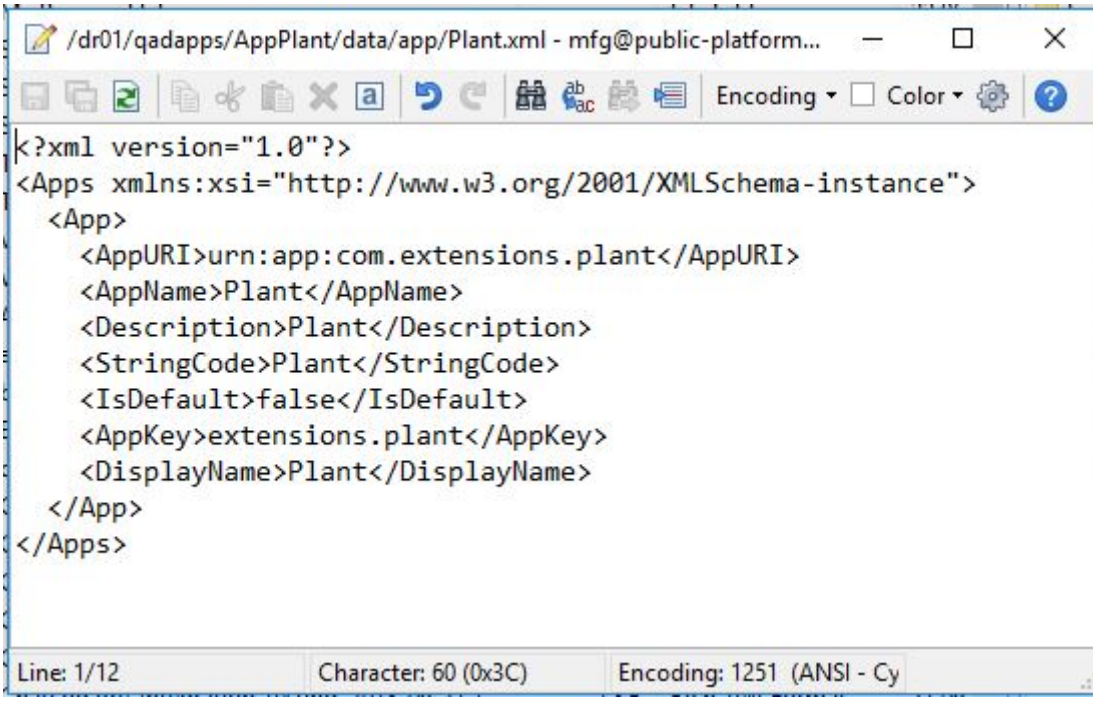
Name	...	Changed	Rights	Owner
..		25.09.2018 12:18:05	rwxrwxrwx	root
ac		25.09.2018 12:18:20	rwxrwxr-x	mfg
config		25.09.2018 12:18:07	rwxrwxr-x	mfg
data		25.09.2018 12:18:06	rwxrwxr-x	mfg
lib		25.09.2018 12:18:07	rwxrwxr-x	mfg
schema		25.09.2018 12:18:27	rwxrwxr-x	mfg
src		25.09.2018 12:18:07	rwxrwxr-x	mfg
work		25.09.2018 12:18:07	rwxrwxr-x	mfg
yab		25.09.2018 12:18:27	rwxrwxr-x	mfg
content-info.properties	..	25.09.2018 12:18:06	rw-rw-rw-	mfg
release-metadata.xml	..	25.09.2018 12:18:07	rw-rw-r--	mfg

The sub-folder "data" contains the earlier created secured resources:

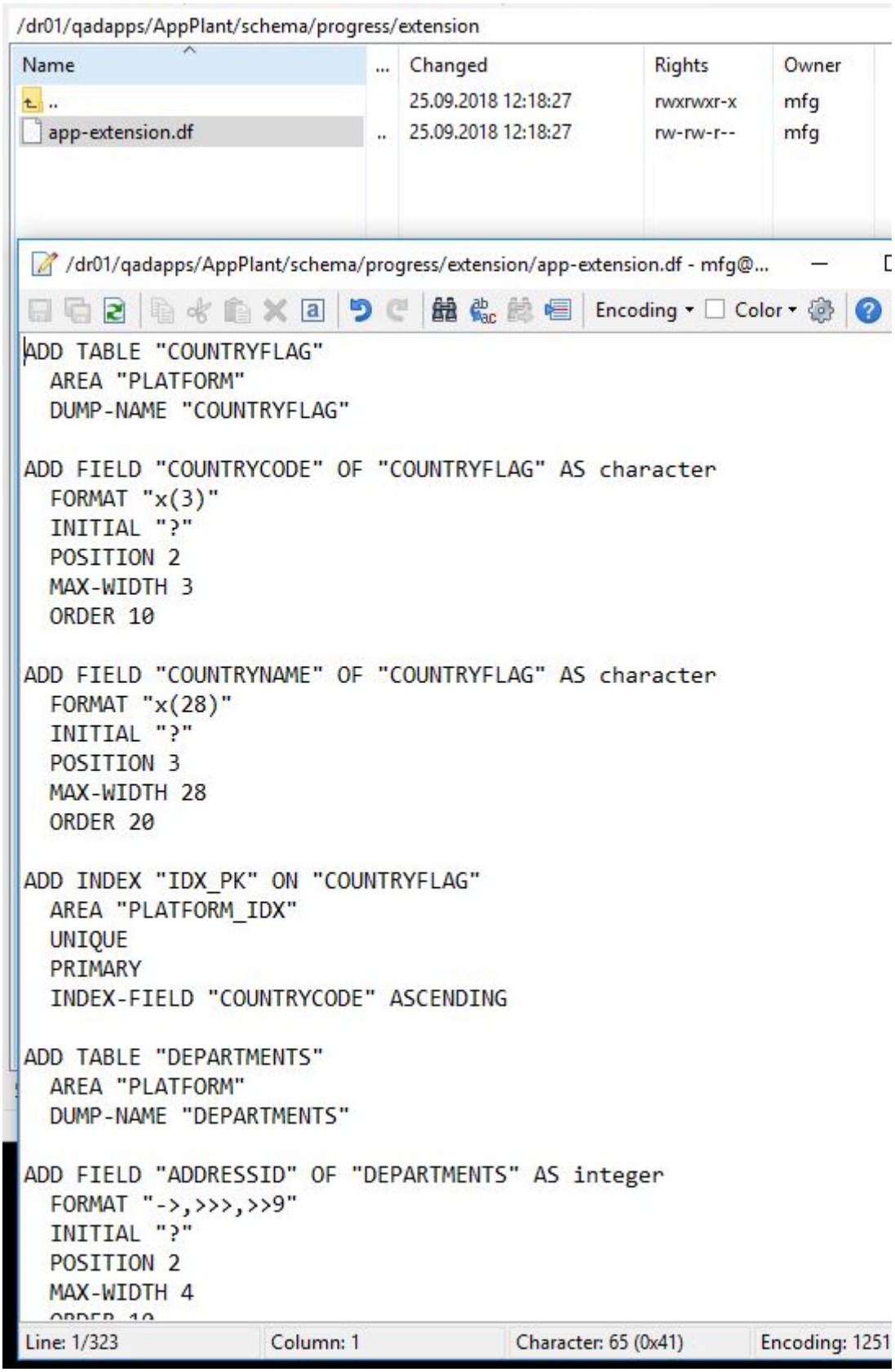


Name	...	Changed	Rights	Owner
..		25.09.2018 12:18:27	rwxrwxr-x	mfg
ace		25.09.2018 12:18:06	rwxrwxrwx	mfg
analytics		25.09.2018 12:18:07	rwxrwxrwx	mfg
app		25.09.2018 12:18:06	rwxrwxrwx	mfg
bebrowse		25.09.2018 12:18:06	rwxrwxrwx	mfg
berelation		25.09.2018 12:18:06	rwxrwxrwx	mfg
entity		25.09.2018 12:18:06	rwxrwxrwx	mfg
entityfieldoverride		25.09.2018 12:18:06	rwxrwxrwx	mfg
eventhandler		25.09.2018 12:18:06	rwxrwxrwx	mfg
fieldsecurity		25.09.2018 12:18:06	rwxrwxrwx	mfg
layout		25.09.2018 12:18:06	rwxrwxrwx	mfg
lookup		25.09.2018 12:18:06	rwxrwxrwx	mfg
menu		25.09.2018 12:18:06	rwxrwxrwx	mfg
meta		25.09.2018 12:18:06	rwxrwxrwx	mfg
role		25.09.2018 12:18:06	rwxrwxrwx	mfg
storedview		25.09.2018 12:18:06	rwxrwxrwx	mfg
strings		25.09.2018 12:18:06	rwxrwxrwx	mfg
view		25.09.2018 12:18:06	rwxrwxrwx	mfg

For example, the exported App has its own xml:



Also, there is a dumped schema:



Load App

To promote your App to another environment (for example a test environment), you need to create a package and install it.

For more information on packaging the app and then installing it in another environment, see Platform Development in YAB.

Configure your platform App in YAB

Edit *configuration.properties* in your environment and add the following line:

platform-extension.*<appname>***.dir=***<directory where you exported the app>*

In our example, that is:

platform-extension.plant.dir=/dr01/qadapps/AppPlant

Create package

1. Connect by SSH to the current environment.
2. Navigate to the working folder of current environment.
3. Run Yab command: **yab dev-***<directory where you exported the app>***-package-create -version:1.0.0.0**

   In our example, that is: yab dev-AppPlant-package-create -version:1.0.0.0

```
[mfg@public-platform-branch sm2]$ yab dev-AppPlant-package-create -vers

                  system-package-create (1 task)
------------------------------------------------------------------------
1/1 system-package-create                                        OK (0.368 s
------------------------------------------------------------------------

BUILD SUCCESSFUL (1.161 s)
[mfg@public-platform-branch sm2]$
```

OR if you want to specify the exact name "testnew" for the package:

yab -class:*<name of package>* **dev-***<directory where you exported the app>***-package-create -version:2.0.0.0**

In our example, that is:

yab -class:testnew dev-AppPlant-package-create -version:2.0.0.0

New package is being created in the working folder:

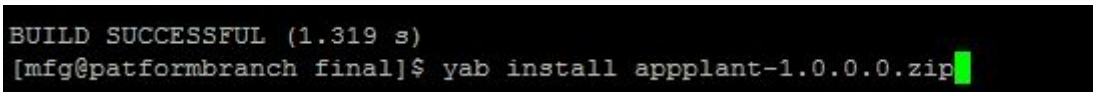
/dr01/qadapps/sm2

Name	...	Changed	Rights	Owner
..		25.09.2018 12:18:05	rwxrwxrwx	root
build		05.09.2018 15:24:09	rwxrwxr-x	mfg
config		05.09.2018 17:49:57	rwxrwxr-x	mfg
customizations		05.09.2018 15:27:07	rwxrwxr-x	mfg
databases		08.09.2018 13:28:36	rwxrwxrwx	mfg
dist		05.09.2018 17:49:57	rwxrwxr-x	mfg
extensions		05.09.2018 15:26:43	rwxrwxr-x	mfg
patches		05.09.2018 15:27:07	rwxrwxr-x	mfg
scripts		08.09.2018 11:55:16	rwxrwxr-x	mfg
servers		08.09.2018 11:55:17	rwxrwxr-x	mfg
storage		12.09.2018 13:10:09	rwxrwxr-x	mfg
appplant-1.0.0.0.zip	..	25.09.2018 13:29:39	rw-rw-r--	mfg

Installing into a new environment

1. Connect by SSH to another target environment, where the package should be installed.

2. Navigate to the working folder of current environment.
3. Copy the new package to the working directory of current environment.
4. Run Yab command: **yab install** *<name of package with version and extension>*



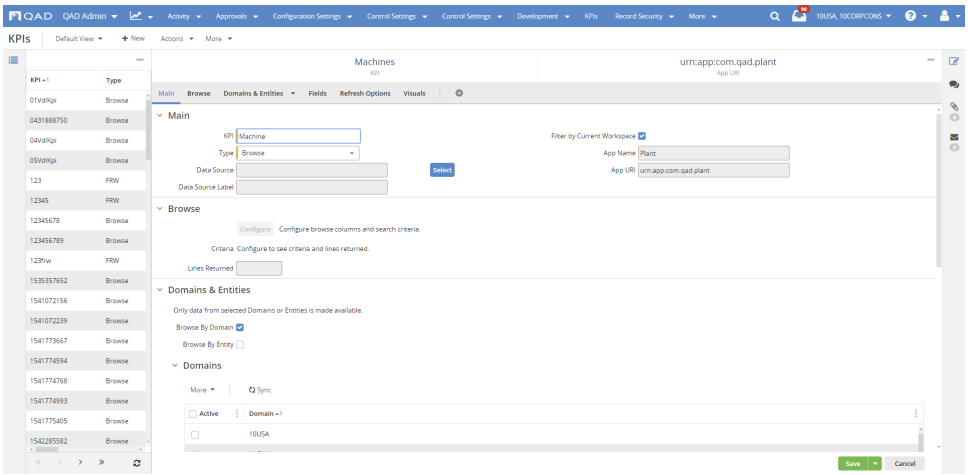
Installing the package can take the same time as updating the environment.

After successful installation of the package, the target environment will contain all the resources and information.

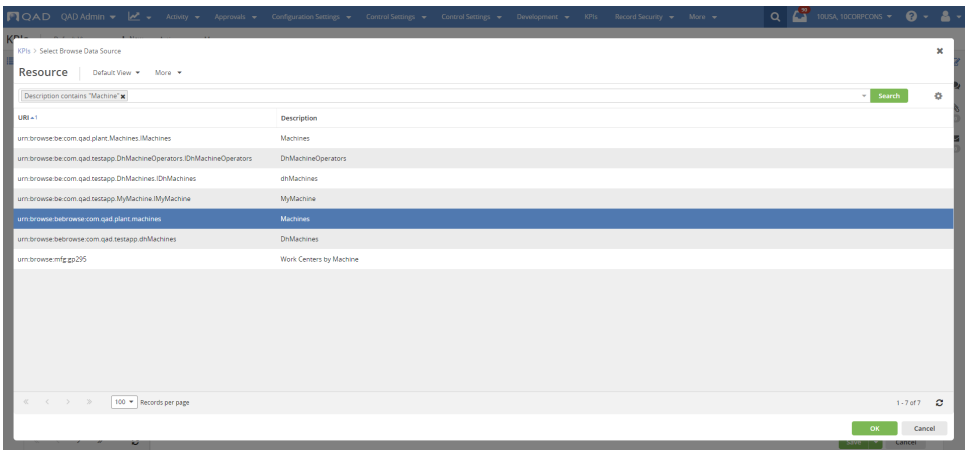
Note: For detailed information, go to the QDN page Platform Development in YAB.

12. Create KPIs

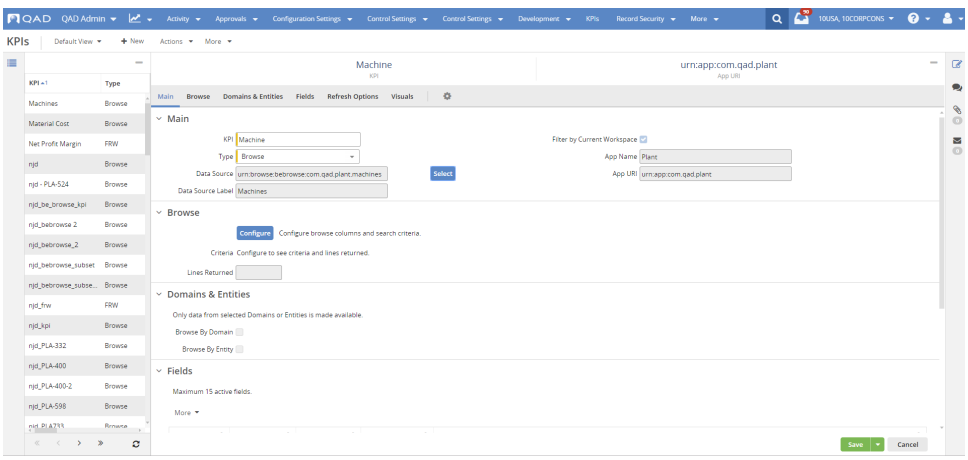
1. Navigate to **KPIs** from the menu search.
2. Click the **New** toolbar button.
3. Complete the **KPI** field.



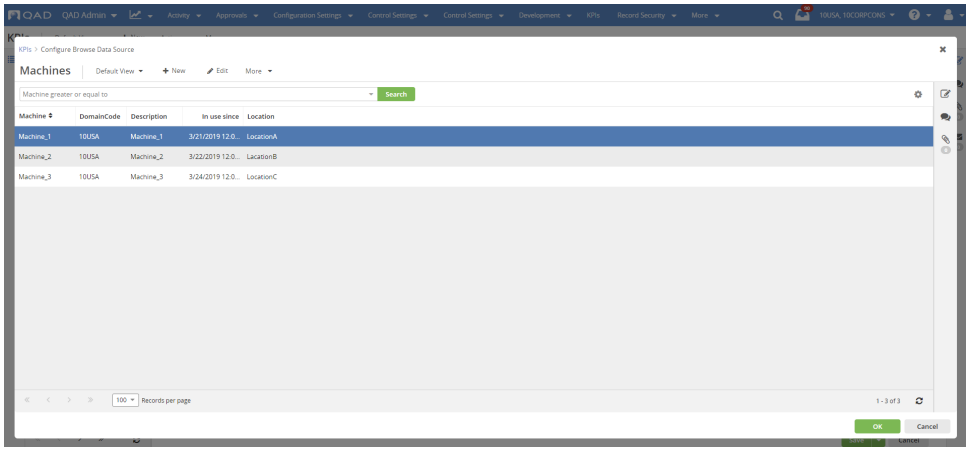
4. Click the **Select** button and choose a business component's browse.



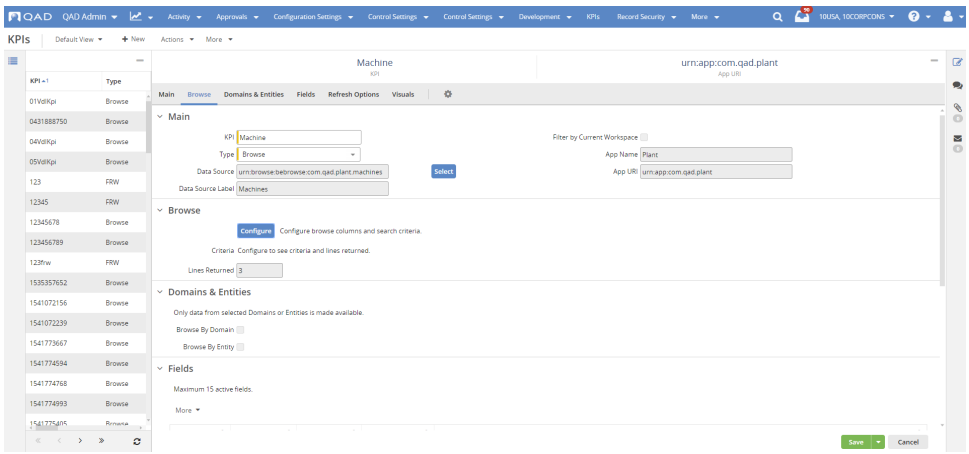
5. Click **OK**.



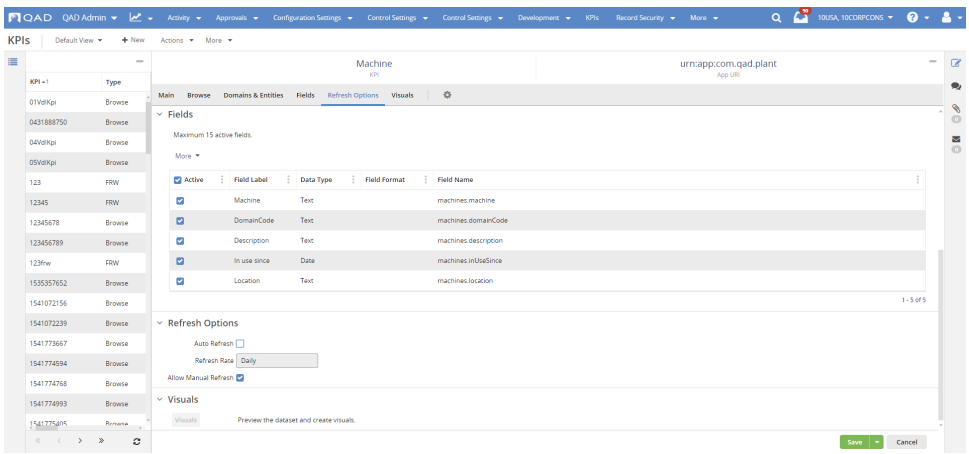
6. Click the **Configure** button on the **Browse** panel to configure browse columns and search criteria.



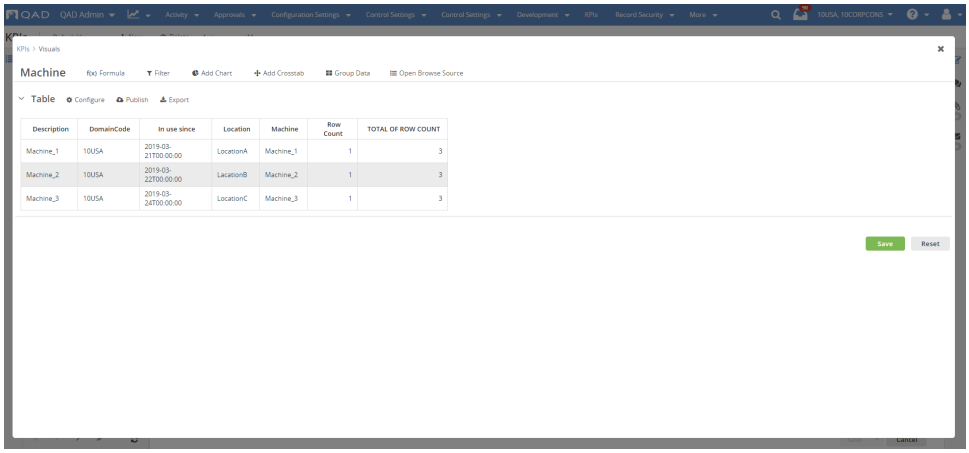
7. Click **OK** and go to the **Fields** panel. The fields from Browse are included.



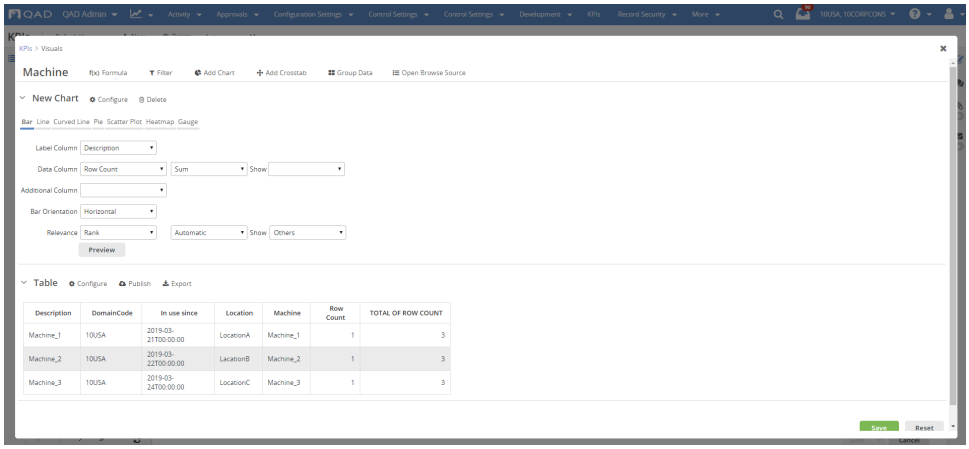
8. On the **Refresh Options** panel, clear the "Auto Refresh" option to prevent exceeding the limit of the auto-refreshed KPIs.



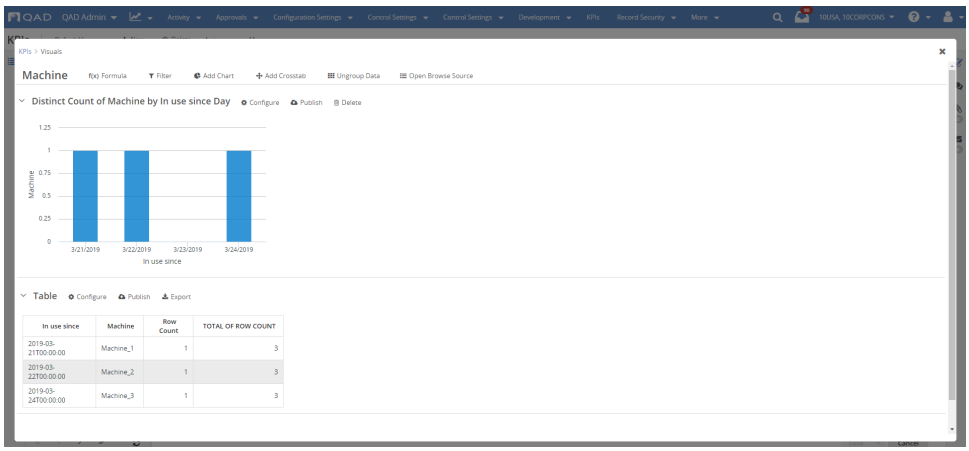
9. Save a new KPI.
10. On the **Visuals** panel, click the **Visuals** button to create visuals.



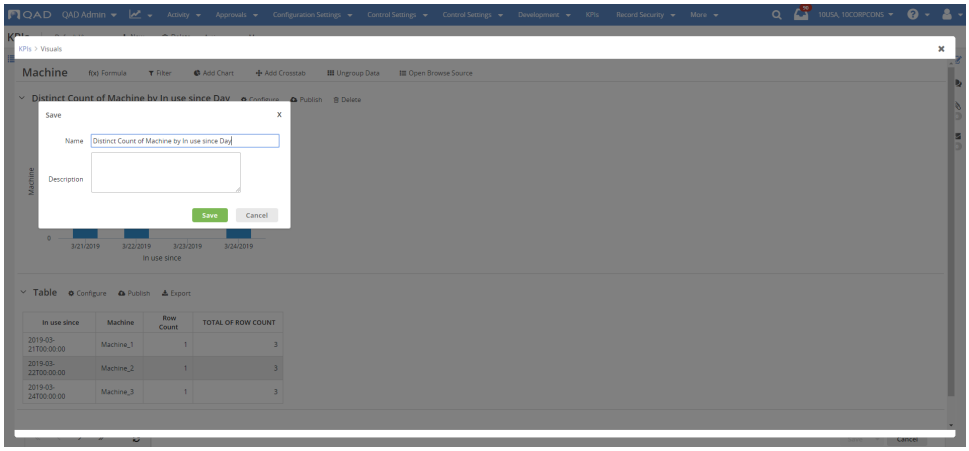
11. Click the **Add Chart** button.



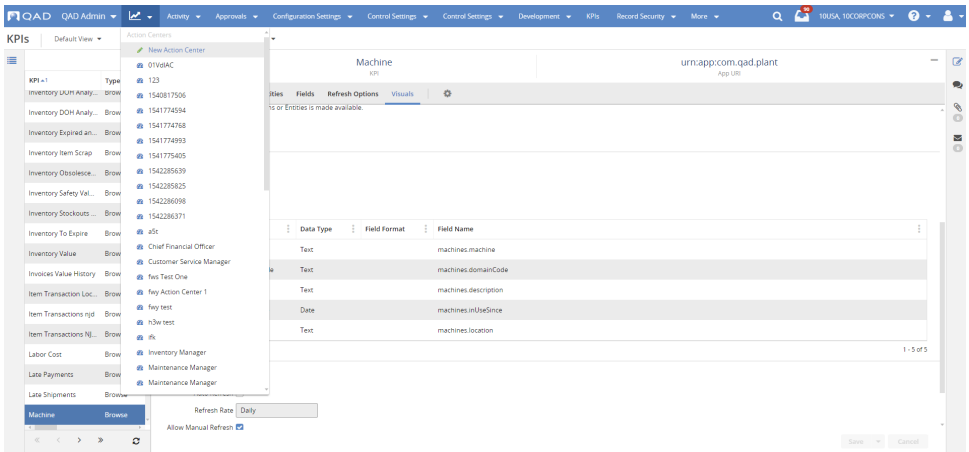
12. From the **Label Column** drop-down menu, select "In use since". From the **Data Column** drop-down menu, select "Machines", and then click the **Preview** button.



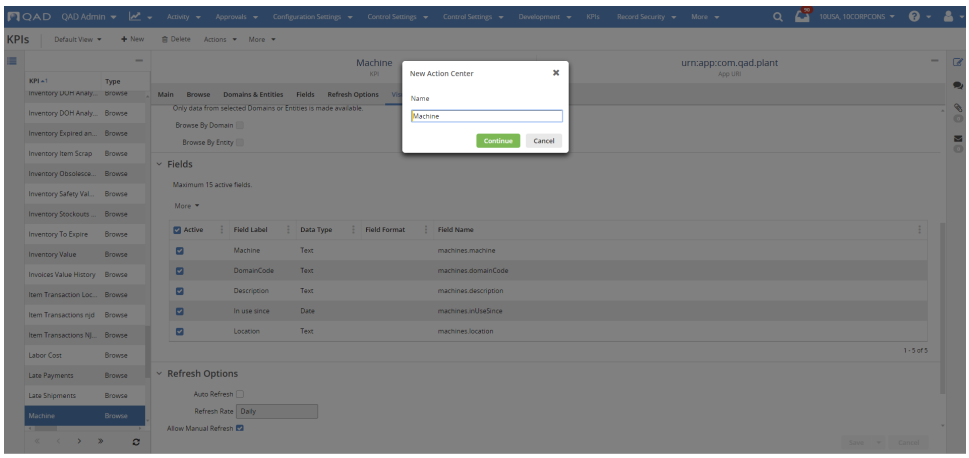
13. On the **Group Count of Machine by In use since Year** panel, click the **Publish** button. Complete the "Name" and "Description" fields if needed.



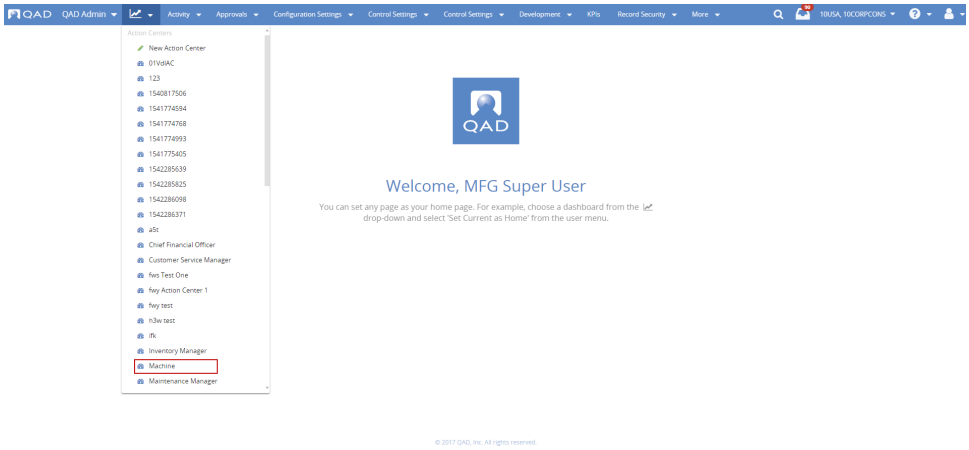
14. Click the **Save** button to publish visual to Gallery.
15. Click **Menu Dashboard** and choose **New Action Center**.



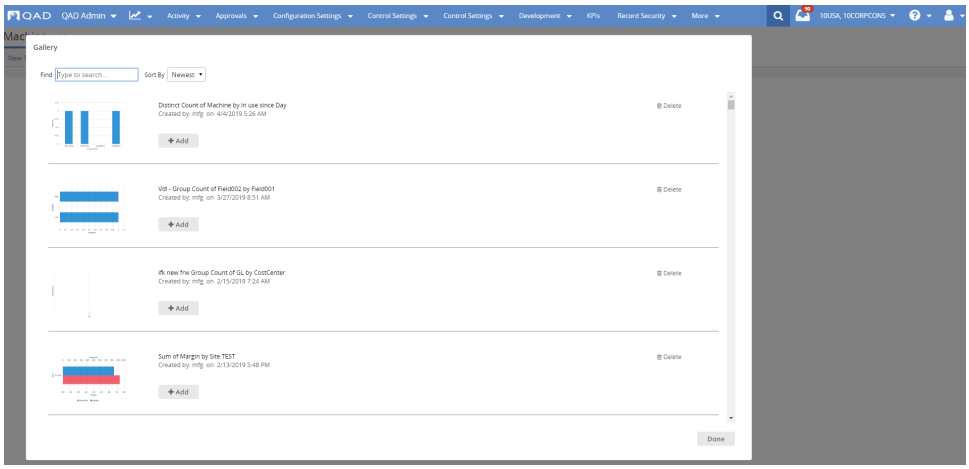
16. Define the name for a new Action Center and click **Continue**.



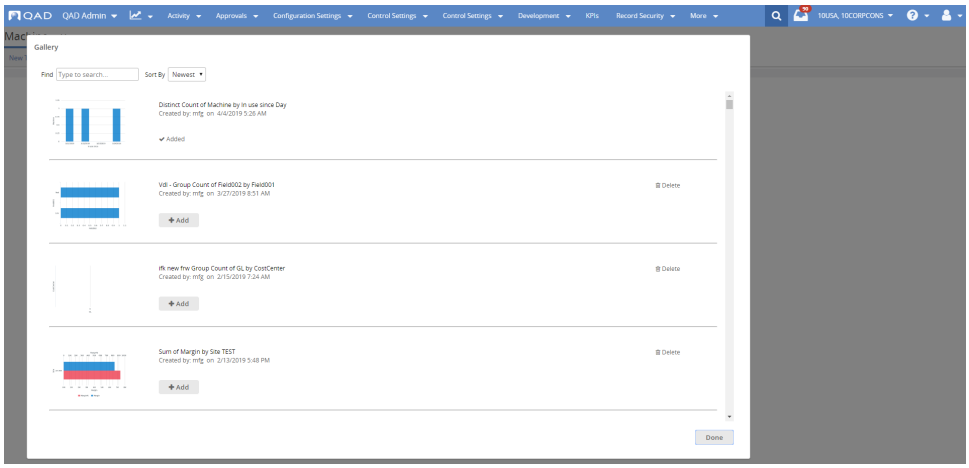
17. Click **Menu Dashboard** and choose the created Action Center.



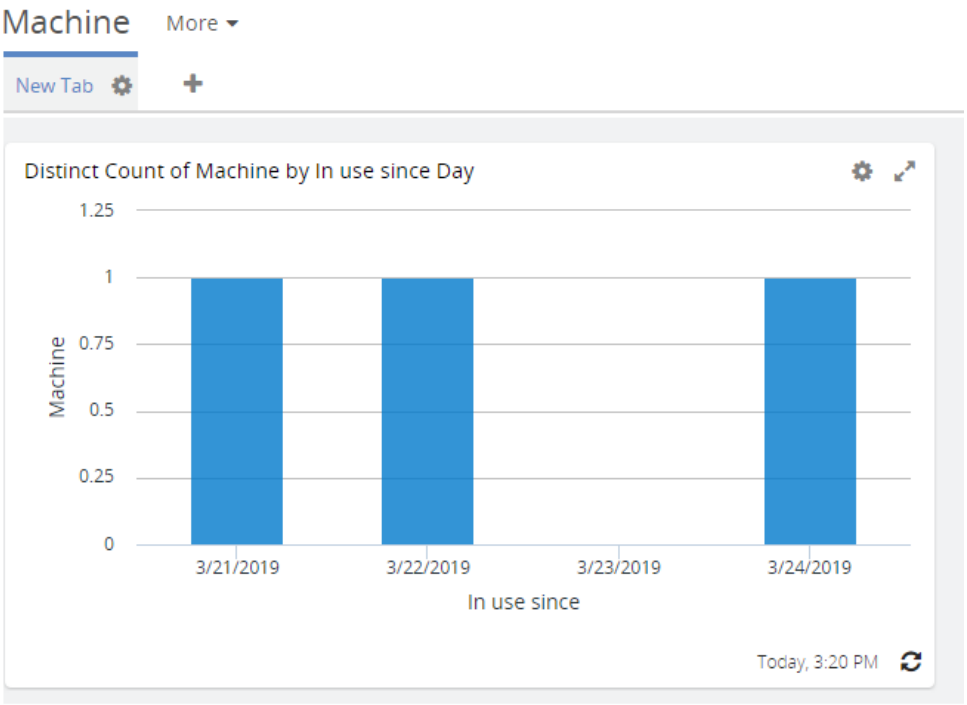
18. A new Action Center opens.



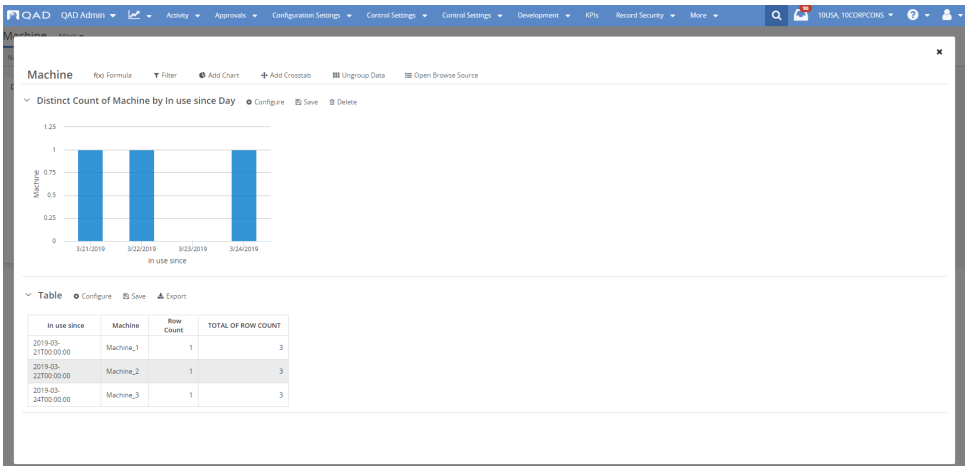
19. From the opened Gallery, add last created visual by clicking the **+Add** button.



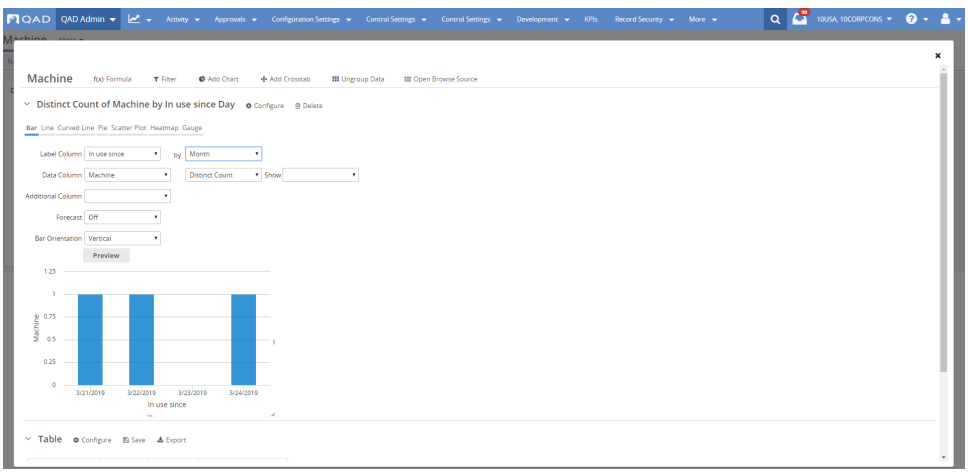
20. Click **Done**. The Action Center is opened with the added visual.



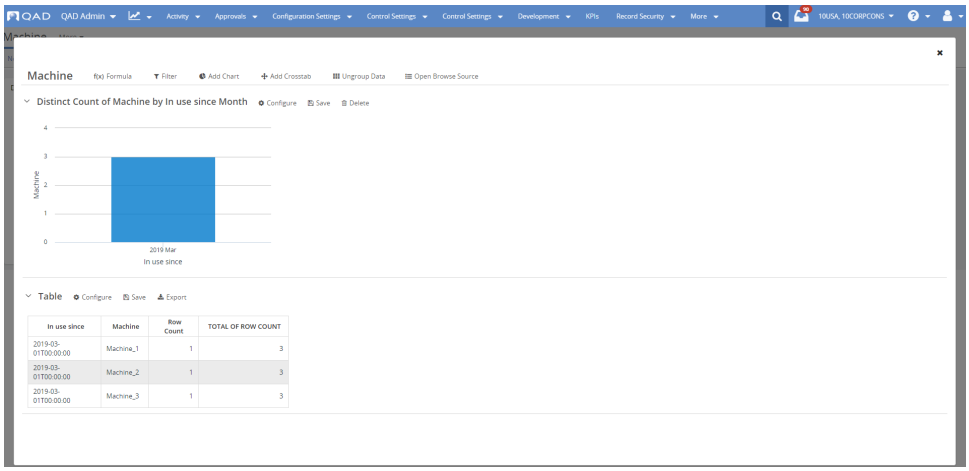
21. Expand chart by clicking the arrows in the top-right corner.



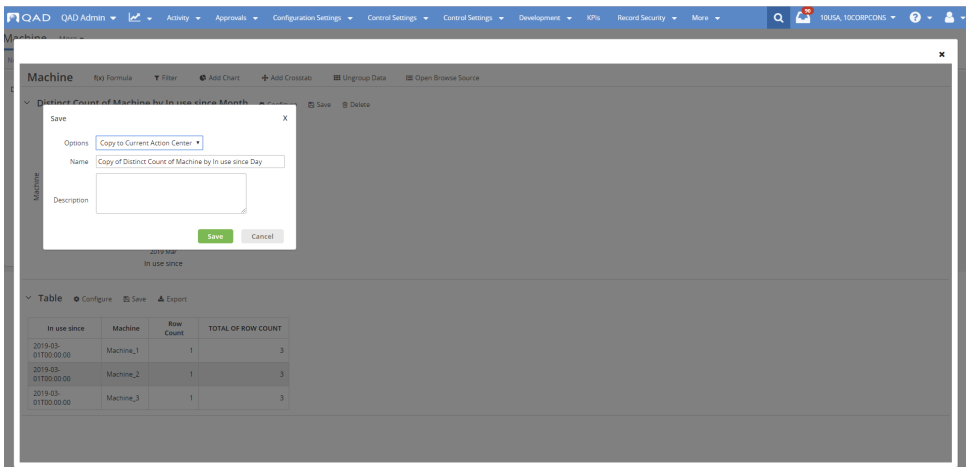
22. Click the **Configure** button and set condition "by" as "Month".



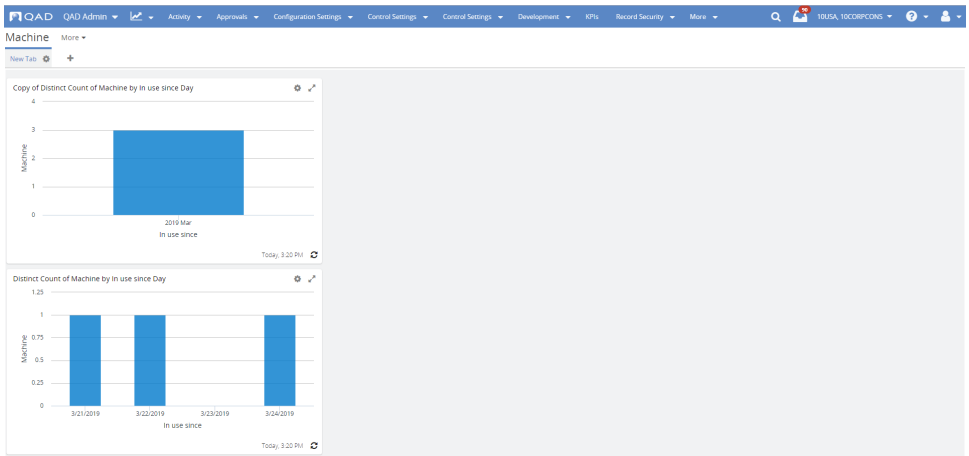
23. Click the **Preview** button.



24. Save visual with redefining "Options" and "Name".



25. Click the **Save** button.



Now there are two Charts on the **Action Center** tab.

Examples - Step by Step

Introduction

This section explains a few examples that can be followed step by step to learn how to create an extension app.

Examples

Example 1 - Extend the UI: Adding new functionality to item maintenance

In this example we will show how to extend Items UI with some extra UI functionality that will do the following:

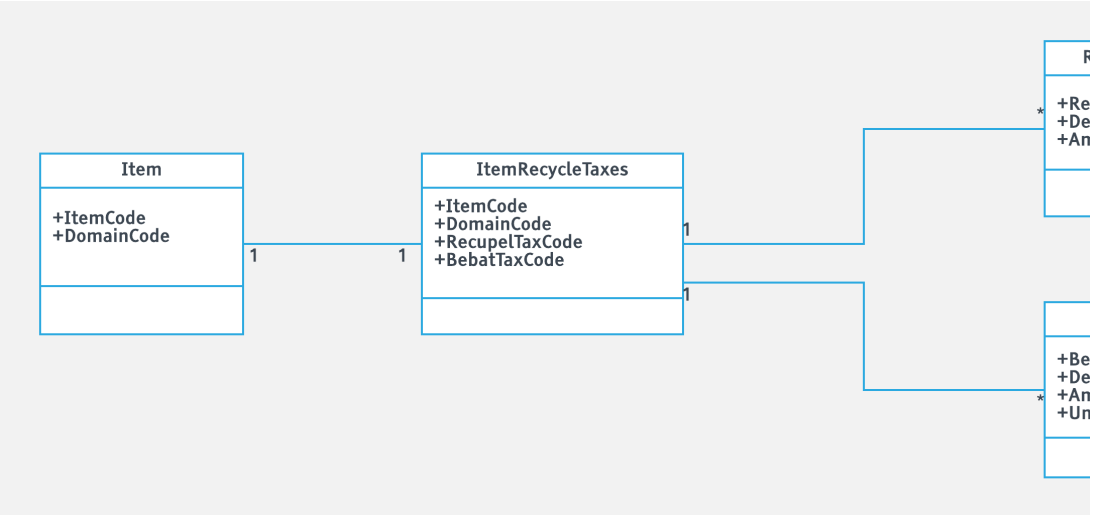
- Extend the UI with extra fields about recycle tax information.
- Extend the UI with extra fields and a grid about electrical compatibility information.

Steps to follow:

Example 1.1: Adding new functionality to item maintenance with a 1-1 extension

In Belgium, there is a recycle tax on electronic devices called recupel tax. And there is also a tax on devices with a battery called bebat. Both taxes are a fixed value depending on the type of the device and battery. In this example, we will set up the necessary business components for these taxes, and extend item with these 2 taxes.

In a schematic view, this is what we will create:



Important here is that the 1-1 relation between Item and ItemRecycleTaxes is an extension relation (because we want to add extra fields on the items UI). The 1-N relations to RecupelTaxCategory and BebatTaxcategory are lookup relations. They make sure that we will see lookups on the fields that we are adding to items. See Business Component Relation for more info on the different types of relations.

The app we will create will allow the following what concerns recycle taxes:

1. Maintain recupel categories.
2. Maintain bebat categories.
3. Extend item with the ability to:
    a. Select a recupel and bebat category.
    b. Display the total price including the 2 taxes.
4. When saving an item, validate that the 2 taxes are in the data set for certain items.
5. Display the taxes and the total price in a browse.

- Create RecupelTaxCategory Components
- Create BebatTaxCategory Business Component
- Create ItemRecycleTaxes Components
- Create Items with Recycle Taxes BC browse
- Make tax value fields read only on view
- Add event handler to retrieve Tax Value data
- Add BL validation to check if one of the tax categories is used when deleting
- Ref: complete event handler code for the Items BC

Create RecupelTaxCategory Components

This section explains all the steps to create Business Components for Recupel Tax:

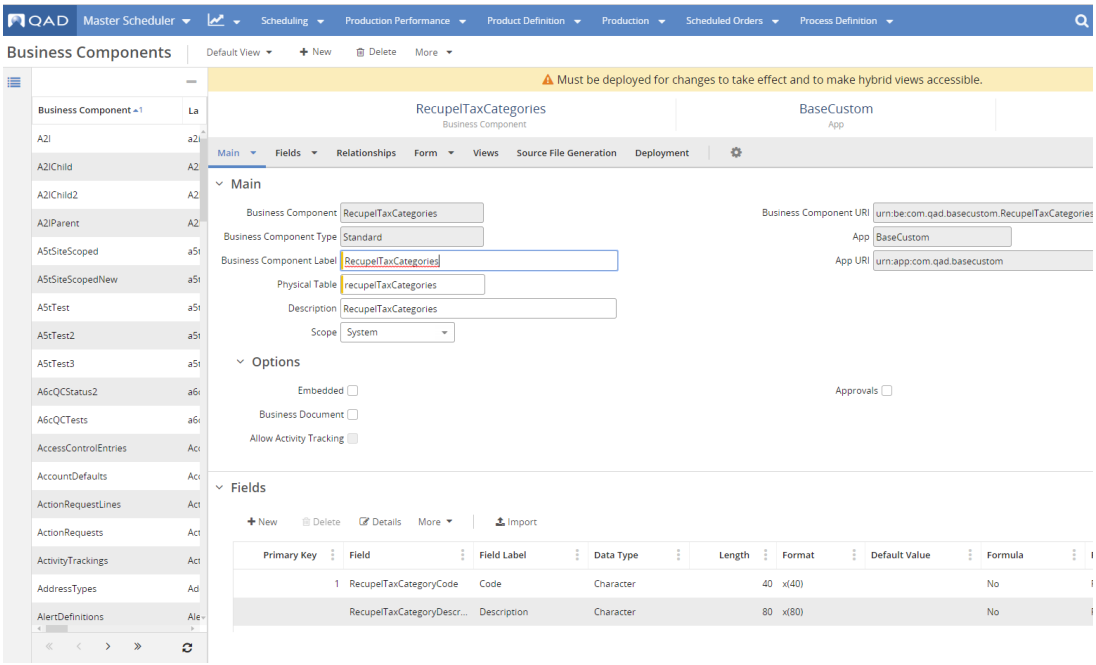
Create RecupelTaxCategories Business Component

The RecupelTaxCategories business component represents the different Belgian Recupel tax categories. Every category is for a certain type of electronic device and has a description of these type of devices and a fixed tax value.

Do the following to create the business component:

1. Navigate to **Business Components** from the menu search and click the **New** toolbar button.
2. Enter the following values on the screen:
    - Business Component: The name of the BC: RecupelTaxCategories
    - Business Component Label: This is the label that is used for the BC to display it in other screens (e.g., in browses/lookups): Recupel Tax Categories
    - Physical Table: The name of the table that will be created in the data base: RecupelTaxCategories
    - Description: Description of the function of the BC: Recupel Tax Categories business component
    - Scope: The scope the BC will run in is System in this case because these taxes are everywhere the same in Belgium
    - Fields:
        - RecupelTaxCategoryCode: A code identifying the tax category (Label Code, Type Character, length 40, display format x(40), primary key 1)
        - RecupelTaxCategoryDescription: Description of the electronic devices this code applies to (Label Description, Type Character, length 80, display format x(80))
        - RecupelTaxCategoryTaxValue: The value of the tax applied (Label Tax Value, Type Decimal, default display format)
3. Click **Save**.

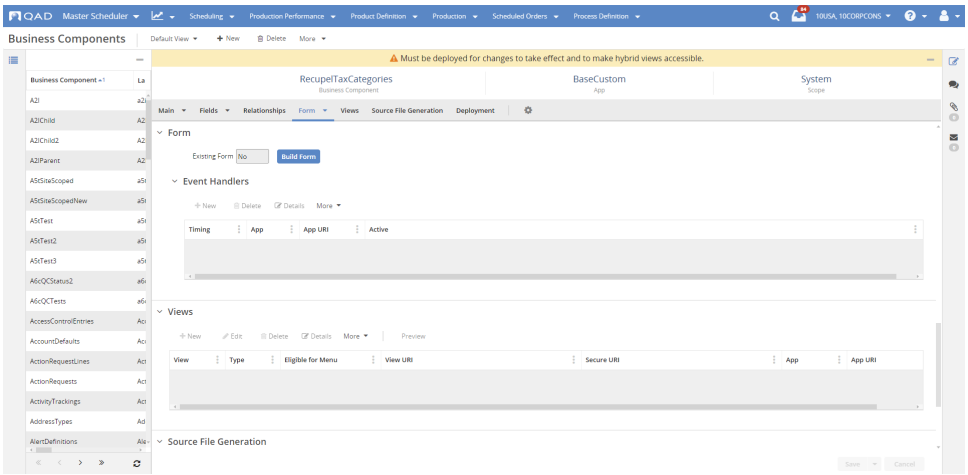
This is what the BC should look like:



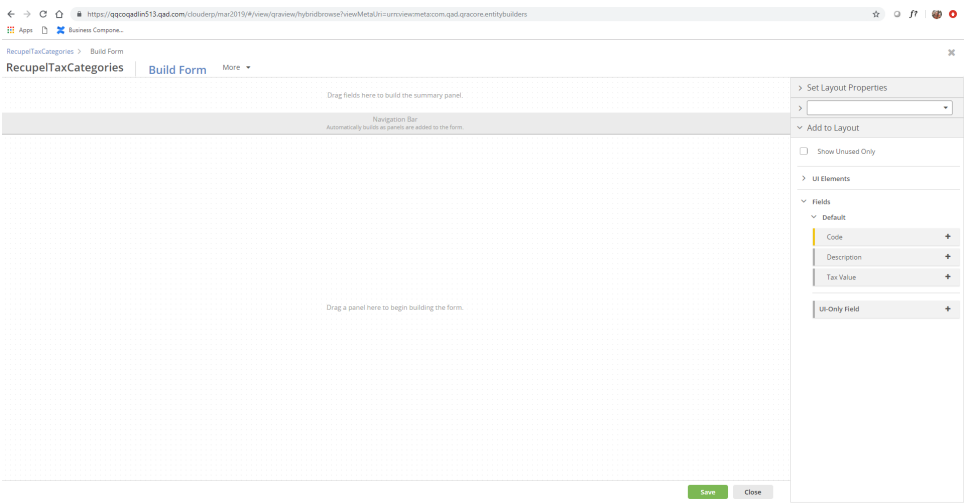
Create RecupelTaxCategories View

The following step is to create a view form for the RecupelTaxCategories BC we just created. In order to do so, do the following steps:

1. Open the **RecupelTaxCategories** BC.
2. Click the **Form** panel.

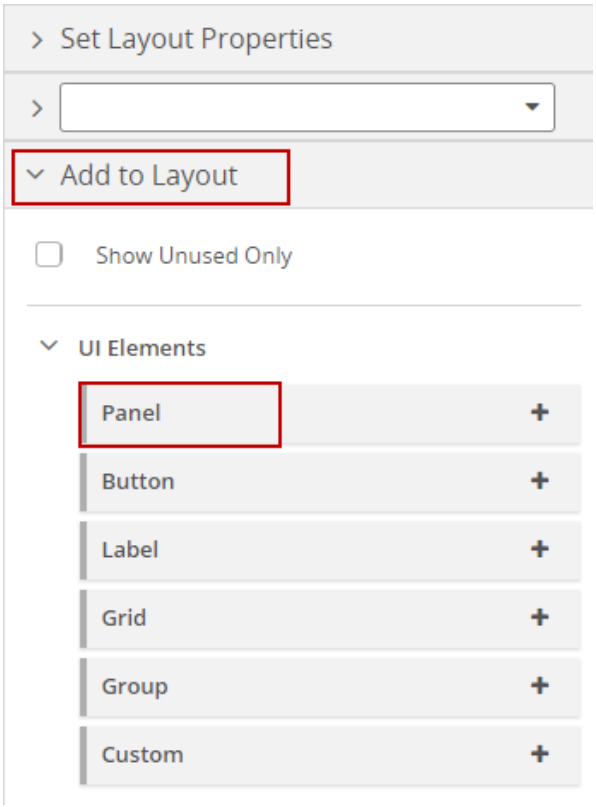


3. Click the **Build Form** button. This will open the form builder.

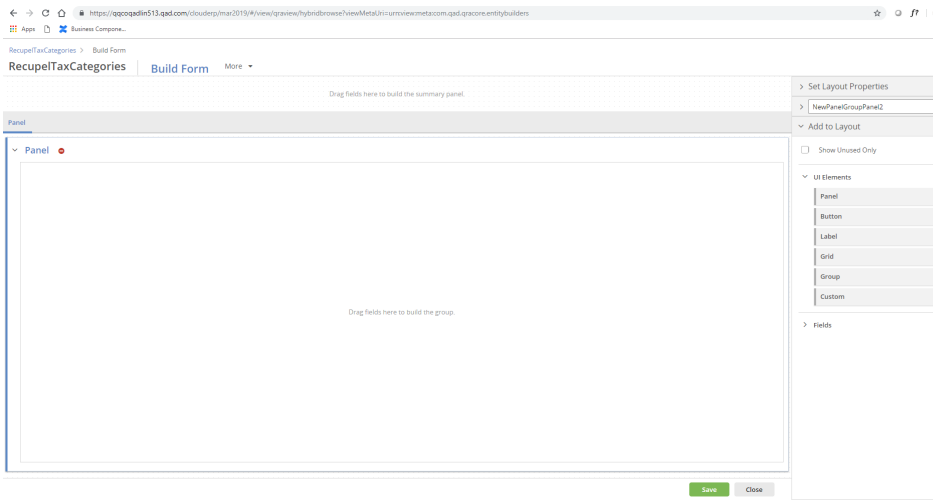


4. We now have an empty view that we can start dragging and dropping UI elements to. Follow these steps to add the elements:

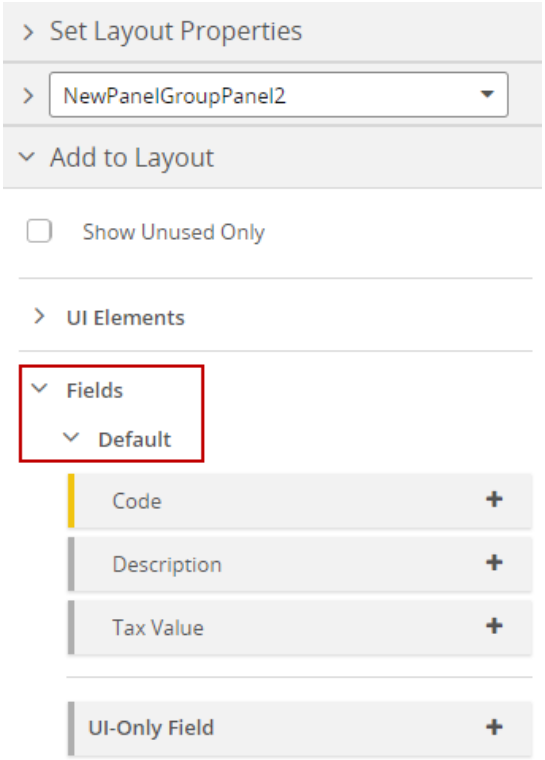
    1. Click the **Add to Layout** menu at the right.



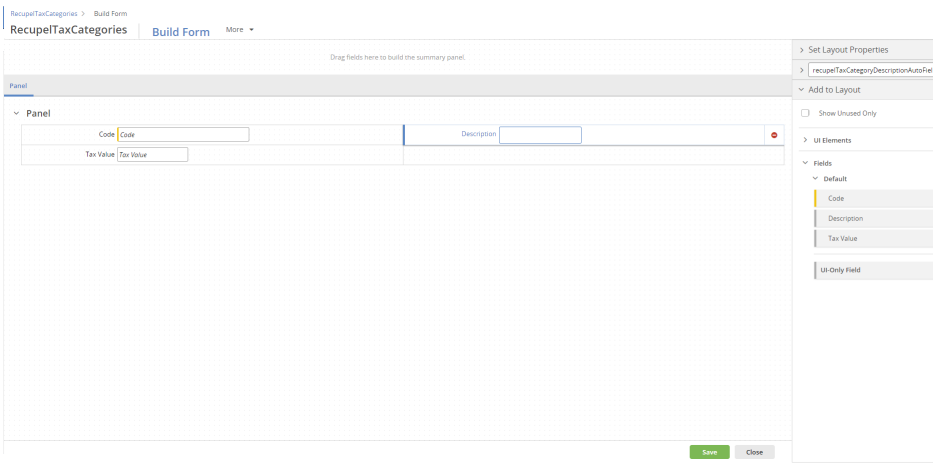
2. Now select the **Panel** box and drag and drop it on the empty view.



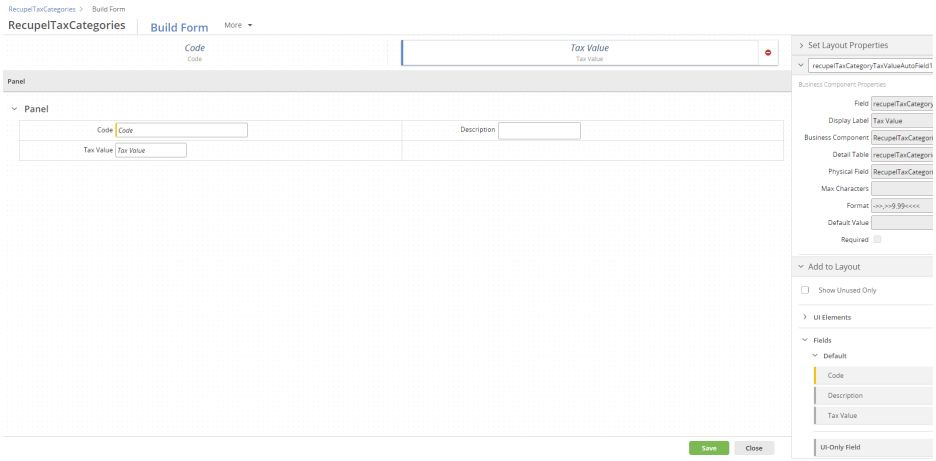
3. Drag and drop fields on the panel. Click the **Fields** and **Default** menu items at the right bottom.



4. Drag and drop the **Code** field on the panel. Do the same for the **Description** and **Tax Value** fields so that the screen looks like this.



5. The next step is to create the **Summary** panel. This can be done by dropping **Code** and **Tax Value** on the **Summary** panel at the top of the screen.

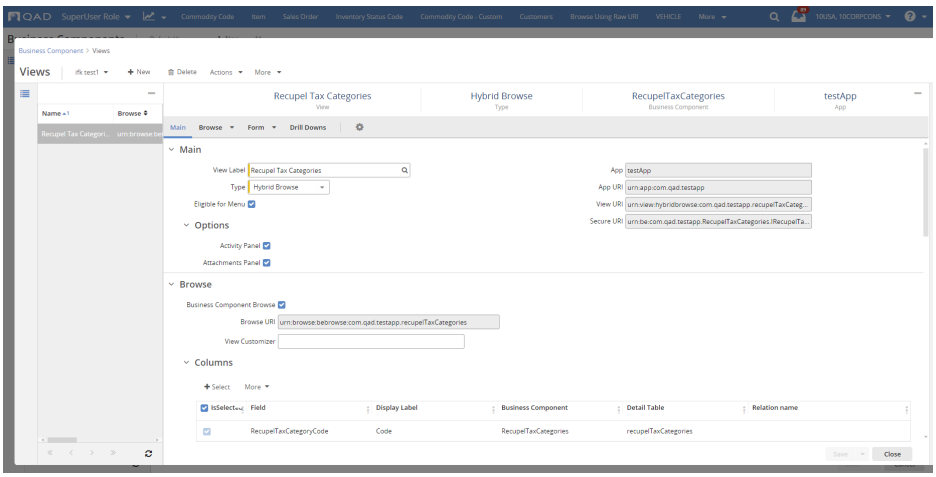


6. Now our form is ready. Click the **Save** button to save the form, and then the **Close** button.
7. After saving the form, we are back on the BC screen ready to create a Hybrid view.

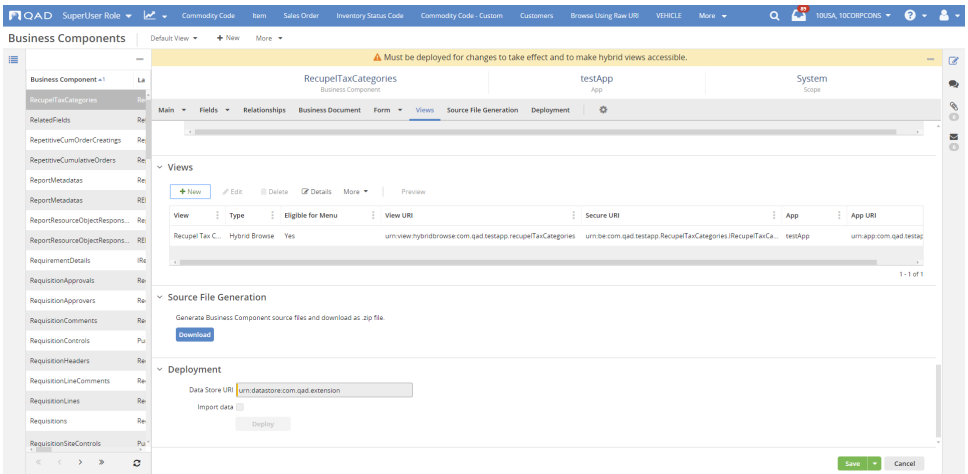
Create RecupelTaxCategories Browse

In order to see the RecupelTaxCategories BC in the menu and open a browse on it, we need to create a BC browse. This can be done by following these steps:

1. On the RecupelTaxCategories BC screen, click the **Views** panel, and then click the **New** button. This will bring up a details screen where you can create the browse. On that screen, enter the following:
    - View Label: Recupel Tax Categories, this is the label the hybrid view will have on the menu.
    - In the **Browse** columns grid, deselect the RecupelTaxCategoryDescription column. This column is too big to show in a browse, this way we remove it from the browse view.
    - Now the screen should look as follows.



2. Click the **Save** button to save this browse, and then **Close**.
3. After saving the new Hybrid view, the UI looks as follows.

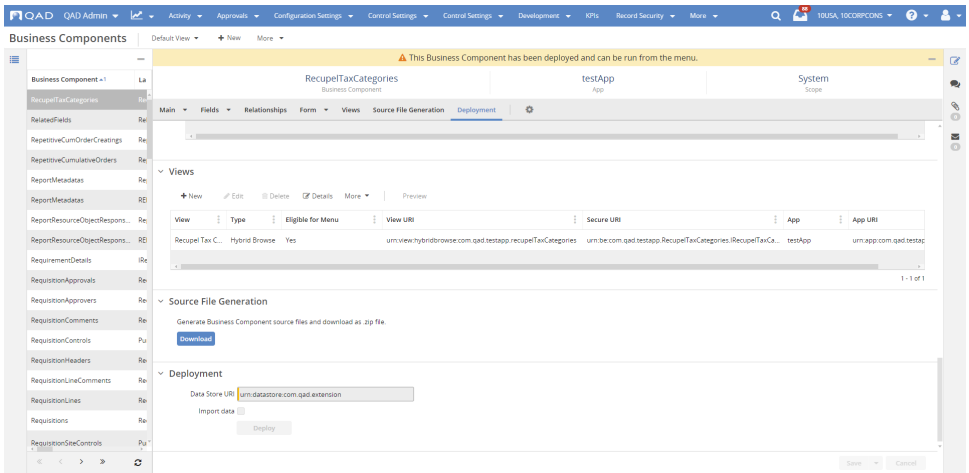


4. Now click the **Save** button again to save changes.

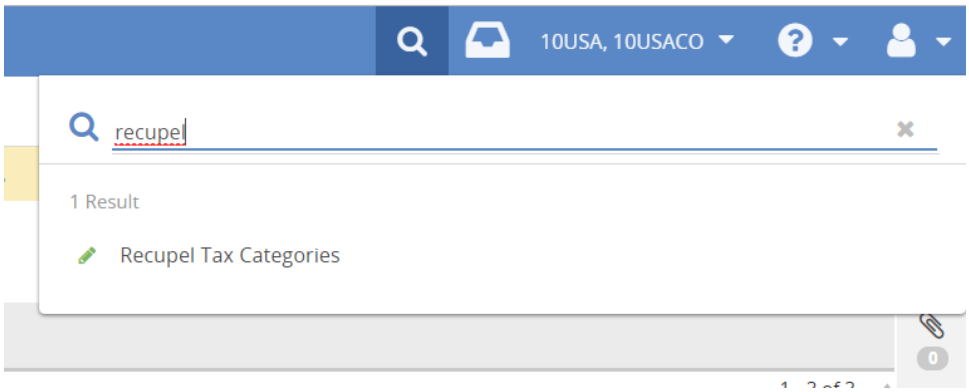
Deploy RecupelTaxCategories Business Component

The last step for the RecupelTaxCategories BC is to deploy it so that it becomes active in the application. To do this, follow these steps:

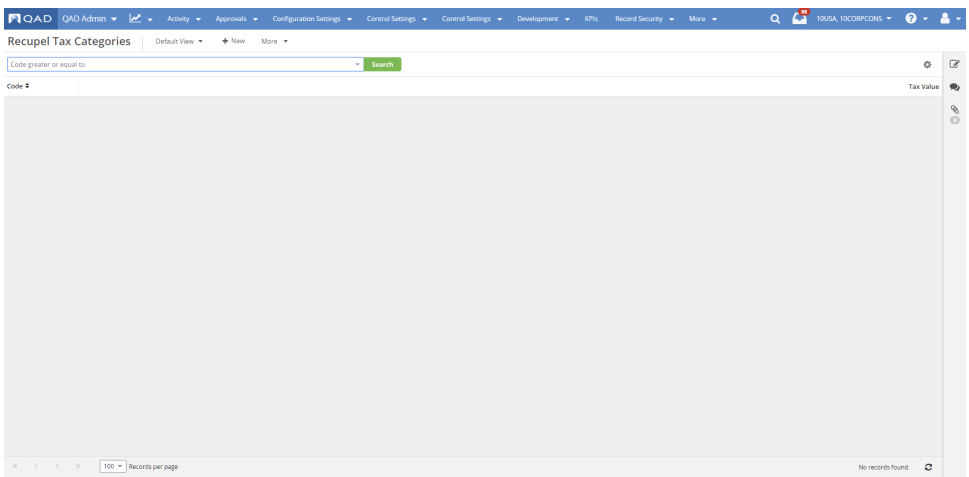
1. Open the Business Components Maintenance screen, search for the RecupelTaxCategories business component, and then click the **Edit** button.
2. Click the **Deployment** panel, and then click the lookup button on the **DataStore URI** field. This will open a lookup with the available data stores. Select your developer data store and click **OK**.
3. Click the **Deploy** button to deploy the BC and make it active.



4. Now that our BC is active, we can search for it in the menu.



5. And open it.



Create BebatTaxCategory Business Component

This section explains all the steps to create the Bebat tax business component:

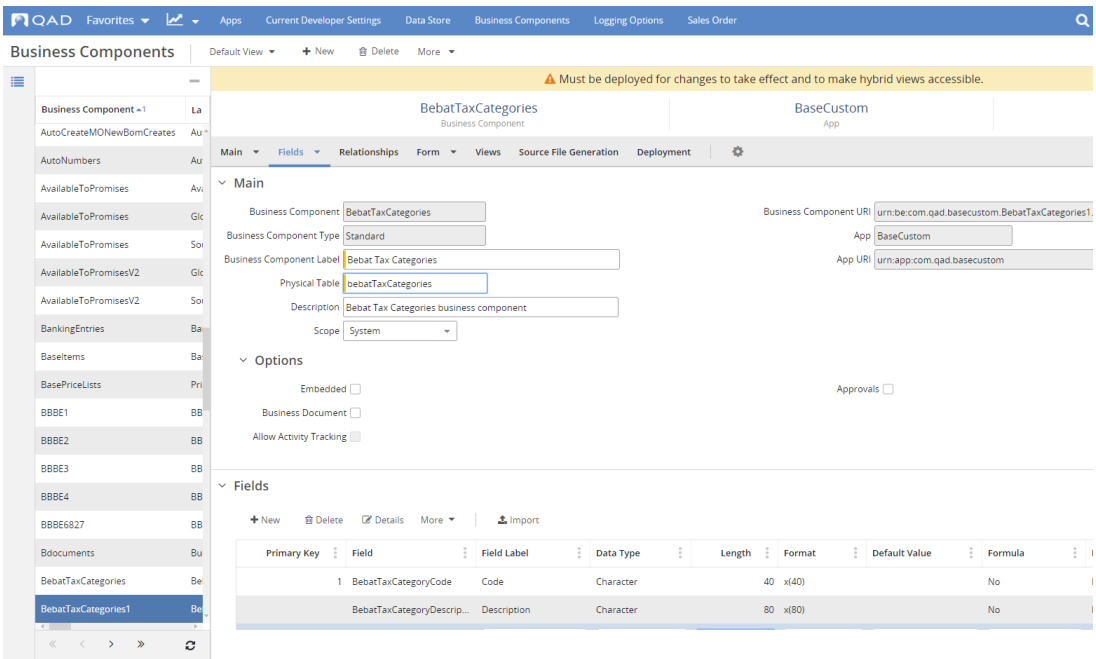
Create BebatTaxCategories Business Component

The BebatTaxCategories business component represents the different Belgian Bebat tax categories. Every category is for a certain type of electronic device and has a description of these type of devices and a fixed tax value.

To create the business component:

1. Open Business Components maintenance UI and click the **New** button.
2. Enter the following values in the fields:
    - Business Component: The name of the BC: BebatTaxCategories
    - Business Component Label: This is the label that is used for the BC to display it in other screens (e.g., in browses/lookups): Bebat Tax Categories
    - Physical Table: The name of the table that will be created in the data base: bebatTaxCategories
    - Description: Description of the function of the BC: Bebat Tax Categories business component
    - Scope: The scope the BC will run in is System in this case because these taxes are everywhere the same in Belgium
    - Fields:
        - BebatTaxCategoryCode: A code identifying the tax category (Label code, Type Character, length 40, display format x(40), primary key 1)
        - BebatTaxCategoryDescription: Description of the electronic devices this code applies to (Label Description, Type Character, length 80, display format x(80))
        - BebatTaxCategoryTaxValue: The value of the tax applied (Label Tax Value, Type Decimal, default display format)
3. Click **Save**.

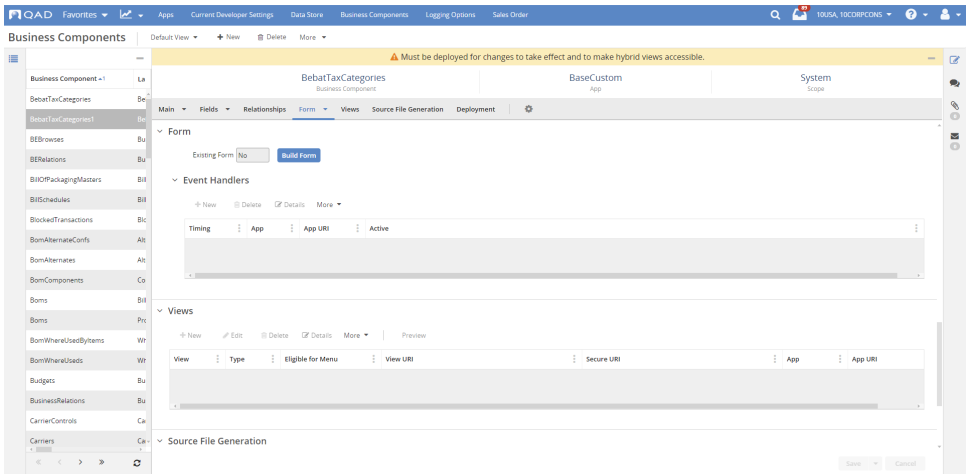
This is what the BC should look like:



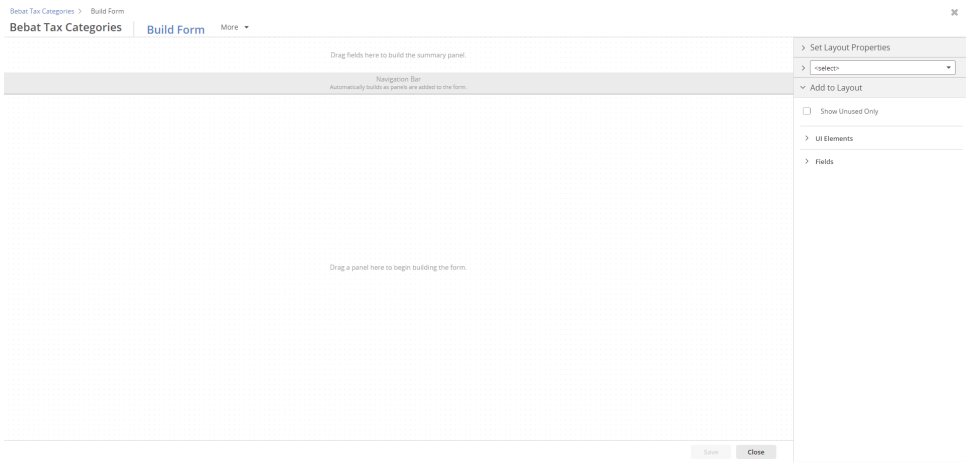
Create BebatTaxCategory View

To create a view form for the BebatTaxCategories BC, do the following steps:

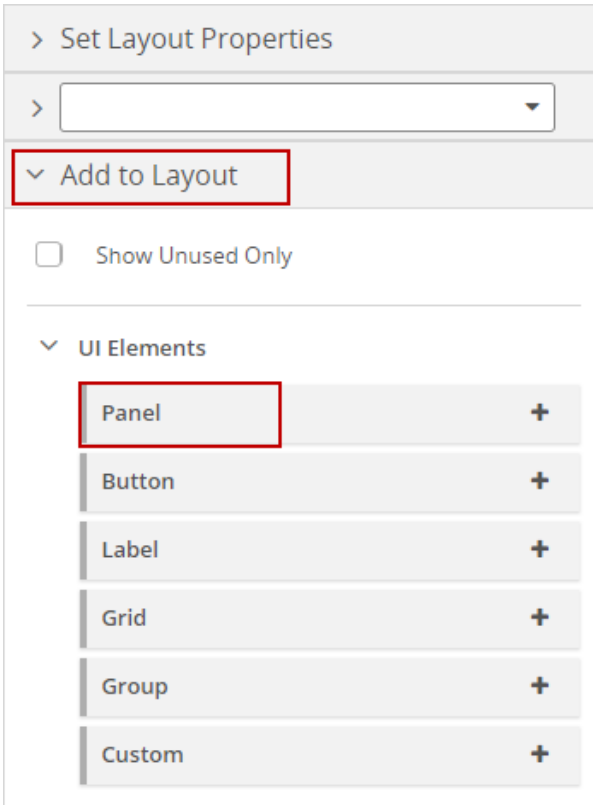
1. Open the **BebatTaxCategories** BC.
2. Click the **Form** panel.



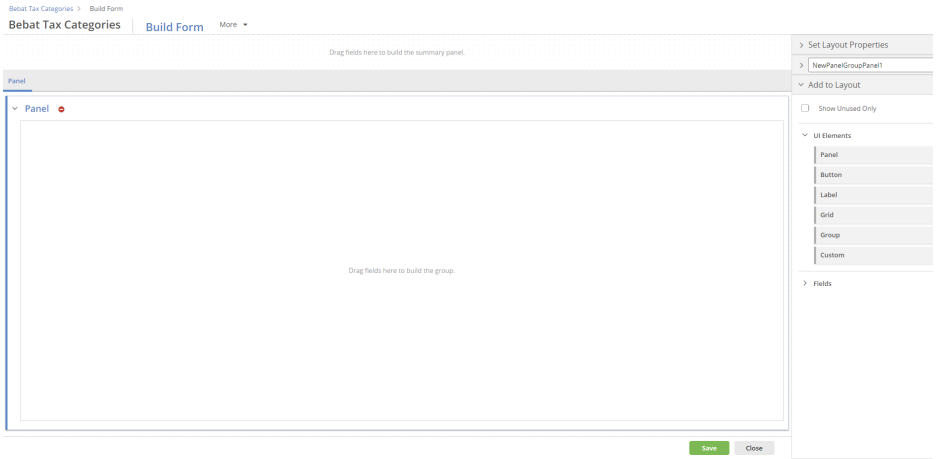
3. Click the **Build Form** button. This will open the form builder.



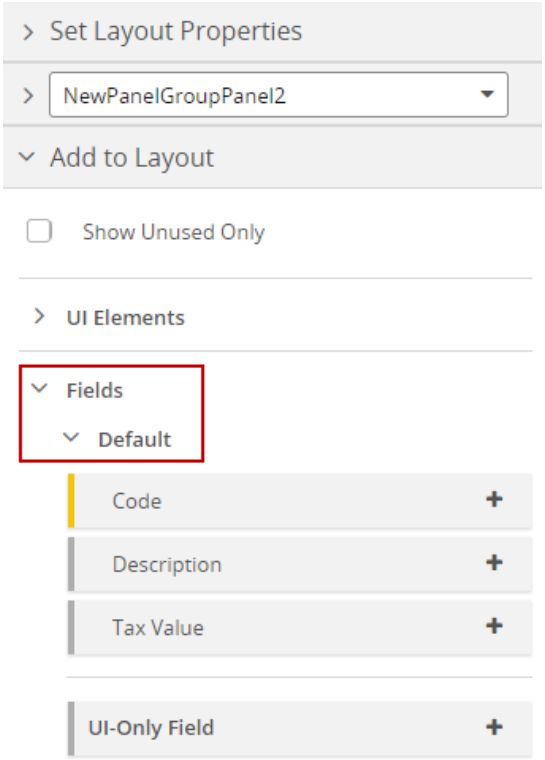
4. We now have an empty view that we can start dragging and dropping UI elements to. Follow these steps to add the elements:
    a. Click the **Add to Layout** menu at the right.



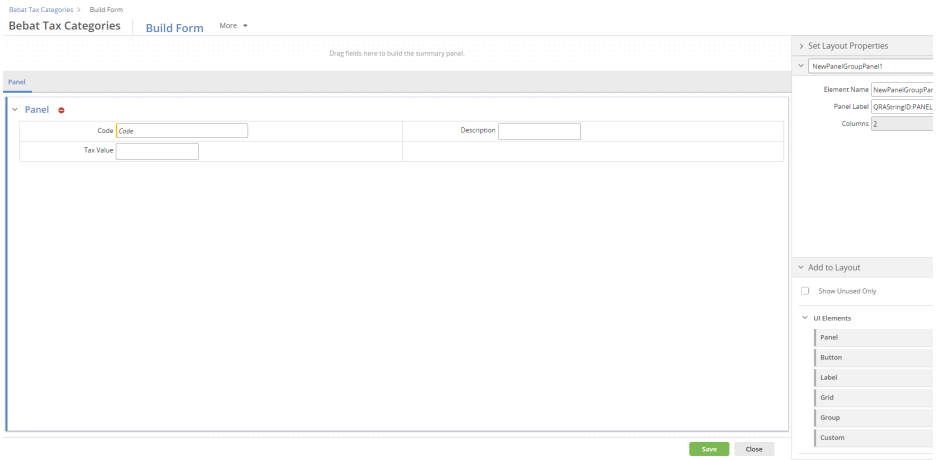
b. Now select the **Panel** box and drag and drop it on the empty view.



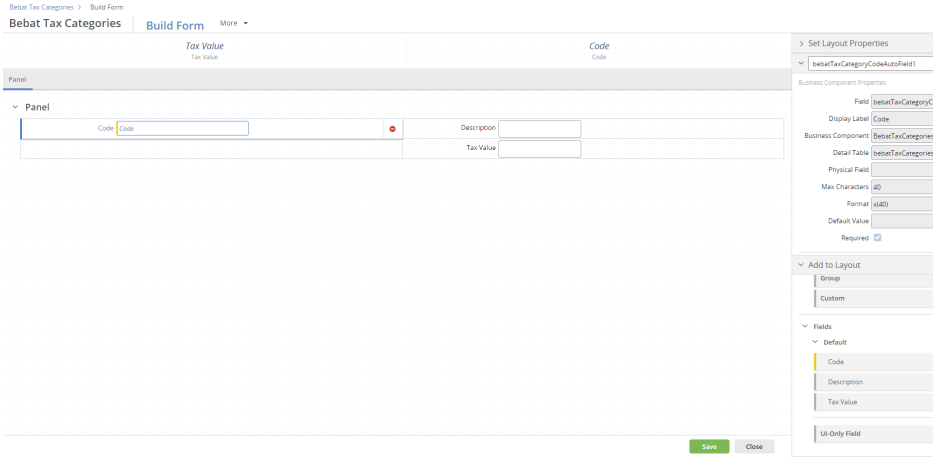
c. Drag and drop fields on the panel. Click the **Fields** and **Default** menu items at the right bottom.



d. Drag and drop the **Code** field on the panel. Do the same for the **Description** and **Tax Value** fields so that the screen looks like this.



e. The next step is to create the **Summary** panel. This can be done by dropping **Code** and **Tax Value** on the **Summary** panel at the top of the screen.

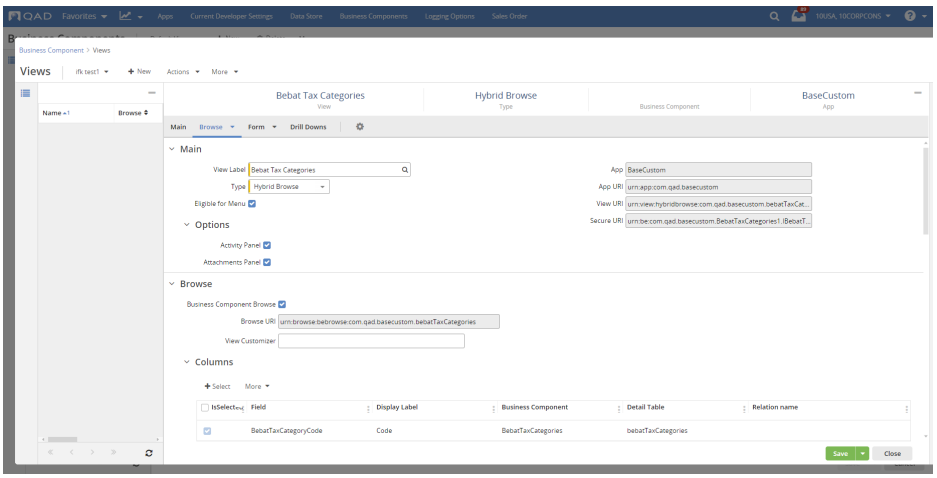


5. Now our form is ready. Click the **Save** button to save the form, and then the **Close** button.
6. After saving the form, we are back on the BC screen ready to create a Hybrid view.

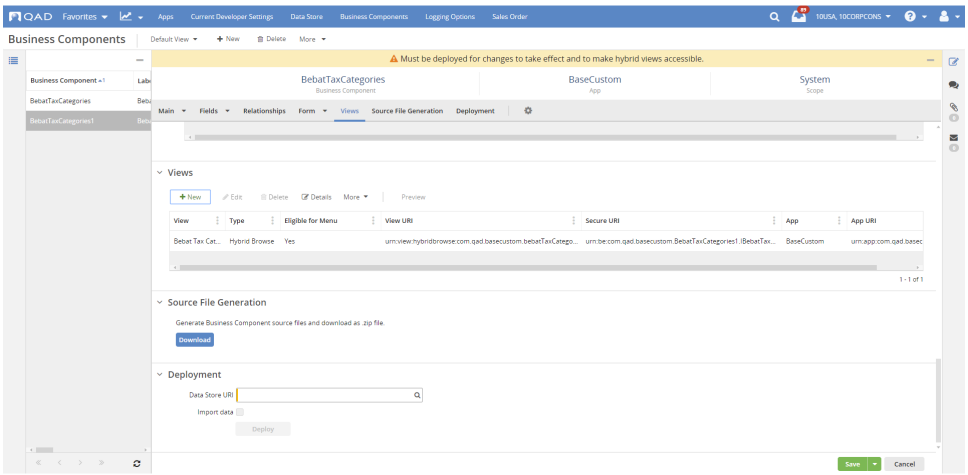
Create BebatTaxCategory Browse

In order to see the BebatTaxCategories BC in the menu and open a browse on it, we need to create a BC browse. This can be done by following these steps:

1. On the BebatTaxCategories BC screen, click the **Views** panel, and then click the **New** button. This will bring up a details screen where you can create the browse. On that screen, enter the following:
   - View Label: Bebat Tax Categories, this is the label the hybrid view will have on the menu.
   - In the browse columns grid, clear the BebatTaxCategoryDescription column. This column is too big to show in a browse, this way we remove it from the browse view.
   - Now the screen should look as follows.



2. Click the **Save** button to save this browse, and then **Close**.
3. After saving the new Hybrid view, the view builder UI looks as follows.

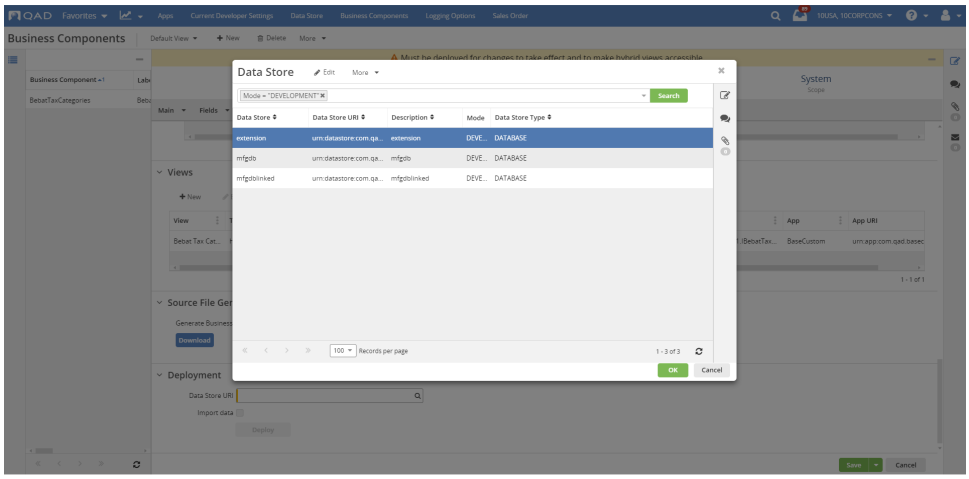


4. Click the **Save** button again to save changes.

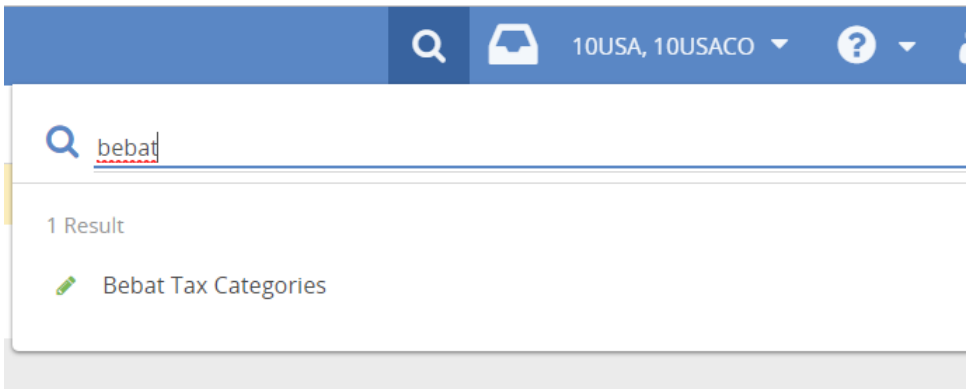
Deploy BebatTaxCategories Business Component

The last step for the BebatTaxCategories BC is to deploy it so that it becomes active in the application. To do this, follow these steps:

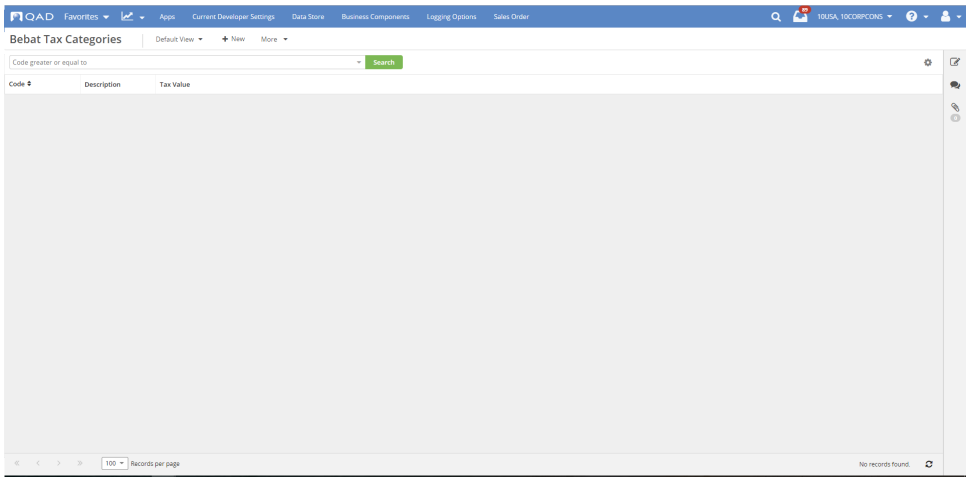
1. Open the Business Components Maintenance screen, search for the **BebatTaxCategories** business component, and then click the **Edit** button.
2. Click the **Deployment** panel and then click the lookup button on the **DataStore URI** field. This will open a lookup with the available data stores. Select your developer data store and click **OK**.
3. Click the **Deploy** button to deploy the BC and make it active.



4. Now that our BC is active, we can search for it in the menu:



5. And open it.



Create ItemRecycleTaxes Components

This section explains all the steps to create a DEO BC that is related to Recupel and Bebat tax components:

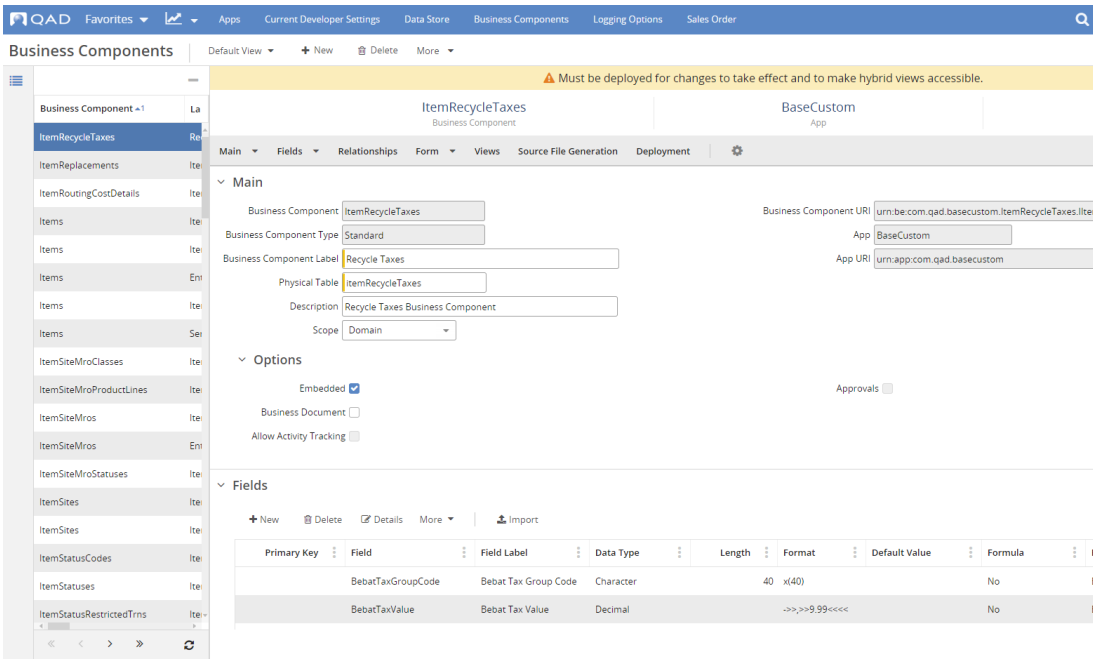
- Create ItemRecycleTaxes Data Embedded Business Component
- Create ItemRecycleTaxes lookup relations
- Create ItemRecycleTaxes one-to-one extension relation
- Add formula to calculate total price
- Deploy ItemRecycleTaxes Business Component

Create ItemRecycleTaxes Data Embedded Business Component

In order to extend Item maintenance with fields for the two recycle taxes, we create the ItemRecycleTaxes embedded BC. This BC will bring the RecupelTaxCategory and BebatCategory together with item. Follow these steps to create the Business Component:

1. Open **Business Components** and click **New**.
2. Enter the following values on the screen:
    - Business Component: The name of the BC: ItemRecycleTaxes
    - Business Component Label: Recycle Taxes
    - Physical Table: The name of the database table: ItemRecycleTaxes
    - Description: Recycle Taxes Business Component
    - Scope: Domain because item runs in domain context, the extension needs to do that too.
    - Embedded: Select
    - Fields (Name is max 32 characters):
        - DomainCode: Because we run in domain context, we need this field (label DomainCode, Type character, length 8, format x(80), key field 1)
        - ItemCode: This is the field that links back to the item (label Item Code, Type character, length 18, format x(18), key field 2)
        - TotalPrice: This field will be used to calculate the total price (label Total Price, Type decimal, formula)
        - RecupelTaxGroupCode: Recupel tax group code that links to RecupelTaxGroups BC (label Recupel Tax Group Code, Type character, length 40, format x(40))
        - RecupelTaxValue: Recupel tax group value: The value of the recupel tax (label Recupel Tax Value, Type Decimal)
        - BebatTaxGroupCode: Bebat tax group code that links to BebatTaxGroups BC (label Bebat Tax Group Code, Type character, length 40)
        - BebatTaxValue: Bebat tax group value: The value of the recupel tax (label Bebat Tax Value, Type Decimal)
3. Click **Save**.

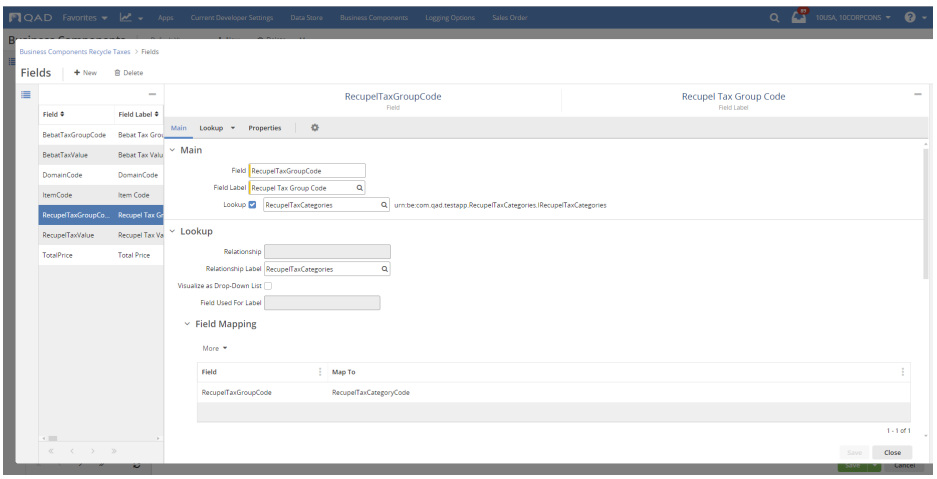
This is how the Business Component should look:



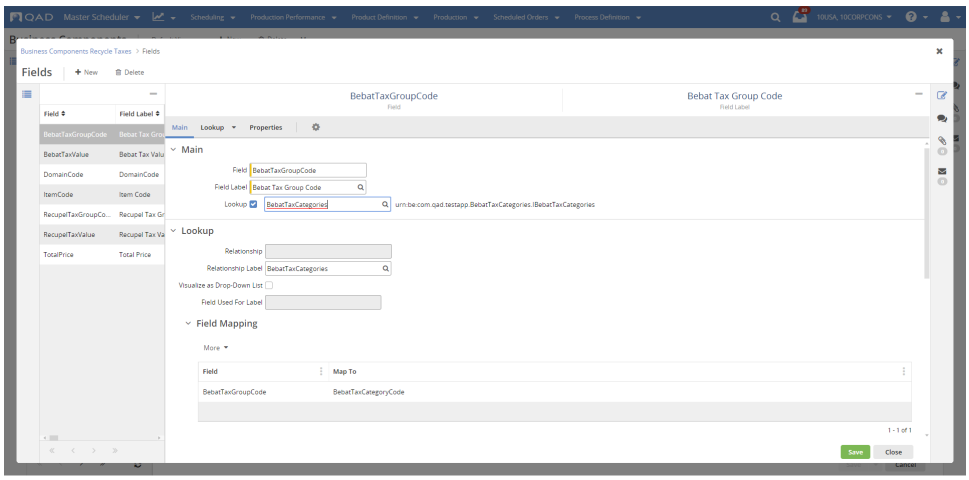
Create ItemRecycleTaxes lookup relations

The ItemRecycleTaxes extension BC displays Recupel and Bebat group code. For that reason, we need to add a 1-N relation to these 2 BCs we created earlier. This will give us the advantage of having automatically lookup buttons on these fields on the screen. Follow these steps to add the relations:

1. Open **Business Components**, search for the **ItemRecycleTaxes** BC, and then click **Edit**.
2. Go to the **Fields** panel, select the **RecupelTaxGroupCode** field, and then click the **Details** button. A new popup screen appears. On this screen, enter the following values:
    - Lookup: Select the checkbox and in the lookup select the **RecupelTaxCategories** BC



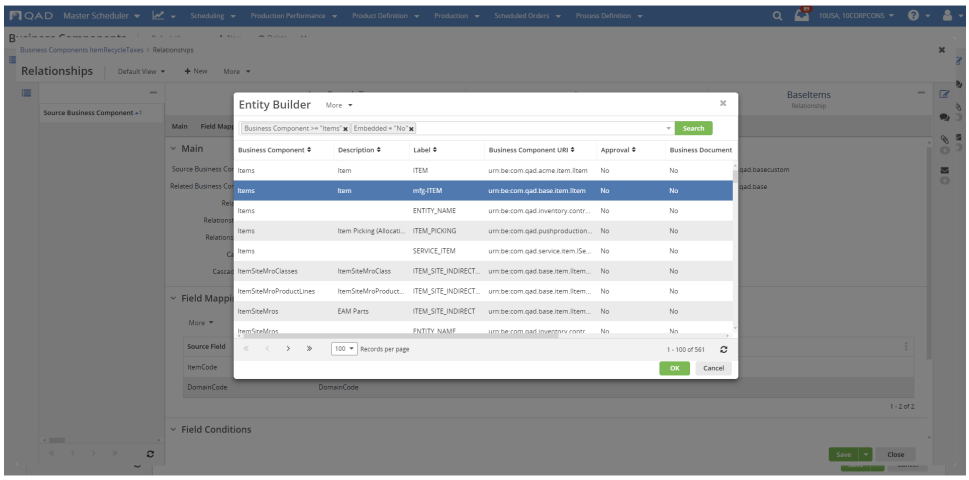
3. Click **Save**, and then click **Close**. This will bring you back to the Business Components screen.
4. We now have a relation to Recupel Taxes, but we need a second one for Bebat Taxes. Follow the same steps as for adding Recupel Taxes relation, but select BebatTaxCategories BC instead.



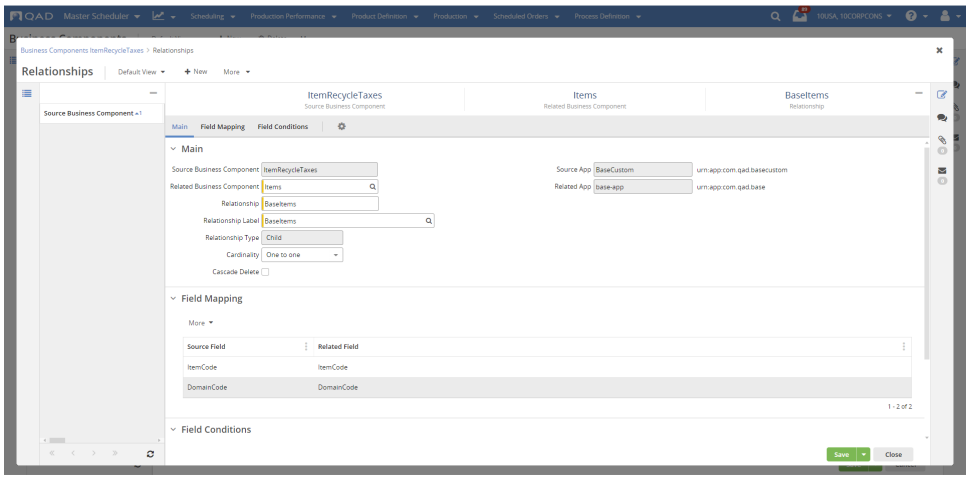
Create ItemRecycleTaxes one-to-one extension relation

In order to extend item with this BC, we need to add a one-to-one relation to the item. Follow these steps to do that:

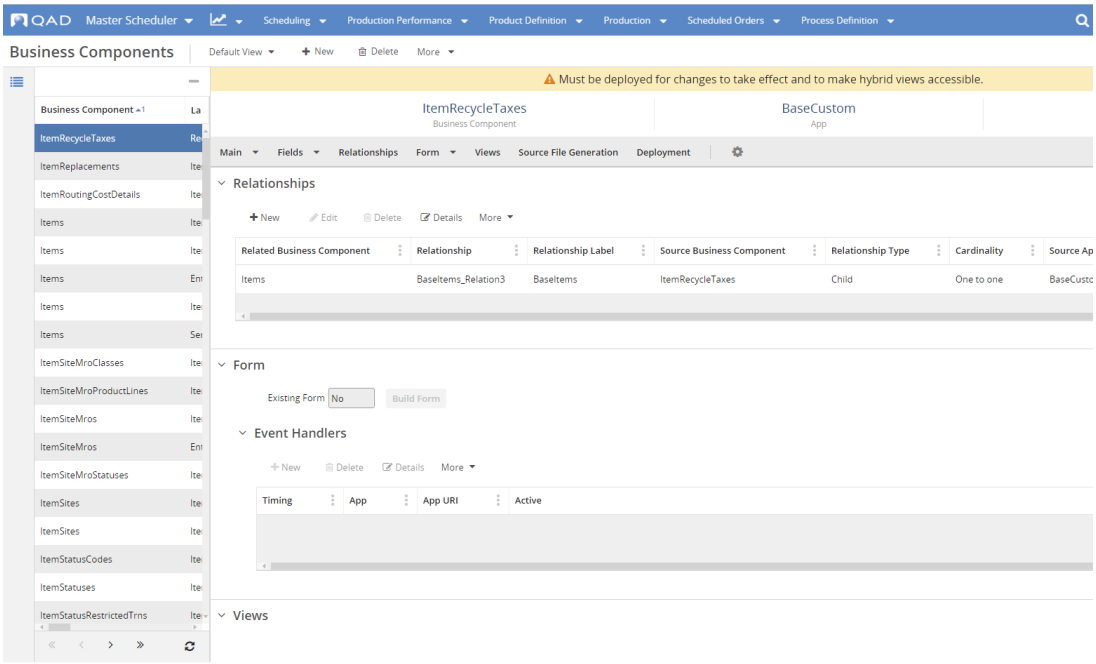
1. In **Business Components**, select **ItemRecycleTaxes**, and then click **Edit**.
2. Go to the **Relationships** panel and click **New**.
3. Click the Related Business Component lookup and select the **Items** business component. The rest of the fields automatically defaults to the correct value.



4. Click **Save** and **Close**.



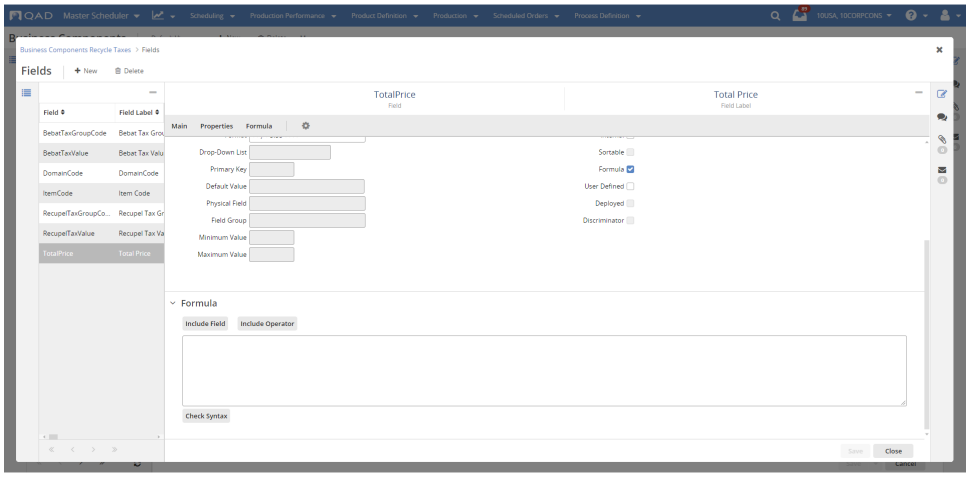
After that, your BC screen should look like this:


QAD
Master Scheduler
Scheduling
Production Performance
Product Definition
Production
Scheduled Orders
Process Definition
Business Components
Default View
New
Delete
More
Must be deployed for changes to take effect and to make hybrid views accessible.
ItemRecycleTaxes
Business Component
BaseCustom
App
Main
Fields
Relationships
Form
Views
Source File Generation
Deployment
Relationships
New
Edit
Delete
Details
More
Related Business Component
Relationship
Relationship Label
Source Business Component
Relationship Type
Cardinality
Items
BaseItems_Relation3
BaseItems
ItemRecycleTaxes
Child
One to one
Form
Existing Form
No
Build Form
Event Handlers
Timing
App
App URI
Active
Views

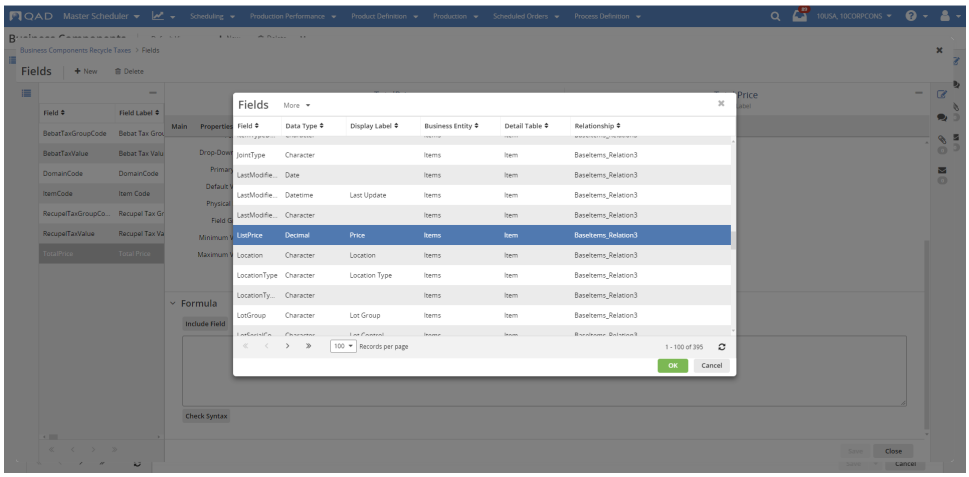
Add formula to calculate total price

On our extension, we will have a total price field. This is the price of the item + the 2 taxes. Here we explain how to add a formula field that calculates this total automatically.

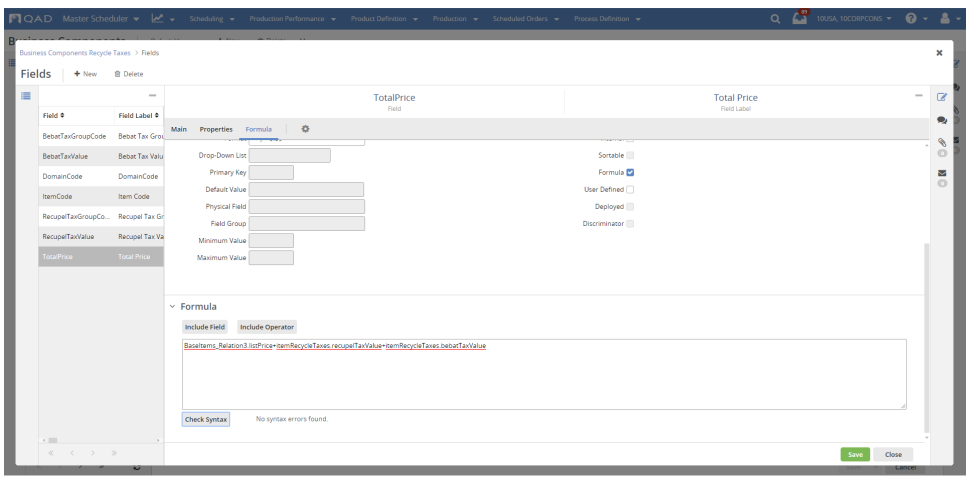
1. Open **Business Components**, select the **ItemRecycleTaxes** BC, and then click **Edit**.
2. Go to the **Fields** panel and select the **TotalPrice** field.
3. Change the **Formula** field to **Yes**.
4. Click the **Details** button, and then click the **Formula** panel.



5. Click the **Include Field** button and select the **listPrice** field.



6. Then enter + and click the **Include Field** button again and select the RecupelTaxValue field; do the same with BebatTaxValue so that you end up with this formula: BaseItems_Relation3.listPrice+itemRecycleTaxes.recupelTaxValue+itemRecycleTaxes.bebatTaxValue
7. Click the **Check Syntax** button.

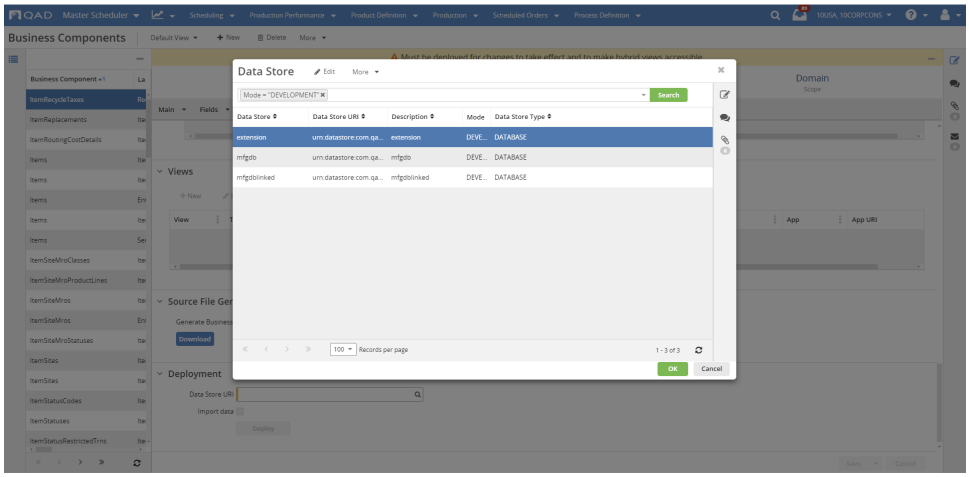


8. Click **Save** and **Close**.

Deploy ItemRecycleTaxes Business Component

The last step to make this Extension BC active is to deploy it. Follow these steps:

1. Open **Business Components**, search for the **ItemRecycleTaxes** BC, and then click **Edit**.
2. Click the **Deployment** panel, and then click the lookup button on the **DataStore URI** field. This will open a lookup with the available data stores. Select your developer data store and click **OK**.

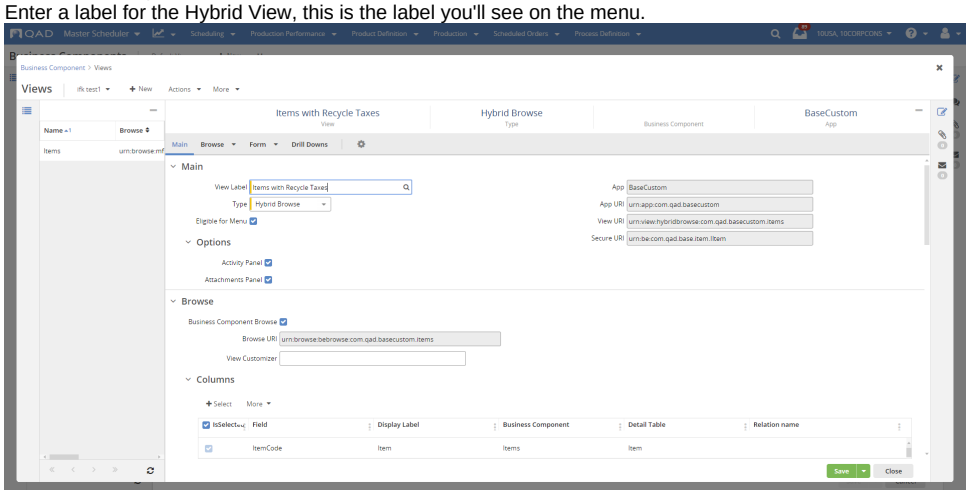


3. Click the **Deploy** button to deploy the BC and make it active.

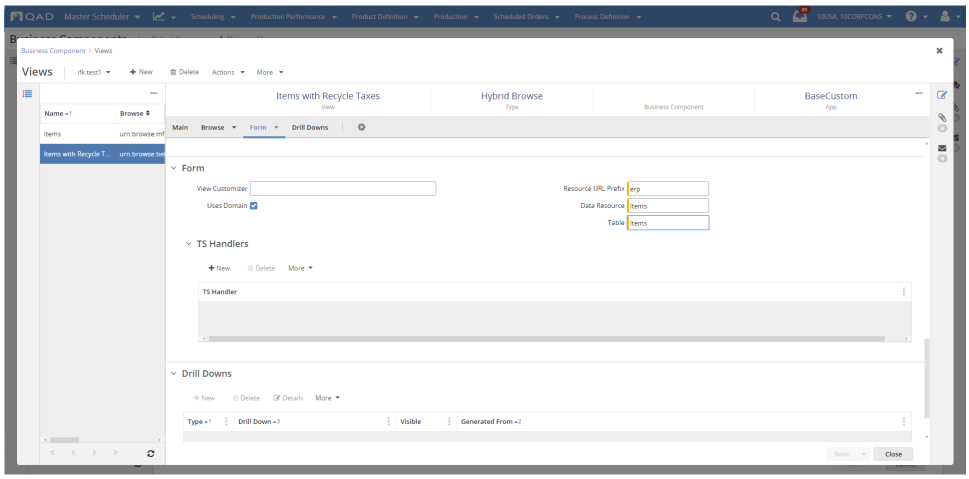
Create Items with Recycle Taxes BC browse

On this page, we show how to create a Business Component browse that contains fields from the BC itself, but also from its extension.

1. Open **Business Components**, search for the **Items** BC, and then click **Edit**.
2. Click the Views panel, and then click the **New** button.
3. Enter a label for the Hybrid View, this is the label you'll see on the menu.



4. After that, clear all columns in the grid except the key fields and the description field. Then click the **+Select** button and select the extension fields too. You should see them now in the **Fields** grid.
5. After that, fill in the following values (Resource URL Prefix and Data Resource and Table).

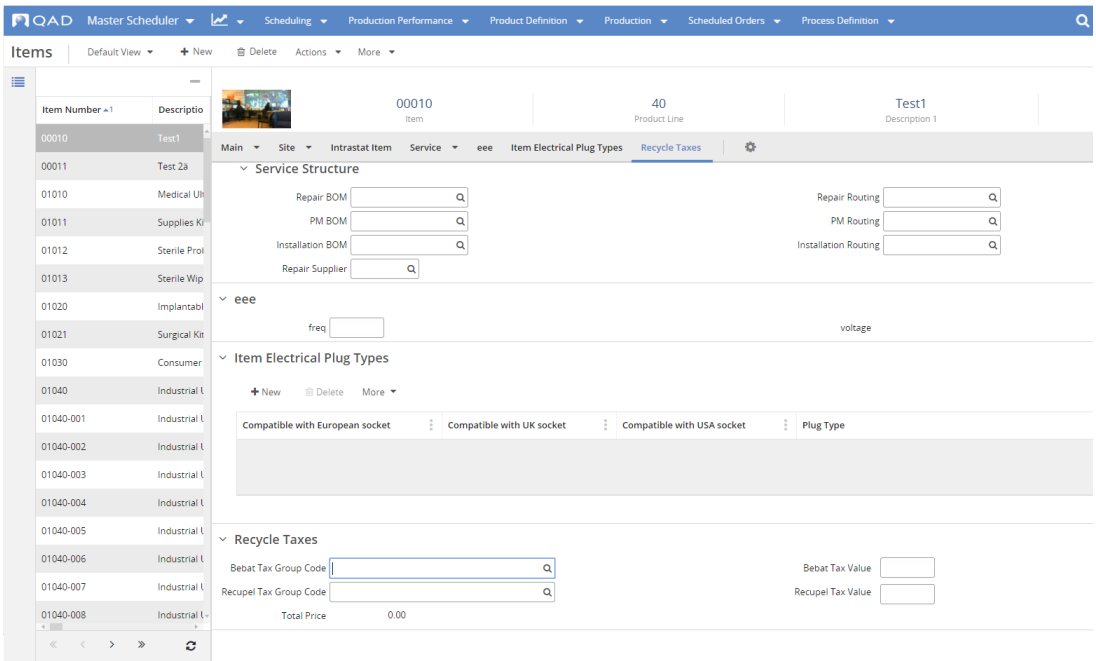


6. Click **Save** and **Close**.

That's it, now you should be able to run the browse from the menu.

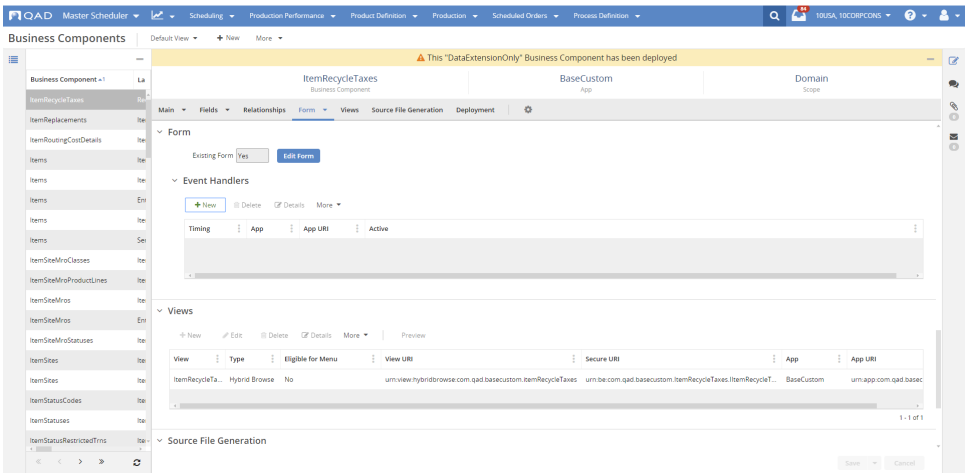
Make tax value fields read only on view

After deploying the **ItemRecycleTaxes** business component, we should see the fields from ItemRecycleTaxes on the Item view.

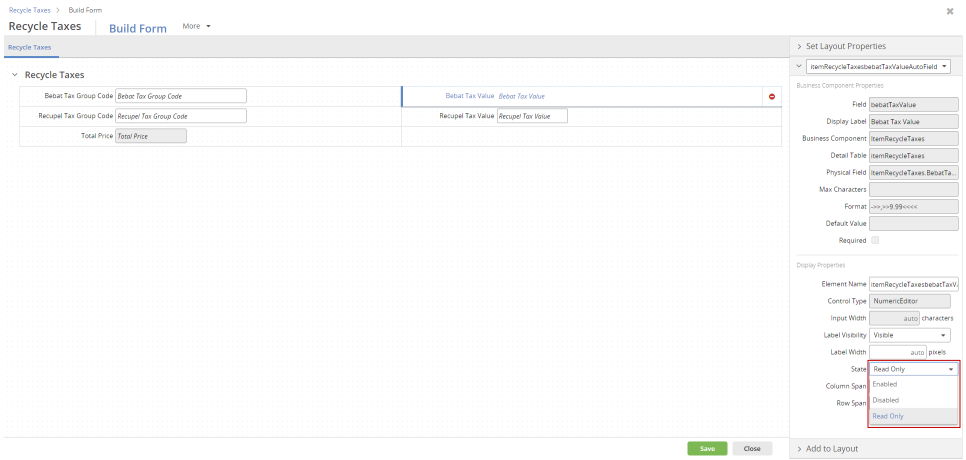


Because the Tax Value fields need to be automatically filled when a Group Code is selected, we need to make them read-only. This can be done using the form builder:

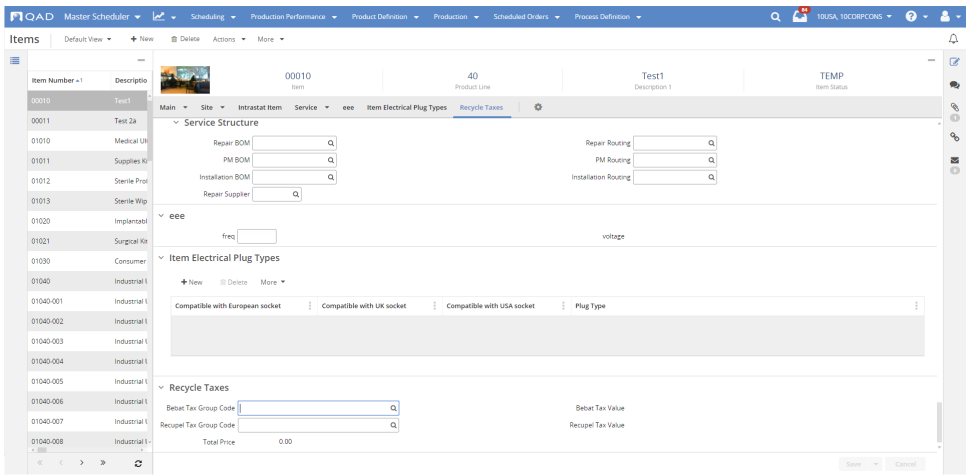
1. Open **Business Components**, search for the **ItemRecycleTaxes** BC, and then click **Edit**.
2. Click the **Form** panel and then the **Edit Form** button.



3. Select the **Bebat Tax Value** field and go to **Display Properties** at the right.



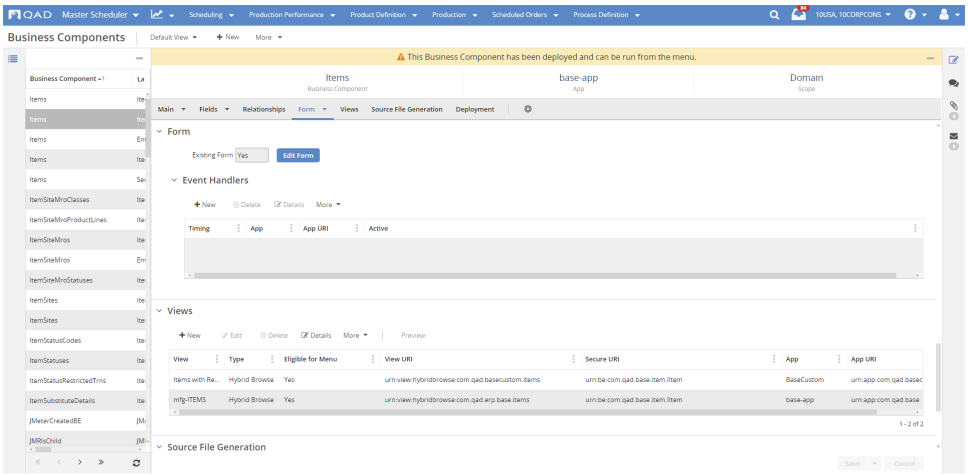
4. From the **State** drop-down menu, select **Read Only**.
5. Do the same for the **Recupel Tax Value** field.
6. Click **Save** and **Close**.
7. Now go back to the Items screen but be sure to click the **Refresh** button from the browses, so that it completely reloads. Open an item, and see that the 2 fields are read-only now.



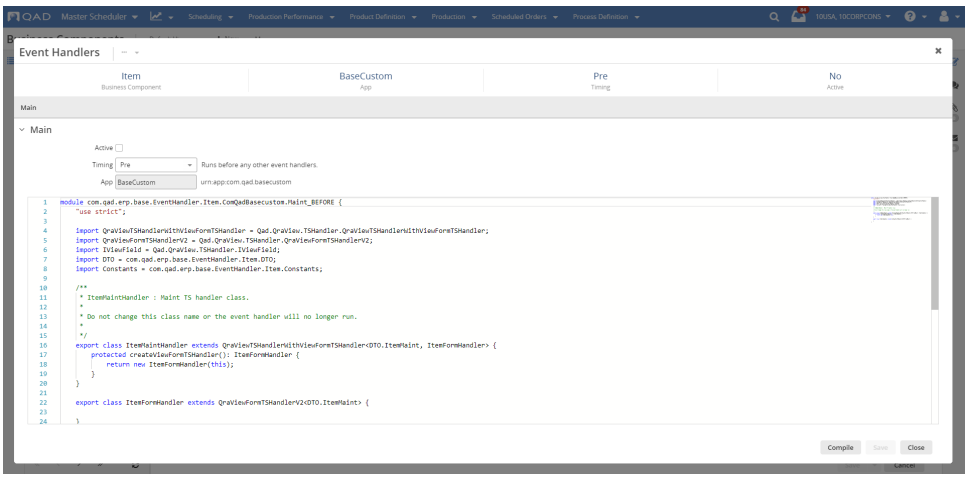
Add event handler to retrieve Tax Value data

At this point, we have Items extended with the 2 tax code and tax price fields. However, what is not working is that when we select another tax code via the lookup, the tax value fields are not automatically updated. For this, we will add an event handler that will retrieve the tax value fields values from the server with http, and then update the UI with the retrieved values. This can be done by doing the following steps:

1. Go to **Business Components**, search for **Items**, and then click the **Edit** button.



2. On the **Event Handlers** panel, click the **New** button.



3. As you can see on the above screenshot, the initial type is a PRE type. It does not matter so much in this case, but usually, we want our custom code to run AFTER all the other UI logic. So, we change the type to POST.

Now we are ready to change the code so that when one of the tax code fields change, we retrieve the corresponding tax value from the server and update the UI with these values. In the initial code, you can see the 2 generated classes, namely ItemMaintHandler and ItemFormHandler. ItemMaintHandler is the class for the main TS handler, it can handle general UI events. ItemFormHandler will handle all events related to fields on the screen. We need to extend this code with the following:

1. Add a method that takes a tax code and calls the server to retrieve the tax value:
   For this, we will add a method getRecycleTaxValue to ItemFormHandler. This method will take the necessary parameters and do an http call to the server:

```
        /**
         * @function getRecycleTaxValue: retrieve Recupel or Bebat tax data from the server and
return the value via the callback method. When the data is not found, an error will be displayed
in the error viewer.
         *
         * @param domainCode : domain code
         * @param taxCode: can be "Recupel" or "Bebat", all other values will be ignored
         * @param recycleTaxCategoryCode : tax code
         * @param callback: callback function that will be called when the result is received
```

```
from the server.
         */
        private getRecycleTaxValue(domainCode: string, taxCode: string,recycleTaxCategoryCode:
string,callback: (recupelTaxPrice: string)=>void) {
            let targetUrl: string;
            let finalUrl: string;
            if (taxCode==="Recupel") {
                targetUrl="api/qracore/be/urn:be:com.extensions.qadextensions.
RecupelTaxCategories.IRecupelTaxCategories?domainCode={domainCode}&recupelTaxCategoryCode=
{recupelTaxCategoryCode}";
                //replace placeholders in the url with the domain code and tax code
                finalUrl=bindTemplateData(targetUrl, {domainCode: domainCode,
recupelTaxCategoryCode: recycleTaxCategoryCode});
            }
            else if (taxCode==="Bebat") {
                targetUrl="api/qracore/be/urn:be:com.extensions.qadextensions.BebatTaxCategories.
IBebatTaxCategories?domainCode={domainCode}&bebatTaxCategoryCode={bebatTaxCategoryCode}";
                //replace placeholders in the url with the domain code and tax code
                finalUrl=bindTemplateData(targetUrl, {domainCode: domainCode,
bebatTaxCategoryCode: recycleTaxCategoryCode});
            }

            if (finalUrl) {
                this.ViewController.doHttpGet(
                    finalUrl,
                    (response: Qad.Common.DTO.DataResult) => {
                        let data = response.data;
                        if (data) {
                            if (taxCode==="Recupel") {
                                if (data.recupelTaxCategories && data.recupelTaxCategories.
length>0) {
                                    //return recupel tax value via callback function
                                    callback(data.recupelTaxCategories[0].
recupelTaxCategoryTaxValue);
                                }
                                else {
                                    this.handleServerError(undefined,"Error retrieving Recupel
tax:","not found");
                                }
                            }
                            else if (taxCode==="Bebat") {
                                if (data.bebatTaxCategories && data.bebatTaxCategories.length>0) {
                                    //return recupel tax value via callback function
                                    callback(data.bebatTaxCategories[0].bebatTaxCategoryTaxValue);
                                }
                                else {
                                    this.handleServerError(undefined,"Error retrieving Bebat
tax:","not found");
                                }
                            }
                        }
                        else {
                            this.handleServerError(undefined,"Error retrieving " + taxCode + "
tax:","not found");
                        }


                    },
                    (data, status) => this.handleServerError(data,undefined,undefined),
                    null,
                    null,
                    true,
                    false,
                    true,
                    false,
                    false
                );
            }
        }
```

2. Notice the handleServerError method. That method will be used to show an error on the UI:

```
        /**
         * @function handleServerError : display error information in error viewer
         *
         * @param submitresults : array of SubmitResult objects
         * @param message : message to show
```

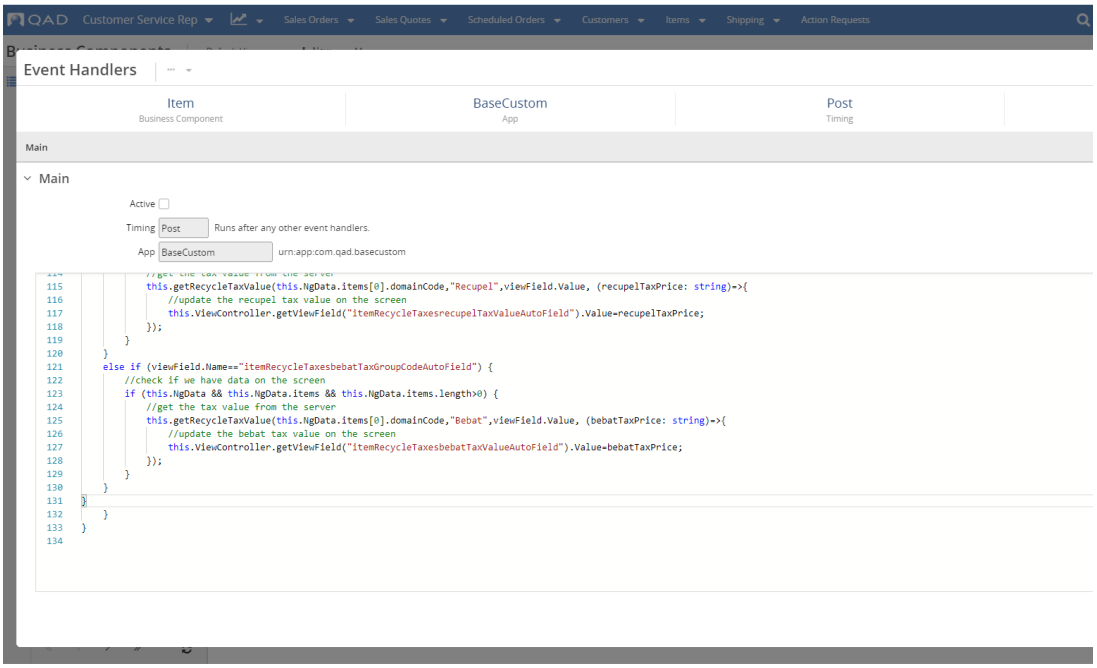
```
         * @param status : status to show in the message
         */
        private handleServerError(submitresults: Qad.Common.DTO.SubmitResult , message:string,
status: string) {
            let serverErrors: Qad.Common.DTO.Error[] = [];
            if (submitresults.errors && submitresults.errors.length && submitresults.errors.
length>0) {
                serverErrors = submitresults.errors;
            } else {
                serverErrors.push(new Qad.Common.DTO.Error());
                serverErrors[0].message = "" + message + " " + status;
                serverErrors[0].severity = 1;
            }
            this.ViewController.ErrorGroupPanel.clearErrorGrid();
            this.ViewController.ErrorGroupPanel.addErrorsToErrorGrid(serverErrors);
            this.ViewController.ErrorGroupPanel.showErrorGrid();
        }
```

3. Add an event handler that acts when one of the tax code fields change:

```
        /**
         * onFieldChange event handler
         */
        public onFieldChange(viewField: IViewField<any>, eventData: Communication.EventData.
QraView.FieldChangeEventData<any>, processEvent: (processIt?: boolean) => void): void{
            if (viewField.Name=="itemRecycleTaxesrecupelTaxGroupCodeAutoField") {
                //check if we have data on the screen
                if (this.NgData && this.NgData.items && this.NgData.items.length>0) {
                    //get the tax value from the server
                    this.getRecycleTaxValue(this.NgData.items[0].domainCode,"Recupel",viewField.
Value, (recupelTaxPrice: string)=>{
                        //update the recupel tax value on the screen
                        this.ViewController.getViewField
("itemRecycleTaxesrecupelTaxValueAutoField").Value=recupelTaxPrice;
                    });
                }
            }
            else if (viewField.Name=="itemRecycleTaxesbebatTaxGroupCodeAutoField") {
                //check if we have data on the screen
                if (this.NgData && this.NgData.items && this.NgData.items.length>0) {
                    //get the tax value from the server
                    this.getRecycleTaxValue(this.NgData.items[0].domainCode,"Bebat",viewField.
Value, (bebatTaxPrice: string)=>{
                        //update the bebat tax value on the screen
                        this.ViewController.getViewField
("itemRecycleTaxesbebatTaxValueAutoField").Value=bebatTaxPrice;
                    });
                }
            }
        }
```

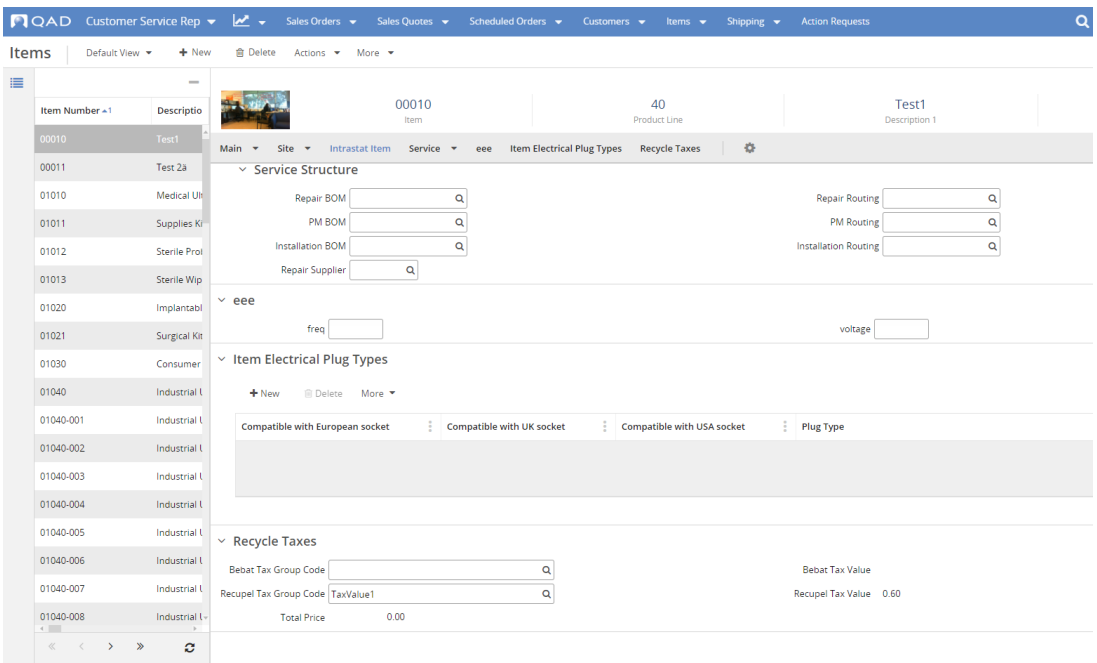
As you can see above, we implement the onFieldChange event by simply overriding the base class method. Inside the method, we check the field name and call the getRecycleTaxValue method to retrieve the tax value. On return of that value, we get a reference to the right tax value field and update its value.

4. Set the event handler **Active**, click the **Compile** button and **Save**:



5. After this we are ready to test: open items maintenance, select a record and click **Edit**. Then, go the **Recycle Taxes** tab and click a lookup button.

6. After selecting a record in the lookup and clicking **OK**, the tax value field should be showing the correct value:



Add BL validation to check if one of the tax categories is used when deleting

Overview

This page provides an overview of what needs to be done to add OOABL code to check if a tax category can be deleted, if it is used in one of the defined items.

In this page, we assume the extension app is called "hackathon", with URI = "urn:app:com.extensions.hackathon"

Step 1: Export the app to make it ready for OOABL code extensions

In your environment, run the following command:

yab app-export -dir:<the folder you want to export your app to> -appuri:<the uri of the app you want to export>

In our example, we move to the root folder of our environment and run this:

yab app-export -dir:./hackathon -appuri:urn:app:com.extensions.hackathon

```
                      app-export (8 tasks)                              [APPLY]
------------------------------------------------------------------------------
1/8 app-export                                              OK (0.008 s)
2/8 metadata-all-export                                     OK (1.224 s)
3/8 app-package-metadata-create                             OK (0.077 s)
4/8 app-export-dependency-update                            OK (3.044 s)
5/8 action-center-dashboard-extract                         OK (2.017 s)
6/8 action-center-kpi-files-extract                         OK (7.753 s)
7/8 action-center-kpi-extract                               OK (4.912 s)
8/8 app-schema-dump                                         OK (0.136 s)
------------------------------------------------------------------------------

BUILD SUCCESSFUL (19.978 s)
```

Step 2: Tell YAB about your new app

Edit build/config/configuration.properties in your environment and add the following line:

platform-extension.<appname>.dir=<folder where you exported the app>

In our example, that is:

platform-extension.hackathon.dir=/dr01/qadapps/systest/hackathon

```
#-----------------------------------------------------------------
# configuration.properties
#
# This file can be used to overwrite any existing YAB configuration
# settings or to add/upgrade new packages to an environment.
#
# Example: Overwriting an existing property
# netui.telnet.user=odnetui
# appserver.fin.maxsrvrinstance=10
# netui.telnet.maxconnections=200
#
# Example: Add/Upgrade new packages to an environment
# packages.avataxeeconnector=1.2.3.4
#
# Note: To apply any changes to an environment execute:
```

```
# yab update
#-----------------------------------------------------------------

platform-extension.hackathon.dir=/dr01/qadapps/systest/hackathon
```

Step 3: First add a class to define your messages

It is a normal practice that messages are kept in a separate class. For specific messages used in the hackathon project, you should create the following class in the app code location for the hackathon project.

1. In your yab environment, go to the folder where your customizations will be kept (this is typically "<FolderWhereYouExported>/src/impl/com/<env namespace>/<appname>"). In our case here, that is hackathon /src/impl/com/extensions/hackathon.
2. Now, add a file called "HackathonMessageKey.cls". This is a class that will hold all message keys that can be used in your code.

src/impl/com/extensions/hackathon/HackathonMessageKey.cls

```
using com.qad.lang.*.
using com.extensions.hackathon.HackathonMessageKey.

routine-level on error undo, throw.

class com.extensions.hackathon.HackathonMessageKey final inherits MessageKey:

    &scope TYPE HackathonMessageKey
    {com/qad/lang/MessageKeyConstructors.i}

    { com/qad/lang/MessageKey.i &NUM=1 &PROP=ITEM_EXISTS_WITH_RECUPEL                  &LITERAL="
Item referencing recupel tax category (&1) still exists, cannot delete recupel tax category." }

end class.
```

Step 4: Write a validation on the deletion of a Recupel tax category

1. In your yab environment, go to the folder where your customizations will be created, in our case here, that is hackathon/src/impl/com/extensions/hackathon.
2. All data extensions are handled by VirtualBusinessEntity so we will have to customize that.
3. Create a subfolder for the customization you're going to do. Since I'm going to customize VirtualBusinessEntity, I create a subfolder be.
4. Write the custom code, in our case in VirtualBusinessEntity.cls.

src/impl/com/extensions/hackathon/be/VirtualBusinessEntity.cls

```
using com.qad.lang.List.
using com.qad.lang.Message.

using com.qad.qra.be.IVirtualBusinessEntity.
using com.qad.qra.be.VirtualBusinessEntityBase.

using com.qad.base.BaseError.
using com.qad.base.BaseMessageKey.
using com.qad.base.BaseServices.

routine-level on error undo, throw.

class com.extensions.hackathon.be.VirtualBusinessEntity inherits VirtualBusinessEntityBase
implements IVirtualBusinessEntity:
    method public override void Delete(
        input entityURI as character,
        input dataset-handle dsEntity):

        define variable buf     as handle             no-undo.
        define variable qry     as handle             no-undo.
        define variable valMsgs as List               no-undo.
        define variable msg     as Message            no-undo.

        /* Only Recupel Tax Categories that are not referenced by items can be deleted */
        if entityURI = "urn:be:com.extensions.hackathon.RecupelCategory.IRecupelCategory"
        then do on error undo, throw:
```

```
            assign valMsgs = new List()
                   buf     = dsEntity:get-buffer-handle("ttRecupelCategory").

            create query qry.
            qry:set-buffers(buf).
            qry:query-prepare("for each ttRecupelCategory").
            qry:query-open().
            qry:get-first().

            do while not qry:query-off-end:
                find first ItemRecycleTax where ItemRecycleTax.RecupelTaxCode = ttRecupelCategory.
RecupelCode no-lock no-error.

                if not available (ItemRecycleTax)
                then do:
                    assign msg = new Message(BaseMessageKey:INVALID_CURRENCY, buf::RecupelCode).
                    msg:SetContext(buf:buffer-field("RecupelCode")).
                    valMsgs:Add(msg).
                end.

                qry:get-next().
            end.

            if not valMsgs:IsEmpty()
            then undo, throw new BaseError(valMsgs).

            finally:
                if qry:is-open
                then qry:query-close().

                delete object qry.
                assign qry = ?.
            end.
        end.

        /* Call the standard code */
        super:Delete(
            input entityURI,
            input dataset-handle dsEntity by-reference).
    end method.

end class.
```

Step 5: Compile the code

In your environment, run the following command:

yab dev-<extractfolder>-update

In our case, that is:

yab dev-hackathon-update

```
                dev-hackathon-update (14 tasks)                     [APPLY]
------------------------------------------------------------------------------
1/14  database-dev-hackathon-extension-structure-list       OK (0.036 s)
2/14  database-dev-hackathon-extension-structure-file-updat OK (0.014 s)
3/14  database-dev-hackathon-extension-structure-validate   OK (0.005 s)
4/14  database-dev-hackathon-extension-create               OK (0.003 s)
5/14  database-dev-hackathon-extension-structure-update     OK (0.017 s)
6/14  database-dev-hackathon-extension-schema-update        OK (0.112 s)
7/14  database-dev-hackathon-extension-schema-snapshot      SKIPPED (0.003 s)
8/14  database-dev-hackathon-extension-data-xml-update      OK (0.013 s)
9/14  database-dev-hackathon-extension-data-dotd-update     OK (0.035 s)
10/14 dev-hackathon-init-check                              OK (0.003 s)
11/14 code-dev-hackathon-db-start-stop                      SKIPPED (0.001 s)
12/14 code-dev-hackathon-update                             OK (0.374 s)
13/14 code-dev-hackathon-db-start-stop                      SKIPPED (0.001 s)
14/14 prolib-dev-hackathon-create                           OK (0.008 s)
------------------------------------------------------------------------------

BUILD SUCCESSFUL (1.306 s)
```

Step 6: Add this new implementation of IVirtualBusinessEntity to module-config.xml

Go to the folder hackathon/config and add the new IVirtualBusinessEntity to module-config.xml.

module-config.xml

```
<Module>
    <ModuleInfo
            Key="extensions.hackathon"
            Version="@module.version@"
            Vendor="@module.vendor@"
            VendorUrl="@module.vendorurl@"
            DisplayName="Hackathon App"
            Description="Hackathon sample app"
            Date="@module.date@"
            ClassName="com.extensions.hackathon.Module"
            Uri="urn:app:com.extensions.hackathon"/>

    <Service
        ServiceClass="com.qad.qra.be.IVirtualBusinessEntity"
        ServiceKey=""
        ImplClass="com.extensions.hackathon.be.VirtualBusinessEntity"
        LifetimePolicy="factory" />
</Module>
```

Step 7: Register the new code in your environment

In your environment, run the following command:

yab dev-<extractfolder>-code-register

In our case, that is:

yab dev-hackathon-code-register

```
              dev-basecustom-code-register (7 tasks)                  [APPLY]
-------------------------------------------------------------------------------
1/7 dev-hackathon-code-register                          OK (0.074 s)
2/7 appserver-fin-trim                                   OK (0.340 s)
3/7 appserver-mfg-trim                                   OK (0.862 s)
4/7 appserver-qra-trim                                   OK (0.610 s)
5/7 appserver-qxosi-trim                                 OK (0.356 s)
6/7 appserver-qxoui-trim                                 OK (0.307 s)
7/7 appserver-qxtnative-trim                             OK (0.308 s)
-------------------------------------------------------------------------------

BUILD SUCCESSFUL (3.378 s)
```

Step 8: Test

Ref: complete event handler code for the Items BC

```
module com.qad.erp.base.EventHandler.Item.ComQadBasecustom.Maint_BEFORE {
    "use strict";


    import QraViewTSHandlerWithViewFormTSHandler = Qad.QraView.TSHandler.
QraViewTSHandlerWithViewFormTSHandler;
    import QraViewFormTSHandlerV2 = Qad.QraView.TSHandler.QraViewFormTSHandlerV2;
    import IViewField = Qad.QraView.TSHandler.IViewField;
    import DTO = com.qad.erp.base.EventHandler.Item.DTO;
    import Constants = com.qad.erp.base.EventHandler.Item.Constants;

    /**
     * ItemMaintHandler : Maint TS handler class.
     *
     * Do not change this class name or the event handler will no longer run.
     *
     */
    export class ItemMaintHandler extends QraViewTSHandlerWithViewFormTSHandler<DTO.ItemMaint,
ItemFormHandler> {
        protected createViewFormTSHandler(): ItemFormHandler {
            return new ItemFormHandler(this);
        }
    }

    export class ItemFormHandler extends QraViewFormTSHandlerV2<DTO.ItemMaint> {

        /**
         * onFieldChange event handler
         */
        public onFieldChange(viewField: IViewField<any>, eventData: Communication.EventData.
QraView.FieldChangeEventData<any>, processEvent: (processIt?: boolean) => void): void{
            if (viewField.Name=="itemRecycleTaxesrecupelTaxGroupCodeAutoField") {
                //check if we have data on the screen
                if (this.NgData && this.NgData.items && this.NgData.items.length>0) {
                    //get the tax value from the server
                    this.getRecycleTaxValue(this.NgData.items[0].domainCode,"Recupel",viewField.
Value, (recupelTaxPrice: string)=>{
                        //update the recupel tax value on the screen
                        this.ViewController.getViewField
("itemRecycleTaxesrecupelTaxValueAutoField").Value=recupelTaxPrice;
                    });
                }
            }
            else if (viewField.Name=="itemRecycleTaxesbebatTaxGroupCodeAutoField") {
                //check if we have data on the screen
                if (this.NgData && this.NgData.items && this.NgData.items.length>0) {
                    //get the tax value from the server
                    this.getRecycleTaxValue(this.NgData.items[0].domainCode,"Bebat",viewField.
Value, (bebatTaxPrice: string)=>{
                        //update the bebat tax value on the screen
                        this.ViewController.getViewField
("itemRecycleTaxesbebatTaxValueAutoField").Value=bebatTaxPrice;
                    });
                }
            }
        }
        /**
         * @function getRecycleTaxValue: retrieve Recupel or Bebat tax data from the server and
return the value via the callback method. When the data is not found, an error will be displayed
in the error viewer.
         *
         * @param domainCode : domain code
         * @param taxCode: can be "Recupel" or "Bebat", all other values will be ignored
         * @param recycleTaxCategoryCode : tax code
         * @param callback: callback function that will be called when the result is received
from the server.
         */
        private getRecycleTaxValue(domainCode: string, taxCode: string,recycleTaxCategoryCode:
string,callback: (recupelTaxPrice: string)=>void) {
            let targetUrl: string;
            let finalUrl: string;
            if (taxCode==="Recupel") {
```

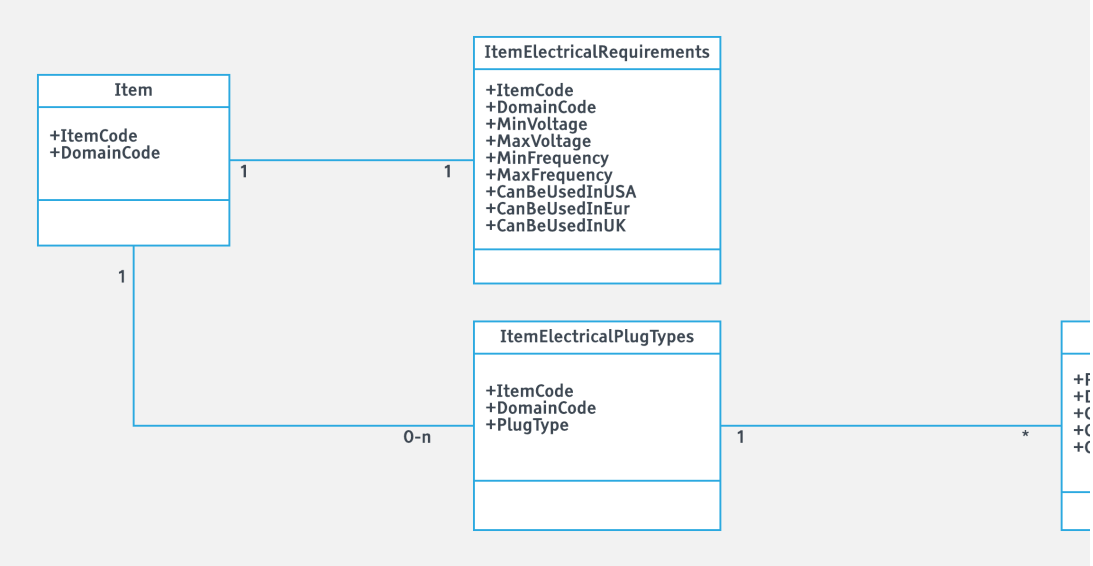
```
                targetUrl="api/qracore/be/urn:be:com.extensions.qadextensionsP.
RecupelTaxCategories.IRecupelTaxCategories?domainCode={domainCode}&recupelTaxCategoryCode=
{recupelTaxCategoryCode}";
                //replace placeholders in the url with the domain code and tax code
                finalUrl=bindTemplateData(targetUrl, {domainCode: domainCode,
recupelTaxCategoryCode: recycleTaxCategoryCode});
            }
            else if (taxCode==="Bebat") {
                targetUrl="api/qracore/be/urn:be:com.extensions.qadextensionsP.BebatTaxCategories.
IBebatTaxCategories?domainCode={domainCode}&bebatTaxCategoryCode={bebatTaxCategoryCode}";
                //replace placeholders in the url with the domain code and tax code
                finalUrl=bindTemplateData(targetUrl, {domainCode: domainCode,
bebatTaxCategoryCode: recycleTaxCategoryCode});
            }

            if (finalUrl) {
                this.ViewController.doHttpGet(
                    finalUrl,
                    (response: Qad.Common.DTO.DataResult) => {
                        let data = response.data;
                        if (data) {
                            if (taxCode==="Recupel") {
                                if (data.recupelTaxCategories && data.recupelTaxCategories.
length>0) {
                                    //return recupel tax value via callback function
                                    callback(data.recupelTaxCategories[0].
recupelTaxCategoryTaxValue);
                                }
                                else {
                                    this.handleServerError(undefined,"Error retrieving Recupel
tax:","not found");
                                }
                            }
                            else if (taxCode==="Bebat") {
                                if (data.bebatTaxCategories && data.bebatTaxCategories.length>0) {
                                    //return recupel tax value via callback function
                                    callback(data.bebatTaxCategories[0].bebatTaxCategoryTaxValue);
                                }
                                else {
                                    this.handleServerError(undefined,"Error retrieving Bebat
tax:","not found");
                                }
                            }
                        }
                        else {
                            this.handleServerError(undefined,"Error retrieving " + taxCode + "
tax:","not found");
                        }


                    },
                    (data, status) => this.handleServerError(data,undefined,undefined),
                    null,
                    null,
                    true,
                    false,
                    true,
                    false,
                    false
                );
            }
        }
        /**
         * @function handleServerError : display error information in error viewer
         *
         * @param submitresults : array of SubmitResult objects
         * @param message : message to show
         * @param status : status to show in the message
         */
        private handleServerError(submitresults: Qad.Common.DTO.SubmitResult , message:string,
status: string) {
            let serverErrors: Qad.Common.DTO.Error[] = [];
            if (submitresults.errors && submitresults.errors.length && submitresults.errors.
length>0) {
                serverErrors = submitresults.errors;
            } else {
                serverErrors.push(new Qad.Common.DTO.Error());
                serverErrors[0].message = "" + message + " " + status;
                serverErrors[0].severity = 1;
```

```
            }
            this.ViewController.ErrorGroupPanel.clearErrorGrid();
            this.ViewController.ErrorGroupPanel.addErrorsToErrorGrid(serverErrors);
            this.ViewController.ErrorGroupPanel.showErrorGrid();
        }
    }
}
```

Example 1.2: Adding new functionality to item maintenance with a 1-N extension

We will add to item information about the electrical compatibility of the item. We add some information about the voltage the item can work with, but also about the electrical plug(s) that are delivered with the item.

In a schematic view, this is what we will create:



Important here is that the 1-1 relation between Item and ItemElectricalRequirements is an extension relation (because we want to add extra fields on the items UI). Also the 1-N relation between Item and ItemElectricalPlugTypes is an extension relation (to add a grid to the items UI).

The 1-N relations to ElectricalPlugTypes is a lookup relation. It makes sure that we will see a lookup on the Plugin Type column of the grid that we are adding to items. See Business Component Relation for more info on the different types of relations.

The app we will create will allow the following what concerns electrical info:

1. Maintain electrical plug types.
2. Extend item with the ability to:
    a. Enter and view Electrical requirements info.
    b. Maintain the different electrical plugs that are delivered with the item.

- Create ElectricalPlugTypes Components
- Create ItemElectricalPlugTypes Components
- Create ItemElectricalRequirements Components
- Add event handler to retrieve electrical plug info
- Add event handler to update electrical requirements panel

Create ElectricalPlugTypes Components

This section explains all the steps to create Business Components for Electrical plug types:

- Create ElectricalPlugTypes Business Component
- Create ElectricalPlugTypes View
- Create ElectricalPlugTypes Browse
- Deploy ElectricalPlugTypes Business Component

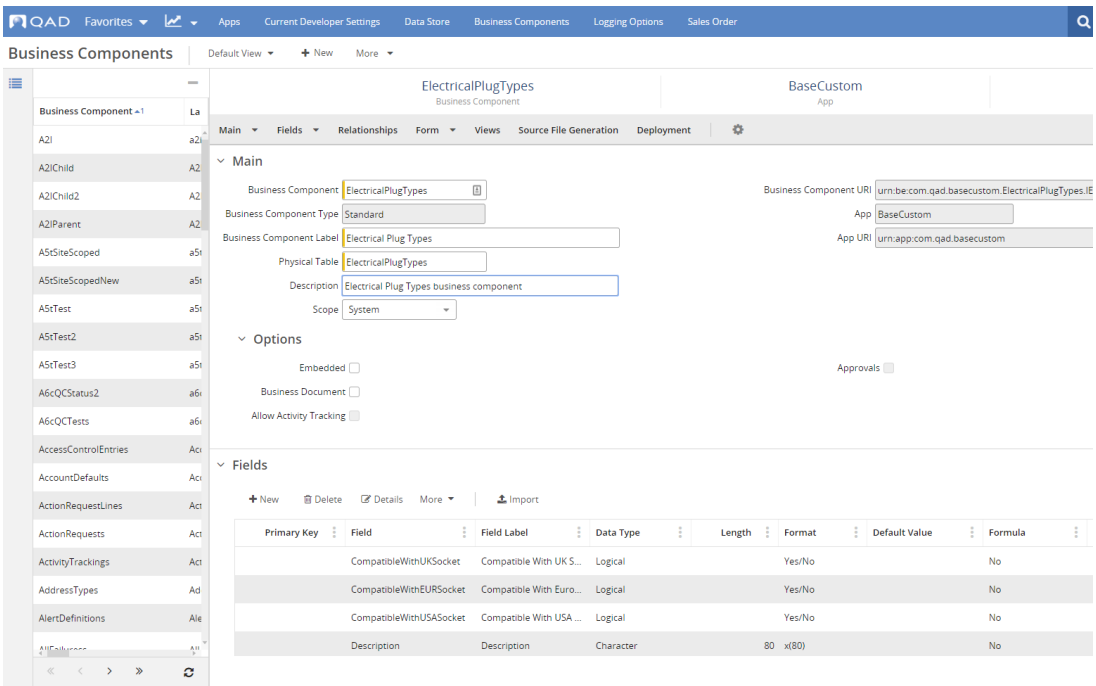
Create ElectricalPlugTypes Business Component

The ElectricalPlugTypes business component represents the different plugs that exist in the world. Every type is for a certain plug type that is used in a certain area in the world.

Do the following to create the business component:

1. Navigate to **Business Components** from the menu search and click the **New** toolbar button.
2. Enter the following values on the screen:
    - Business Component: The name of the BC: ElectricalPlugTypes
    - Business Component Label: This is the label that is used for the BC to display it in other screens (e.g., in browses/lookups): Electrical Plug Types
    - Physical Table: The name of the table that will be created in the data base: ElectricalPlugTypes
    - Description: Description of the function of the BC: Electrical Plug Types business component
    - Scope: The scope the BC will run in is System in this case because these plugin types have no relation with a domain or anything like that.
    - Fields:
        - PlugType: A code identifying the plug (Label Plug Type, Type Character, length 1, display format x(1)). See https://www.worldstandards.eu/electricity/plugs-and-sockets/.
        - Description: Description of the plug type
        - CompatibleWithUSASocket: Determines whether the plug is compatible with sockets in the USA (Label Compatible with USA socket, Type logical)
        - CompatibleWithEURSocket: Determines whether the plug is compatible with sockets in Europe (Label Compatible with European socket, Type logical)
        - CompatibleWithUKSocket: Determines whether the plug is compatible with sockets in the UK (Label Compatible with UK socket, Type logical)
3. Click **Save**.

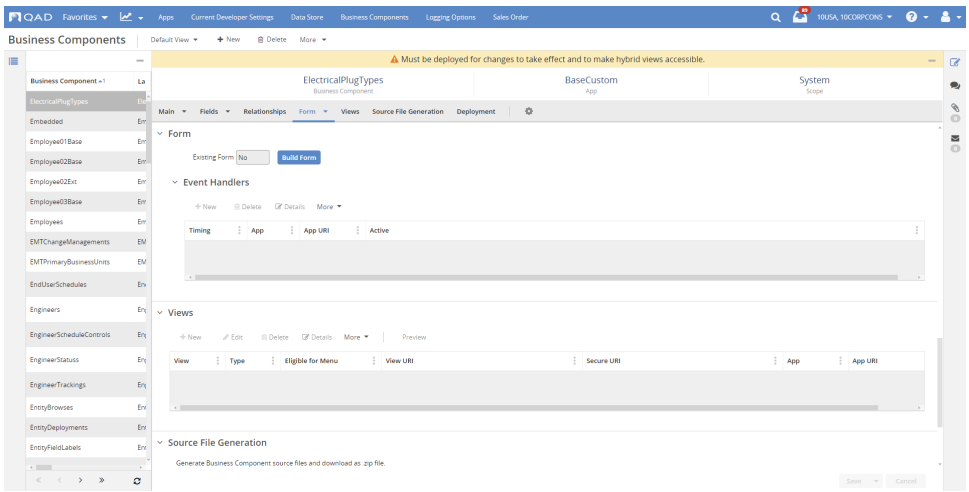
This is what the BC should look like:



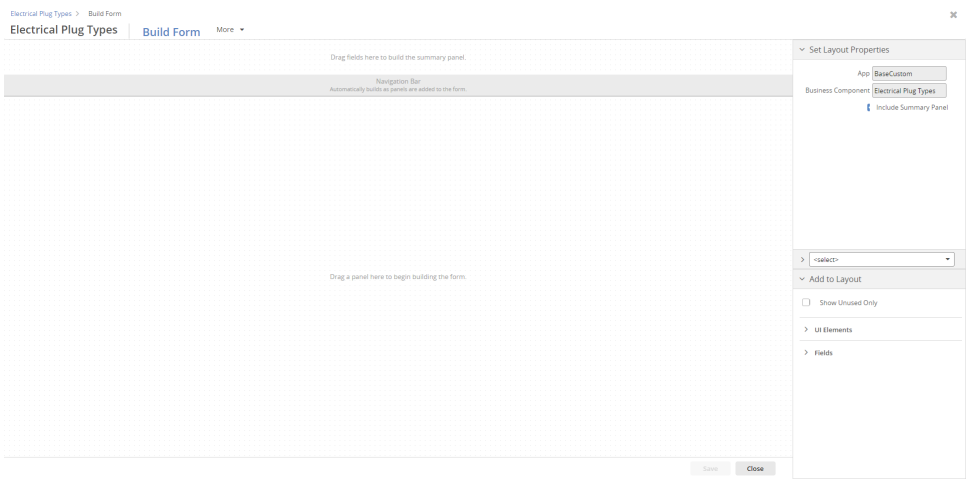
Create ElectricalPlugTypes View

The following step is to create a view form for the ElectricalPlugTypes BC we just created. In order to do so, do the following steps:

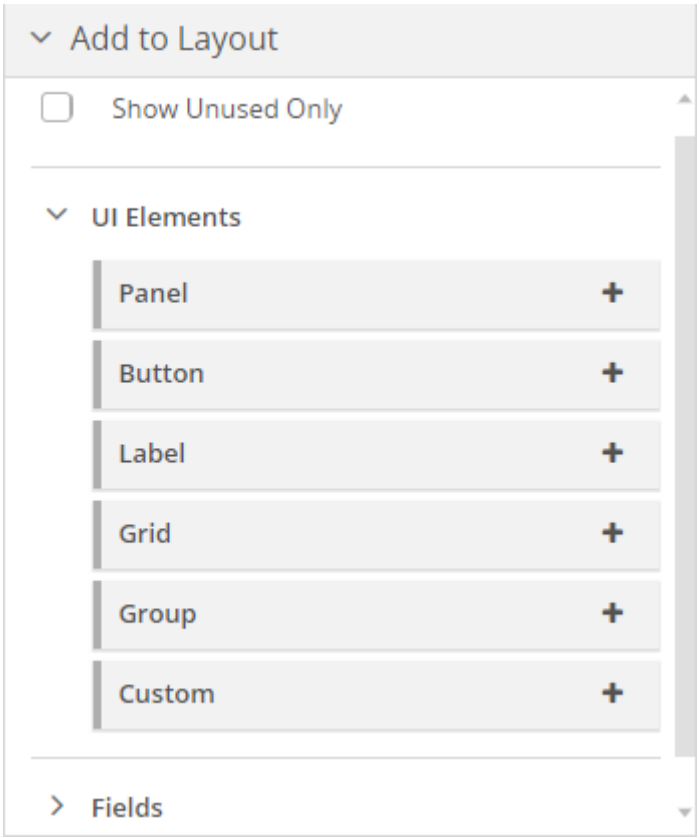
1. Open the **ElectricalPlugTypes** BC.
2. Click **Form** to go to the **Form** panel.



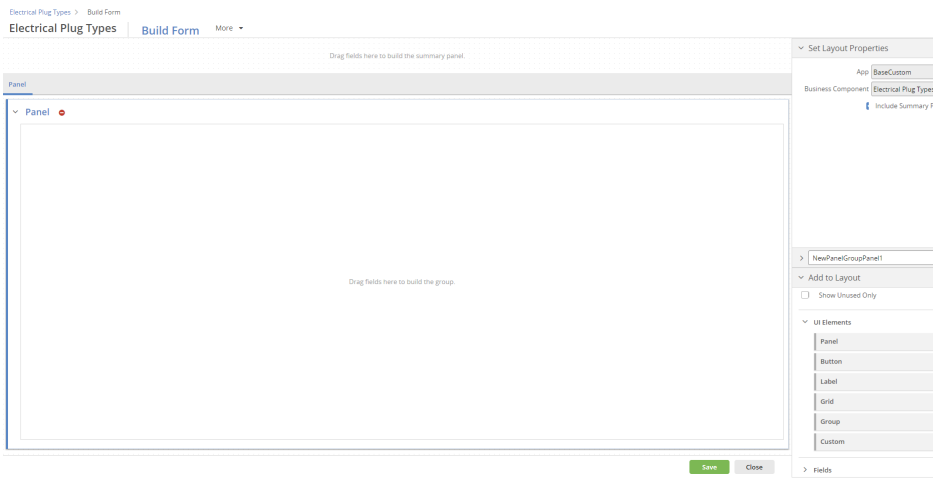
3. Click the **Build Form** button. This will open the form builder.



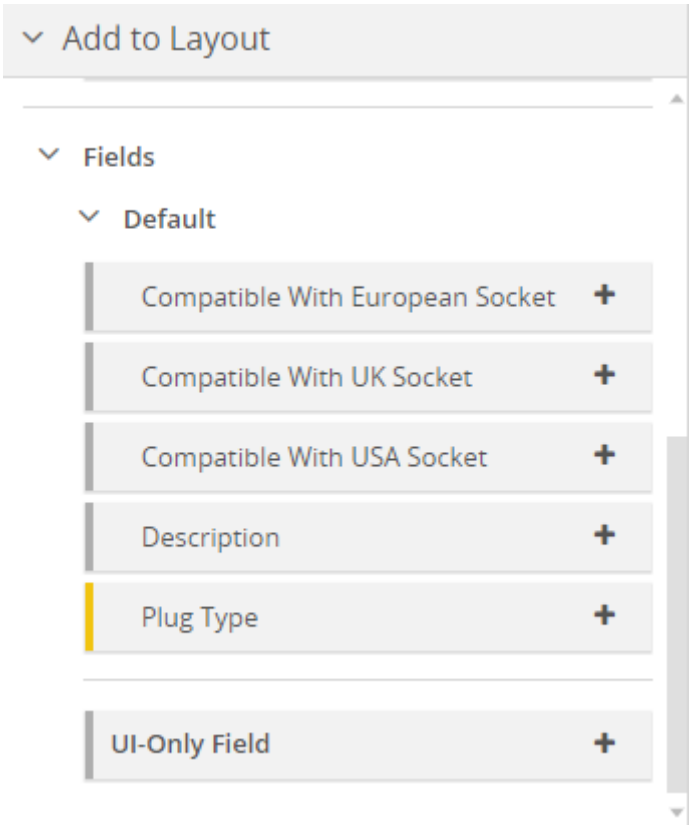
4. We now have an empty view that we can start dragging and dropping UI elements to. Follow these steps to add the elements:
    a. Click the **Add to Layout** menu at the right.



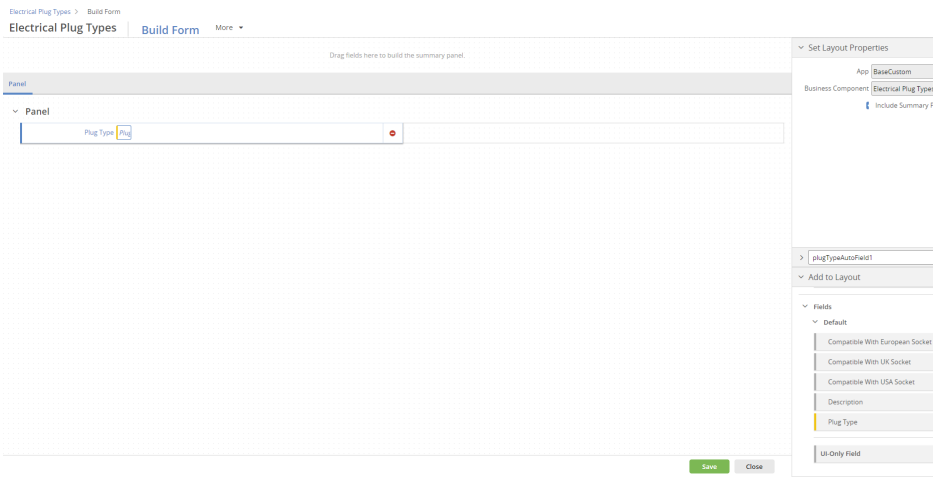
b. Now select the **Panel** box and drag and drop it on the empty view.



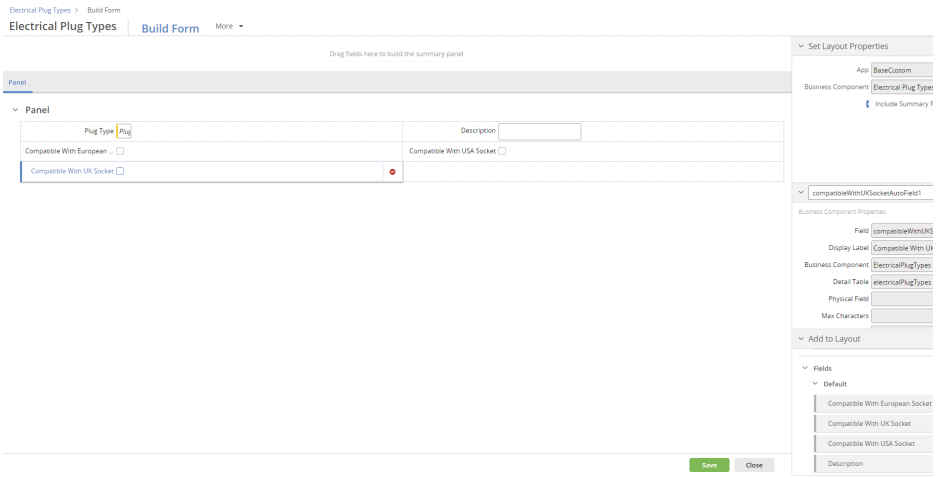
c. The next step is to drag and drop fields on the panel. Click the **Fields** and **Default** menu items at the right bottom to open them.



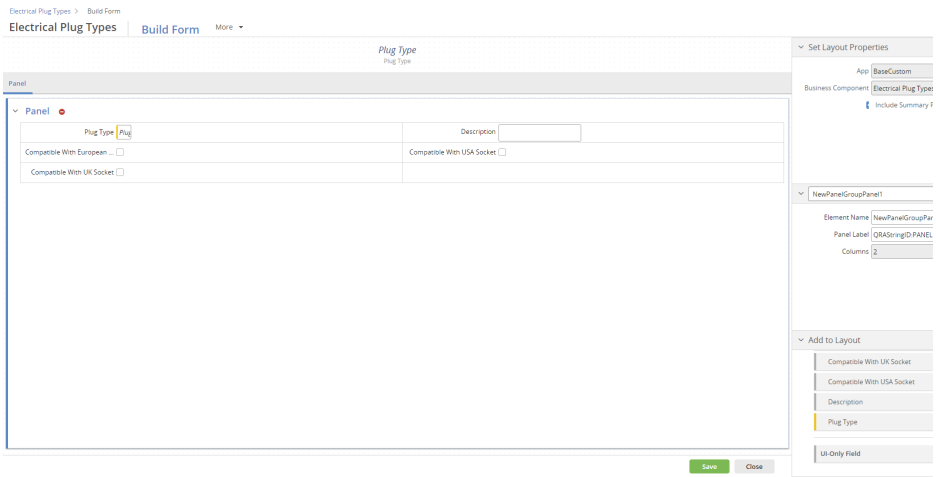
d. Now select the **Plug Type** field and drop it on the panel.



e. Do the same for **Description** and the other fields so that the screen looks like this.



f. The next step is to create a summary panel. This can be done by dropping **Plug Type** on the summary panel at the top of the screen.



g. Now our view form is ready. Click **Save**, and then **Close**.

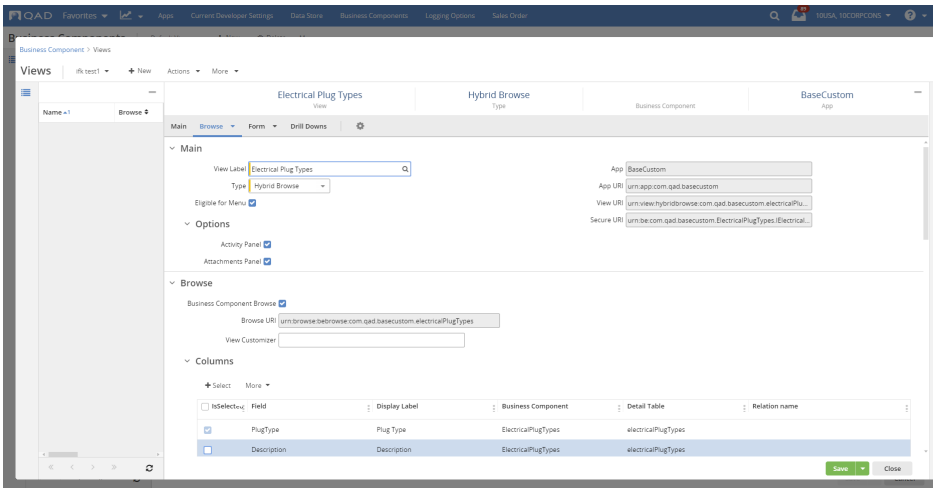
5. After saving the view form, we are back on the BC screen ready to create a Hybrid view.

Create ElectricalPlugTypes Browse

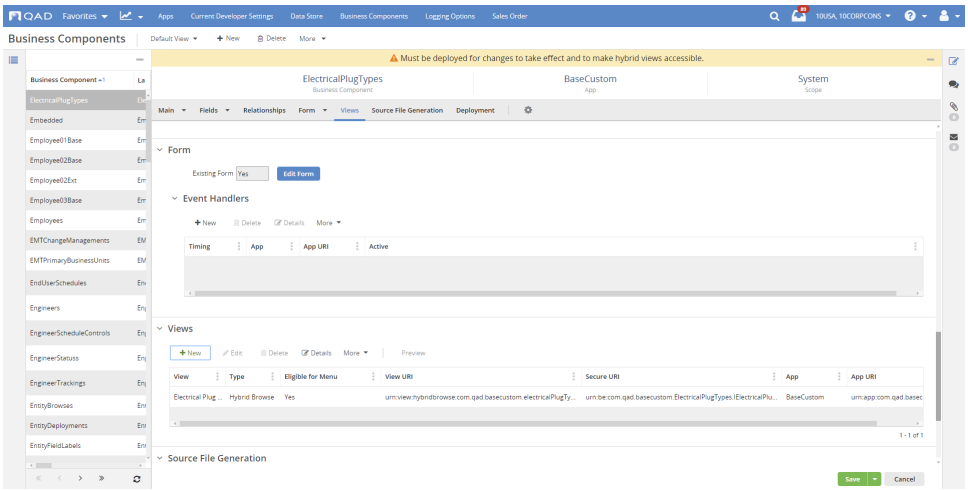
In order to see the ElectricalPlugTypes BC in the menu and open a browse on it, we need to create a BC browse. This can be done by following these steps:

1. On the ElectricalPlugTypes screen, click the **Views** panel, and then click the **New** button. This will bring up a details screen where you can create the browse. On that screen, enter the following:

    - View Label: Electrical Plug Types, this is the label the hybrid view will have on the menu.
    - In the **Browse** columns grid, deselect the **Description** column. This column is too big to show in a browse, this way we remove it from the browse view.
    - Now the screen should look as follows.



2. Click **Save** to save this browse, and then **Close**.
3. After saving the new Hybrid view, the view builder UI looks as follows.

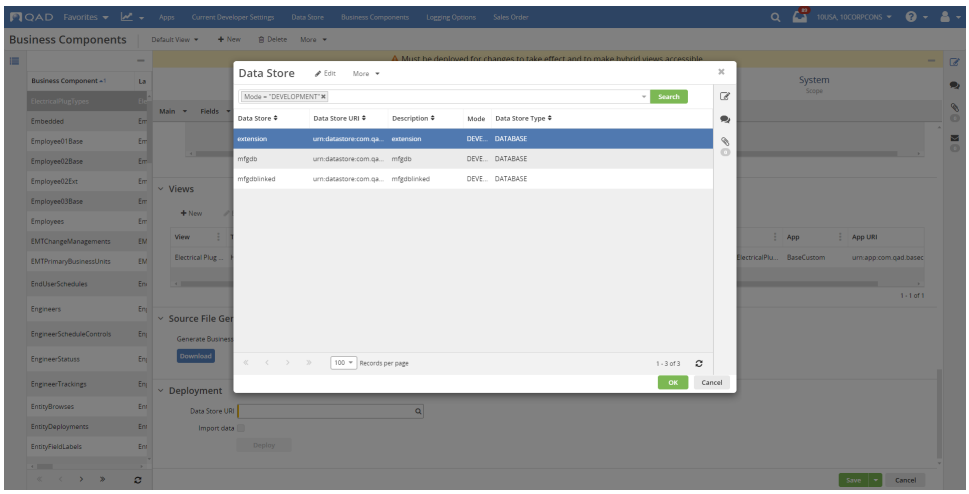


4. Now click **Save** again to save everything.

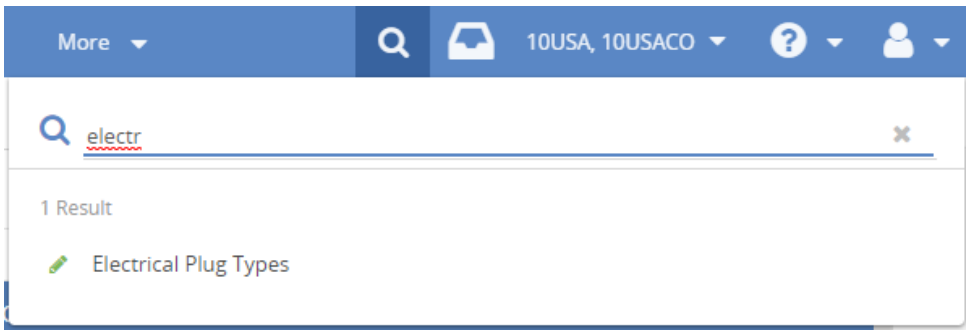
Deploy ElectricalPlugTypes Business Component

The last step for the ElectricalPlugTypes BC is to deploy it so that it becomes active in the application. In order to do this, follow these steps:

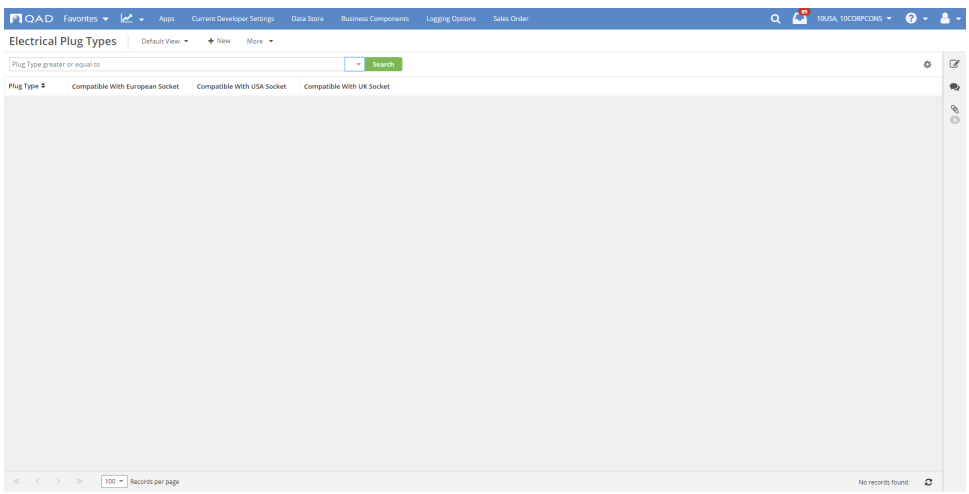
1. Open the Business Components Maintenance screen, search for the **ElectricalPlugTypes** business component, and then click the **Edit** button.
2. Scroll to the bottom of the screen and click the lookup button on the **DataStore URI** field. This will open a lookup with the available data stores. Select your developer data store and click **OK**.



3. Click the **Deploy** button to deploy the BC and make it active.
4. Now that our BC is active, we can search for it in the menu.



5. And open it.



Create ItemElectricalPlugTypes Components

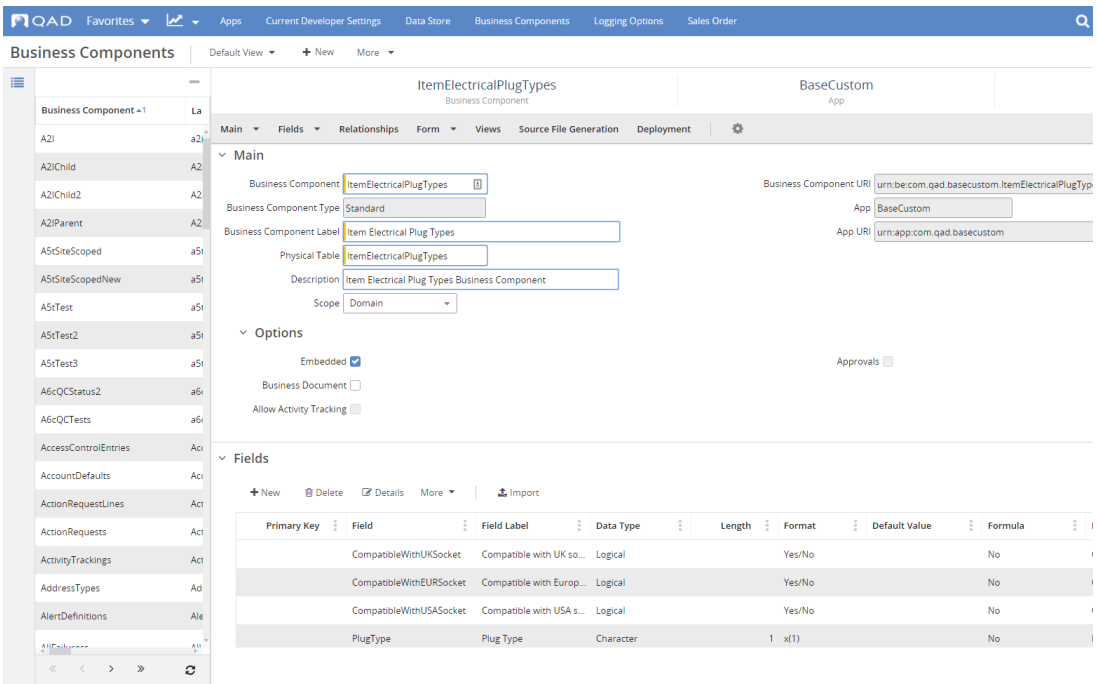
This section explains all the steps to create an embedded BC that is related to the ElectricalPlugTypes component:

Create ItemElectricalPlugTypes Embedded Business Component

In order to extend Item maintenance with fields for the two recycle taxes, we create the ItemElectricalPlugTypes embedded BC. This BC will bring ElectricalPlugTypes together with item. Follow these steps to create the Business Component:

1. Open **Business Components** and click **New**.
2. Enter the following values on the screen:
    - Business Component: The name of the BC: ItemElectricalPlugTypes
    - Business Component Label: Item Electrical Plug Types
    - Physical Table: The name of the database table: ItemElectricalPlugTypes
    - Description: Item Electrical Plug Types Business Component
    - Scope: Domain because item runs in domain context, the extension needs to do that too.
    - Embedded: Select
    - Fields (Name is max 32 characters):
        - DomainCode: Because we run in domain context, we need this field (label DomainCode, Type character, length 8, format x(8), key field 1)
        - ItemCode: This is the field that links back to the item (label Item Code, Type character, length 18, format x(18), key field 2)
        - PlugType: Plug type (label Plug Type, Type character, length 1, format x(1))
        - CompatibleWithUSASocket: Determines whether the plug is compatible with sockets in the USA (Label Compatible with USA socket, Type logical)
        - CompatibleWithEURSocket: Determines whether the plug is compatible with sockets in Europe (Label Compatible with European socket, Type logical)
        - CompatibleWithUKSocket: Determines whether the plug is compatible with sockets in the UK (Label Compatible with UK socket, Type logical)
3. Click **Save**.

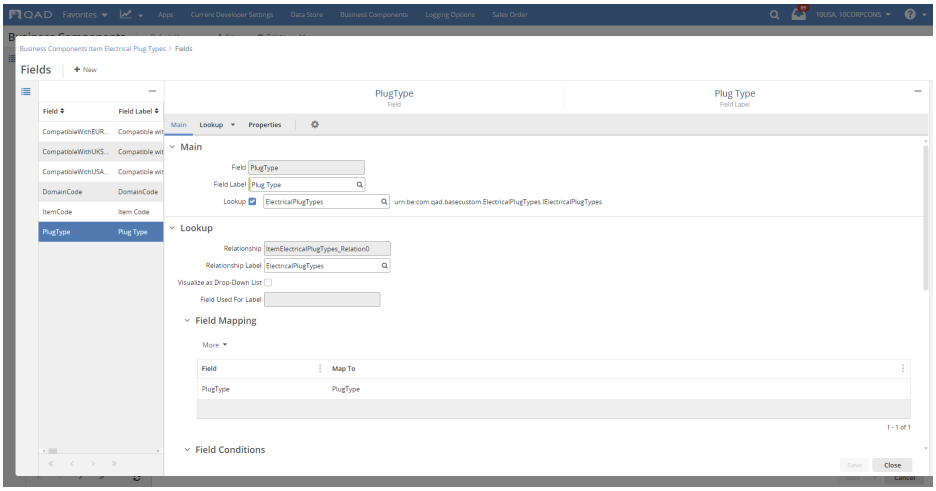
This is how the Business Component should look like:



Create ItemElectricalPlugTypes lookup relations

The ItemElectricalPlugTypes extension BC displays Electrical Plug Types. For that reason, we need to add a 1-N relation to the ElectricalPlugTypes BC we created earlier. This will give us the advantage of having automatically lookup buttons on these fields on the screen. Follow these steps to add the relations:

1. Open **Business Components**, select the **ItemElectricalPlugTypes** BC, and then click **Edit**.
2. Go to the **Fields** panel, select the **PlugType** field, and then click the **Details** button. A new popup screen appears. On this screen, enter the following values:
    - Lookup: Select the checkbox and in the lookup select the Electrical Plug Types BC

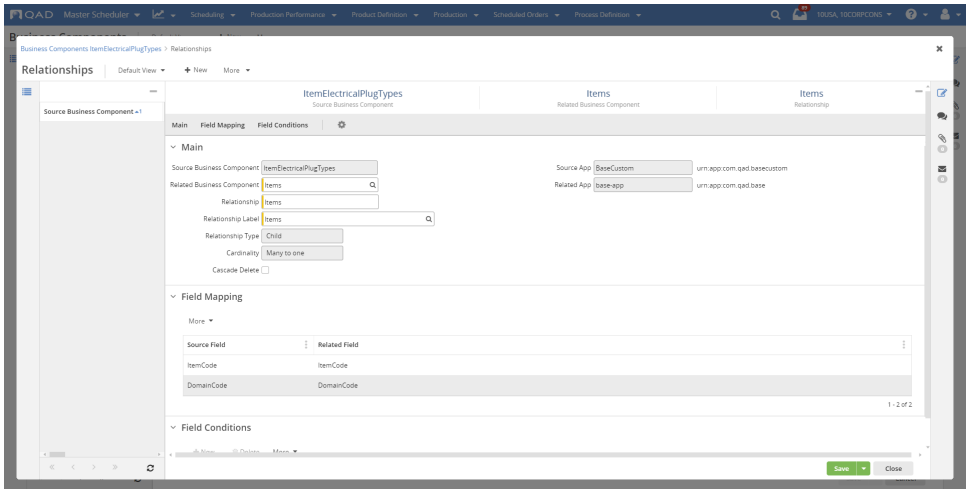


3. Click **Save**, and then click **Close**. This will bring you back to the Business Components screen.
4. Click the **Save** button again to finish this task.

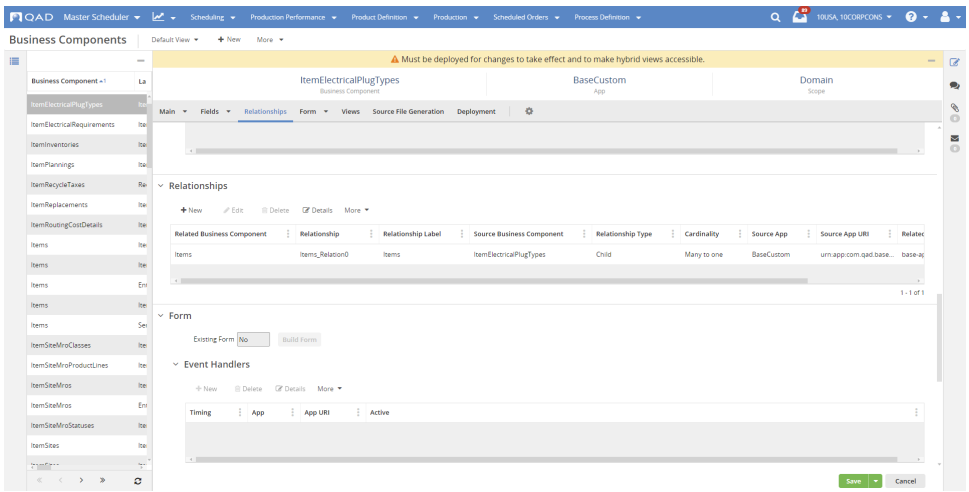
Create ItemElectricalPlugTypes N-1 extension relation

In order to extend item with this BC, we need to add a N-1 relation to the item. Follow these steps to do that:

1. In **Business Components**, select **ItemElectricalPlugTypes**, and then click **Edit**.
2. Go to the **Relationships** panel and click the **New** button.
3. Click the Related Business Component's lookup, select the **Items** business component, and then click **OK**. The rest of the fields automatically defaults to the correct value.
4. Select the correct mapping fields:



5. Click **Save**, and then click **Close**.
6. After that, your BC screen should look like this:

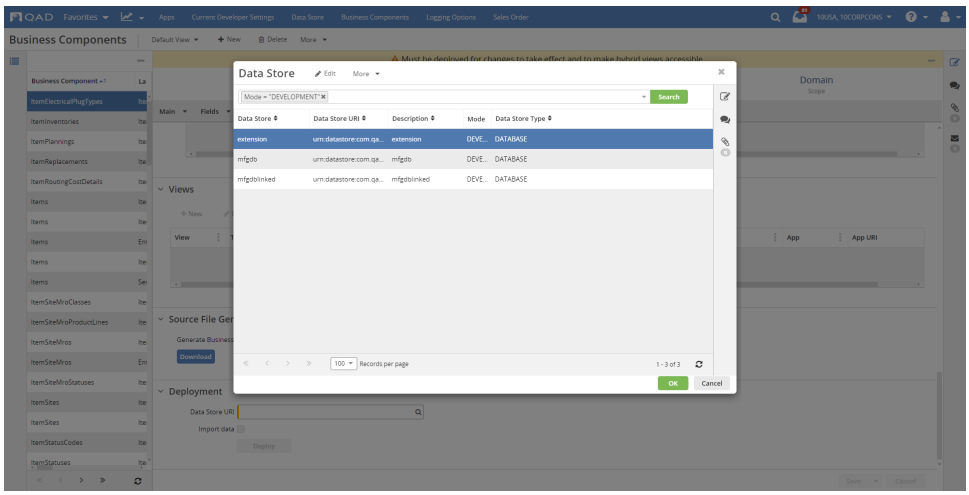


7. Click **Save** again to finish this task.

Deploy ItemElectricalPlugTypes Business Component

The last step to make this Extension BC active is to deploy it. Follow these steps to do so:

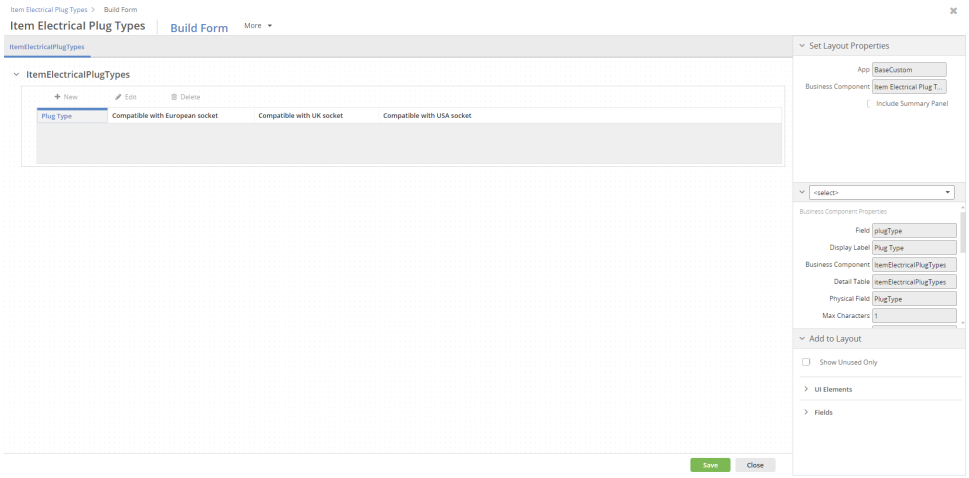
1. Open **Business Components**, select the **Item Electrical Plug Types** component, and then click **Edit**.
2. Go to the bottom of the screen, click the **Data Store URI** lookup, and then select your development data base:



3. Click **OK**.
4. Click the **Deploy** button to deploy.
5. Now when opening the Item BC, you should see a grid where you can add plug types.

Modify View to change grid column order

1. Open **Business Components**, select the **Item Electrical Plug Types** component, and then click **Edit**.
2. Go to the Form Builder and click the **Edit Form** button.
3. Click the **Plug Type** column and drag it to the front of the grid.



4. Click **Save**, and then **Close**.

Create ItemElectricalRequirements Components

This section explains all the steps to create an embedded BC that is related to the ElectricalRequirements component:

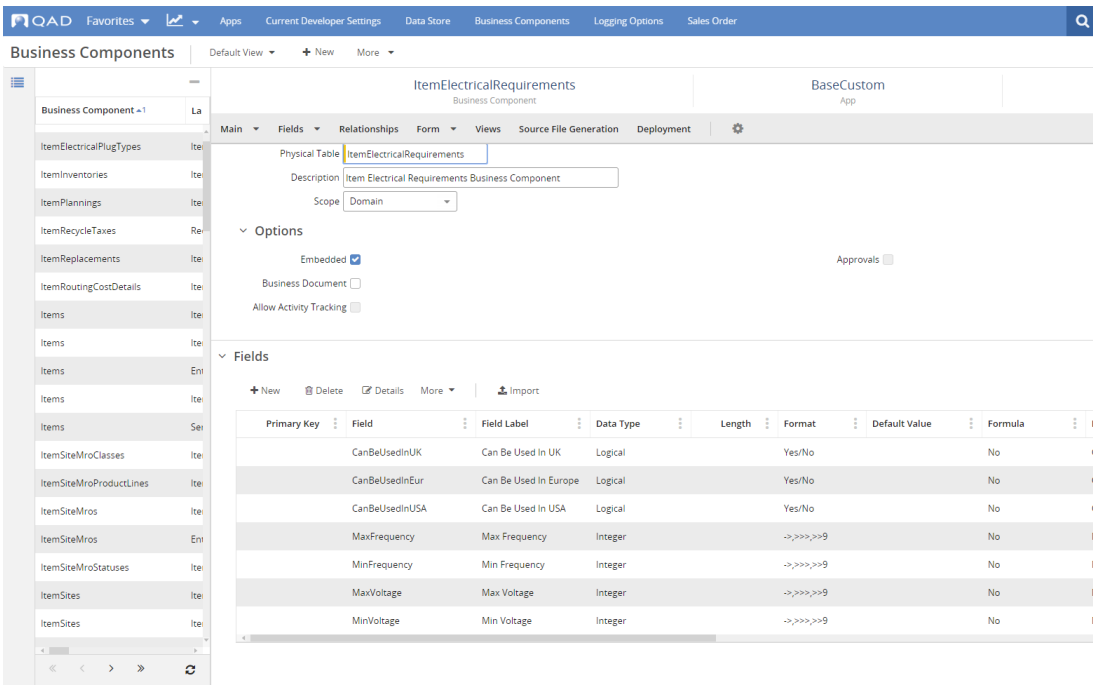
- Create ItemElectricalRequirements Embedded Business Component
- Create ItemElectricalRequirements 1-1 extension relation
- Deploy ItemElectricalRequirements Business Component

Create ItemElectricalRequirements Embedded Business Component

In order to extend Item maintenance with fields for the electrical requirements info, we create the ItemElectricalRequirements embedded BC. This BC will bring ItemElectricalRequirements together with item. Follow these steps to create the Business Component:

1. Open **Business Components** and click **New**.
2. Enter the following values on the screen:
    - Business Component: The name of the BC: ItemElectricalRequirements
    - Business Component Label: Item Electrical Requirements
    - Physical Table: name of the database table: ItemElectricalRequirements
    - Description: Item Electrical Requirements Business Component
    - Scope: Domain because item runs in domain context, the extension needs to do that too.
    - Embedded: Select
    - Fields (Name is max 32 characters):
        - DomainCode: Because we run in domain context, we need this field (label DomainCode, Type character, length 8, format x(8), key field 1)
        - ItemCode: This is the field that links back to the item (label Item Code, Type character, length 18, format x(18), key field 2)
        - MinVoltage: This is the minimum voltage the item can work with (label Min Voltage, integer)
        - MaxVoltage: This is the maximum voltage the item can work with (label Max Voltage, integer)
        - MinFrequency: This is the minimum frequency the item can work with (label Min Frequency, integer)
        - MaxFrequency: This is the maximum frequency the item can work with (label Max Frequency, integer)
        - CanBeUsedInUSA: This field indicates that the item can be used in the USA (label Can Be Used In USA, logical, read-only)
        - CanBeUsedInEur: This field indicates that the item can be used in Europe (label Can Be Used In Europe, logical, read-only)
        - CanBeUsedInUK: This field indicates that the item can be used in the UK (label Can Be Used In UK, logical, read-only)
3. Click **Save**.

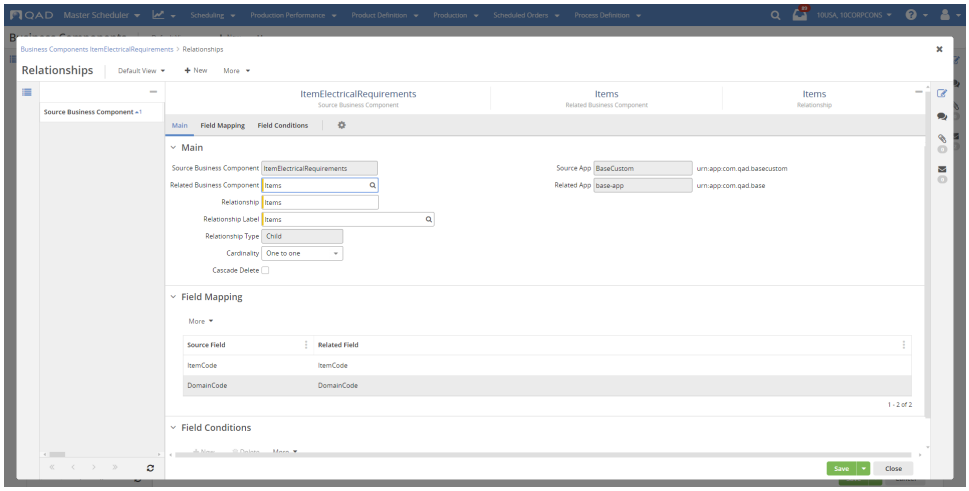
This is how the Business Component should look like:



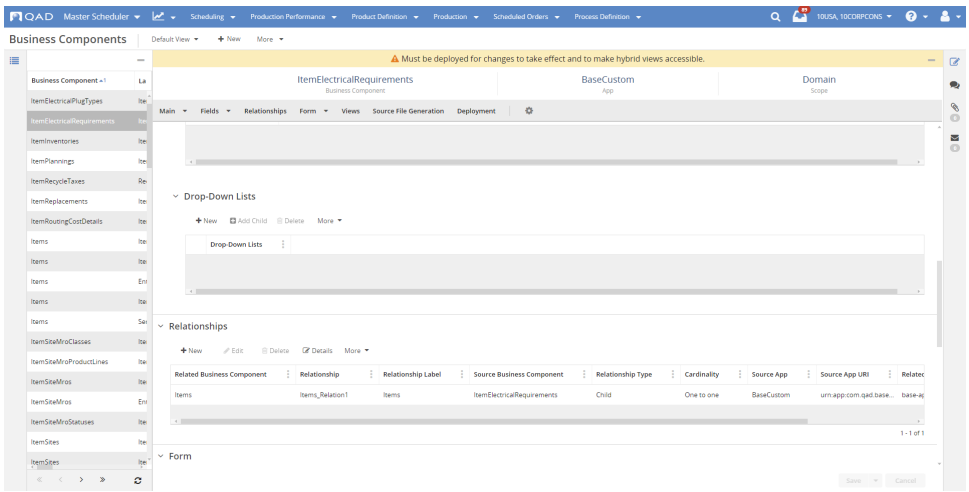
Create ItemElectricalRequirements 1-1 extension relation

In order to extend item with this BC, we need to add a 1-1 relation to the item. Follow these steps to do that:

1. In **Business Components**, select **ItemElectricalRequirements**, and then click **Edit**.
2. Go to the **Relationships** panel and click the **New** button.
3. Click the Related Business Component's lookup, select the **Items** business component, and then click **OK**. The rest of the fields automatically defaults to the correct value.
4. Select the correct mapping fields:



5. Click **Save**, and then click **Close**.
6. After that, your BC screen should look like this:

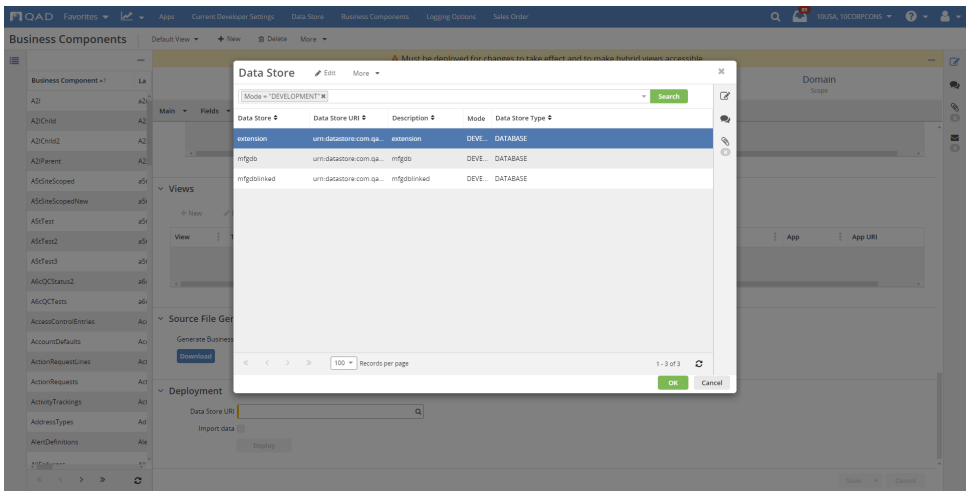


7. Click **Save** again to finish this task.

Deploy ItemElectricalRequirements Business Component

The last step to make this Extension BC active is to deploy it. Follow these steps to do so:

1. Open **Business Components**, select the **Item Electrical Requirements** component, and then click **Edit**.
2. Go to the bottom of the screen, click the **Data Store URI** lookup, and then select your development data base:

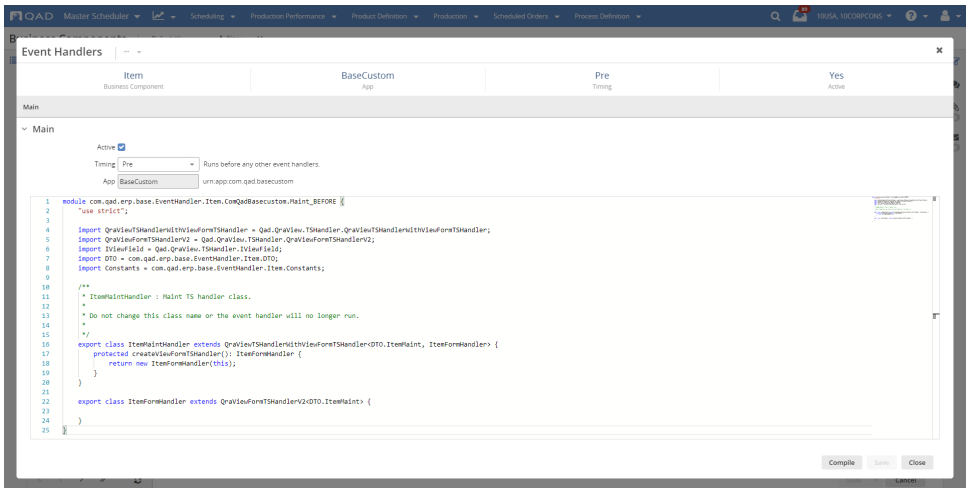


3. Click **OK**.
4. Click the **Deploy** button to deploy.
5. Now when opening the Item BC, you should see an extra panel with electrical requirements info.

Add event handler to retrieve electrical plug info

At this point, we have item extended with the electrical plug info and with electrical requirements. We also need to retrieve extra values from the plug type data and store it in the grid from the ItemsElectricalPlugTypes extension. This can be done by doing the following steps:

1. Open **Business Components**, select **Items**, and then click **Edit**.
2. Go to the **Form Builder** > **Event Handlers** and open the event handler code.



3. At the bottom of the code, add a class for the grid TS handler:

```
    export class ItemElectricalPlugInfoGrid extends Qad.QraView.TSHandler.
ViewGridTSHandlerV2<DTO.ItemMaint,DTO.Items_Relation1,DTO.ItemElectricalPlugTypes | kendo.
data.ObservableObject> {
    }
```

4. And in the main handler, add the methods init and createViewGridTSHandler to assign an instance of the class we just created:

```
    /**
     * ItemMaintHandler : Maint TS handler class.
     *
     * Do not change this class name or the event handler will no longer run.
     *
     */
    export class ItemMaintHandler extends QraViewTSHandlerWithViewFormTSHandler<DTO.
ItemMaint, ItemFormHandler> {
        protected init() {
            this.ViewGridsToHandleList=["itemElectricalPlugTypesOneToManyAutoGrid"];
        }
        protected createViewFormTSHandler(): ItemFormHandler {
            return new ItemFormHandler(this);
        }
        protected createViewGridTSHandler(viewGrid: Qad.QraView.TSHandler.IViewGrid<any,
any, kendo.data.ObservableObject>): Qad.QraView.TSHandler.ViewGridTSHandler<any, any,
any, any> {
            if (viewGrid.GridID==="itemElectricalPlugTypesOneToManyAutoGrid") {
                return new ItemElectricalPlugTypesOneToManyAutoGridHandler(viewGrid,this);
            }

        }

    }
```

5. On the grid handler, add a method to make the compatibility fields read-only:

```
        public onAutoGridAfterInit(eventData: Communication.EventData.AutoGrid.
AfterInitEventData): void {
```

```
            this.ViewGrid.setGridColumnDisabled("compatibleWithUSASocket",true);
            this.ViewGrid.setGridColumnDisabled("compatibleWithEURSocket",true);
            this.ViewGrid.setGridColumnDisabled("compatibleWithUKSocket",true);
        }
```

6. Then, on the grid handler, add a method to retrieve the plug type info and a method to handle server errors:

```
         /**
         * @function updatePlugTypeInfo: retrieve plug info and update grid. When the
data is not found, an error will be displayed in the error viewer.
         *
         * @param domainCode : domain code
         * @param plugType: plugType code
         */
        private updatePlugTypeInfo(domainCode: string, gridDataRow:DTO.
ItemElectricalPlugTypes & kendo.data.ObservableObject,plugType: string): void {
            let targetUrl: string;
            let finalUrl: string;
            targetUrl="api/qracore/be/urn:be:com.extensions.basecustom.
ElectricalPlugTypes.IElectricalPlugTypes?domainCode={domainCode}&plugType={plugType}";
            //replace placeholders in the url with the domain code and tax code
            finalUrl=bindTemplateData(targetUrl, {domainCode: domainCode,plugType:
plugType});

            if (finalUrl) {
                this.ViewController.doHttpGet(
                    finalUrl,
                    (response: Qad.Common.DTO.DataResult) => {
                        let data = response.data;
                        if (data) {
                            if (data.electricalPlugTypes && data.electricalPlugTypes.
length>0) {
                                this.ViewGrid.setRowFieldValue(gridDataRow,"
compatibleWithUSASocket",data.electricalPlugTypes[0].compatibleWithUSASocket?true:false,
false);
                                this.ViewGrid.setRowFieldValue(gridDataRow,"
compatibleWithEURSocket",data.electricalPlugTypes[0].compatibleWithEURSocket?true:false,
false);
                                this.ViewGrid.setRowFieldValue(gridDataRow,"
compatibleWithUKSocket",data.electricalPlugTypes[0].compatibleWithUKSocket?true:false,
false);
                            }
                            else {
                                this.handleServerError(undefined,"Error retrieving plug
type info with code : " + plugType,"not found");
                            }
                        }
                        else {
                            this.handleServerError(undefined,"Error retrieving plug type
info with code : " + plugType,"not found");
                        }


                    },
                    (data, status) => this.handleServerError(data,undefined,undefined),
                    null,
                    null,
                    true,
                    false,
                    true,
                    false,
                    false
                );
            }
        }
        /**
         * @function handleServerError : display error information in error viewer
         *
         * @param submitresults : array of SubmitResult objects
         * @param message : message to show
         * @param status : status to show in the message
         */
        private handleServerError(submitresults: Qad.Common.DTO.SubmitResult , message:
string, status: string) {
            debugger;
            let serverErrors: Qad.Common.DTO.Error[] = [];
            if (submitresults.errors && submitresults.errors.length && submitresults.
```

```
errors.length>0) {
                serverErrors = submitresults.errors;
            } else {
                serverErrors.push(new Qad.Common.DTO.Error());
                serverErrors[0].message = "" + message + " " + status;
                serverErrors[0].severity = 1;
            }
            this.ViewController.ErrorGroupPanel.clearErrorGrid();
            this.ViewController.ErrorGroupPanel.addErrorsToErrorGrid(serverErrors);
            this.ViewController.ErrorGroupPanel.showErrorGrid();
        }
```

7. Then add a field change event handler:

```
        public onAutoGridFieldChangeEvent(eventData: Communication.EventData.AutoGrid.
FieldChangeEventData<kendo.data.ObservableObject | (DTO.ItemElectricalPlugTypes & kendo.
data.ObservableObject)>, processEvent: (processIt?: boolean) => void): void {
            if (eventData.fieldName==="plugType") {
                this.updatePlugTypeInfo(this.NgData.items[0].domainCode,<DTO.
ItemElectricalPlugTypes & kendo.data.ObservableObject>eventData.dataRow, eventData.
fieldValue);
            }
        }
```

Add event handler to update electrical requirements panel

The Electrical requirements panel has a few flags that are calculated depending on the voltage/frequency and available plugs that determines if the item can be used in the USA, Europe, and/or the UK.

These flags will be calculated in an event handler as follows:

1. Open **Business Components**, select **Items**, and then click **Edit**.
2. Go to the **Form Builder** > **Event Handlers** and open the event handler code.
3. On the grid handler, add a function to calculate the electrical usability:

```
        /**
         * @function checkItemElectricalUsability : Check if the item is usable in USA,
Europe, and or UK
         */
        public checkItemElectricalUsability() {
            //if there is no data, there is nothing to check
            if (!(this.NgData && this.NgData.items && this.NgData.items.length>0 && this.
NgData.Items_Relation2 && this.NgData.Items_Relation2.length>0)) {
                return;
            }
            let hasUSACompatiblePlug=false;
            let hasEURCompatiblePlug=false;
            let hasUKCompatiblePlug=false;
            let allFieldsHaveValidValue=this.NgData.Items_Relation2[0].minVoltage && this.
NgData.Items_Relation2[0].maxVoltage &&
                this.NgData.Items_Relation2[0].minFrequency && this.NgData.Items_Relation2
[0].maxFrequency;
            let hasCompatibleVoltageAndFrequencyForUSA=allFieldsHaveValidValue && this.
NgData.Items_Relation2[0].minVoltage<=110 && this.NgData.Items_Relation2[0].
maxVoltage>=110 &&
                this.NgData.Items_Relation2[0].minFrequency<=60 && this.NgData.
Items_Relation2[0].maxFrequency>=60;
            let hasCompatibleVoltageAndFrequencyForEurope=allFieldsHaveValidValue && this.
NgData.Items_Relation2[0].minVoltage<=230 && this.NgData.Items_Relation2[0].
maxVoltage>=230 &&
                this.NgData.Items_Relation2[0].minFrequency<=50 && this.NgData.
Items_Relation2[0].maxFrequency>=50;
            let hasCompatibleVoltageAndFrequencyForUK=allFieldsHaveValidValue && this.
NgData.Items_Relation2[0].minVoltage<=230 && this.NgData.Items_Relation2[0].
maxVoltage>=230 &&
                this.NgData.Items_Relation2[0].minFrequency<=50 && this.NgData.
Items_Relation2[0].maxFrequency>=50;


            //check plug types
            let gridData=<(DTO.ItemElectricalPlugTypes & kendo.data.ObservableObject)[]
>this.ViewGrid.Data;
            gridData.forEach((value: DTO.ItemElectricalPlugTypes & kendo.data.
ObservableObject, index: number, array: (DTO.ItemElectricalPlugTypes & kendo.data.
ObservableObject)[]) => {
                if (value.compatibleWithUSASocket) {
                    hasUSACompatiblePlug=true;
                }
                if (value.compatibleWithEURSocket) {
                    hasEURCompatiblePlug=true;
                }
                if (value.compatibleWithUKSocket) {
                    hasUKCompatiblePlug=true;
                }
            }, this);
            this.NgData.Items_Relation2[0].
canBeUsedInUSA=hasCompatibleVoltageAndFrequencyForUSA && hasUSACompatiblePlug;
            this.NgData.Items_Relation2[0].
canBeUsedInEur=hasCompatibleVoltageAndFrequencyForEurope && hasEURCompatiblePlug;
            this.NgData.Items_Relation2[0].
canBeUsedInUK=hasCompatibleVoltageAndFrequencyForUK && hasUKCompatiblePlug;

        }
```

4. Now that we have a function that checks the electrical compatibility, we need to call it on the following UI events:
    - When the plug type in the grid changes and the plug type info is updated with data from the back-end
    - When one of the voltage or frequency fields of the electrical requirements extension changes.

5. For the plug type in the grid, we need to add one line (this.checkItemElectricalUsability();) to the updatePlugTypeInfo method:

```
        /**
         * @function updatePlugTypeInfo: retrieve plug info and update grid. When the
data is not found, an error will be displayed in the error viewer.
         *
         * @param domainCode : domain code
         * @param plugType: plugType code
         */
        private updatePlugTypeInfo(domainCode: string, gridDataRow:DTO.
ItemElectricalPlugTypes & kendo.data.ObservableObject,plugType: string): void {
            let targetUrl: string;
            let finalUrl: string;
            targetUrl="api/qracore/be/urn:be:com.extensions.basecustom.
ElectricalPlugTypes.IElectricalPlugTypes?domainCode={domainCode}&plugType={plugType}";
            //replace placeholders in the url with the domain code and tax code
            finalUrl=bindTemplateData(targetUrl, {domainCode: domainCode,plugType:
plugType});

            if (finalUrl) {
                this.ViewController.doHttpGet(
                    finalUrl,
                    (response: Qad.Common.DTO.DataResult) => {
                        let data = response.data;
                        if (data) {
                            if (data.electricalPlugTypes && data.electricalPlugTypes.
length>0) {
                                this.ViewGrid.setRowFieldValue(gridDataRow,"
compatibleWithUSASocket",data.electricalPlugTypes[0].compatibleWithUSASocket?true:false,
false);
                                this.ViewGrid.setRowFieldValue(gridDataRow,"
compatibleWithEURSocket",data.electricalPlugTypes[0].compatibleWithEURSocket?true:false,
false);
                                this.ViewGrid.setRowFieldValue(gridDataRow,"
compatibleWithUKSocket",data.electricalPlugTypes[0].compatibleWithUKSocket?true:false,
false);
                                this.checkItemElectricalUsability();


                            }
                            else {
                                this.handleServerError(undefined,"Error retrieving plug
type info with code : " + plugType,"not found");
                            }
                        }
                        else {
                            this.handleServerError(undefined,"Error retrieving plug type
info with code : " + plugType,"not found");
                        }


                    },
                    (data, status) => this.handleServerError(data,undefined,undefined),
                    null,
                    null,
                    true,
                    false,
                    true,
                    false,
                    false
                );
            }
        }
```

6. For the voltage and frequency fields, it is a little harder. Because these fields change events fire in the form handler and not in the grid handler, we need to call the grid handler from the form handler. In order to do so, we need to:
    - Have a reference to the grid handler in the form handler. However, we cannot rely on the order of objects being created, that's why we keep a reference to the grid handler in the main handler, and we pass a reference to that main handler to the form handler:

```
    /**
     * ItemMaintHandler : Maint TS handler class.
     *
     * Do not change this class name or the event handler will no longer run.
     *
```

```
     */
    export class ItemMaintHandler extends
QraViewTSHandlerWithViewFormTSHandler<DTO.ItemMaint, ItemFormHandler> {
        public itemElectricalPlugTypesOneToManyAutoGridHandler:
ItemElectricalPlugTypesOneToManyAutoGridHandler;


        protected init() {
            this.ViewGridsToHandleList=
["itemElectricalPlugTypesOneToManyAutoGrid"];
        }

        protected createViewFormTSHandler(): ItemFormHandler {
            let itemFormHandler=new ItemFormHandler(this);
            itemFormHandler.mainTSHandler=this;
            return itemFormHandler;
        }

        protected createViewGridTSHandler(viewGrid: Qad.QraView.TSHandler.
IViewGrid<any, any, kendo.data.ObservableObject>): Qad.QraView.TSHandler.
ViewGridTSHandler<any, any, any, any> {
            if (viewGrid.GridID==="itemElectricalPlugTypesOneToManyAutoGrid") {
                this.itemElectricalPlugTypesOneToManyAutoGridHandler=new
ItemElectricalPlugTypesOneToManyAutoGridHandler(viewGrid,this);
                return this.itemElectricalPlugTypesOneToManyAutoGridHandler;
            }

        }

    }
```

Note the modified createViewFormTSHandler and createViewGridTSHandler methods and the added itemElectricalPlugTypesOneToManyAutoGridHandler class variable.

- After adding the above code, we need to call the checkItemElectricalUsability method from the right event handlers:

```
    export class ItemFormHandler extends QraViewFormTSHandlerV2<DTO.ItemMaint> {
        public mainTSHandler:ItemMaintHandler;


        /**
         * onFieldChange event handler
         */
        public onFieldChange(viewField: IViewField<any>, eventData: Communication.
EventData.QraView.FieldChangeEventData<any>, processEvent: (processIt?: boolean)
=> void): void{
            if (viewField.Name=="itemRecycleTaxes3recupelTaxGroupCodeAutoField") {
                //check if we have data on the screen
                if (this.NgData && this.NgData.items && this.NgData.items.
length>0) {
                    this.updateRecupelTaxValue(this.NgData.items[0].domainCode,
viewField.Value);
                }
            }
            else if (viewField.Name=="
itemRecycleTaxes3bebatTaxGroupCodeAutoField") {
                //check if we have data on the screen
                if (this.NgData && this.NgData.items && this.NgData.items.
length>0) {
                    this.updateBebatTaxValue(this.NgData.items[0].domainCode,
viewField.Value);
                }
            }
            else if (viewField.Name=="
itemElectricalRequirementsmaxVoltageAutoField" || viewField.Name=="
itemElectricalRequirementsminVoltageAutoField" ||
                     viewField.Name=="
itemElectricalRequirementsmaxFrequencyAutoField" || viewField.Name=="
itemElectricalRequirementsminFrequencyAutoField") {
                this.mainTSHandler.itemElectricalPlugTypesOneToManyAutoGridHandler.
checkItemElectricalUsability();
            }

        }
```

Note the added code at the bottom of the onFieldChange event handler.

7. After this, the event handler can be compiled and saved, and you will see that the electrical requirements extension checkboxes are automatically updated when updating the voltages, frequencies, or plug type.

Example 2 - Extend the UI: Extending standard functionality with extra UI validation

In this example, we will show how to extend Customers UI with some extra UI functionality that will do the following:

- When there is a country and postal code, we will call an external system called zippopotam.us (www.zippopotam.us) to validate the postal code and retrieve some extra information.
- With the extra information, we automatically fill in the city name. Or, when there are multiple city names for the same postal code (this happens in Belgium), we change the input to a drop-down box with the possible city names.
- With the extra information we receive from zippopotam, we also show a link to google maps that will show the city location on the map.

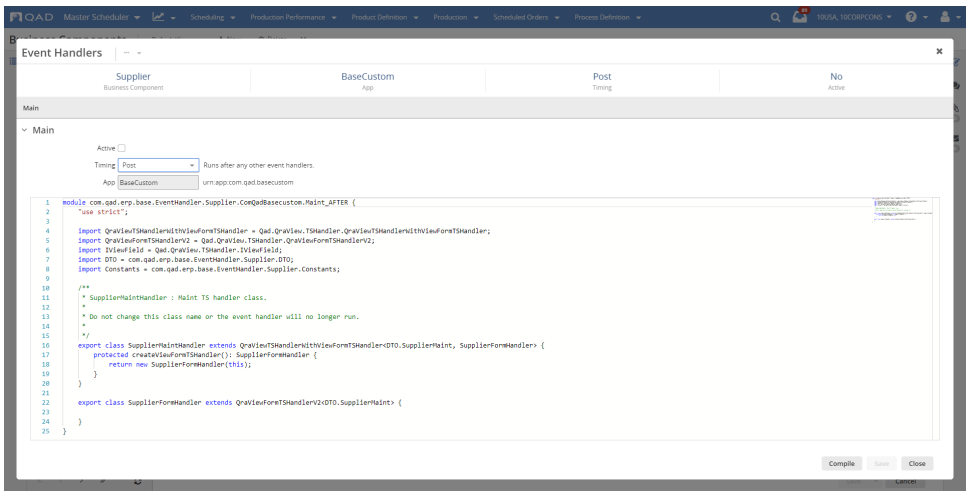
Steps to follow:

- Add an event handler to Suppliers maintenance UI
- Add method to call external system to event handler
- Add code to act on UI events and call the external system
- Add code to add link to google maps on UI
- Add code to replace city field with drop-down box
- Ref: complete event handler code for Suppliers BC

Add an event handler to Suppliers maintenance UI

For this example, we will be adding code to the UI logic. This will be done with an event handler. Follow these steps to add the event handler:

1. Open **Business Componentes**, select **Suppliers** (base app), and then click **Edit**.
2. Go to the **Form Builder** > **Event Handlers**, click the **New** button and change the timing to **Post**.



3. You are now ready to add code: Add method to call external system to event handler

Add method to call external system to event handler

In order to query zippopotam, we have to do an http call. How this call should look like is explained here: http://www.zippopotam.us/. It should be a get call in this format: https://api.zippopotam.us/countryCode/postalCode. Note that we are using https because the Web UI is running in an https context, and that means that we can only do https call, and not http calls. The format of the replied json is as follows:

```
{ "post code": "90210", "country": "United States", "country abbreviation": "US", "places": [ {
"place name": "Beverly Hills", "longitude": "-118.4065", "state": "California", "state
abbreviation": "CA", "latitude": "34.0901" } ] }
```

This corresponds to the following DTO classes in TypeScript that we add to the event handler at the bottom:

```
    //dto class for Zippopotam data
    class ZippopotamDTO {
        "post code": string;
        "country": string;
        "country abbreviation": string;
        "places": ZippopotamPlaceDTO[];
    }
    //dto class for Zippopotam city data
    class ZippopotamPlaceDTO {
        "place name": string;
        "longitude": number;
        "state": string;
        "state abbreviation": string;
        "latitude": number;
    }
```

Note that these classes have to be declared inside the module (so, right above the last closing curly brace).

The next step is to add the method getZippopotamDataAndUpdateCityAndState that will do the https call. Because this function uses fields from the UI, it needs to go into the view form TS handler:

```
        private get CityField():IViewField<string> {
            return this.ViewController.getViewField(Constants.FieldNames.CityAutoField);
        }
        private get StateCodeField():IViewField<string> {
            return this.ViewController.getViewField(Constants.FieldNames.StateCodeAutoField);
        }


        private getZippopotamDataAndUpdateCityAndState(countryCode:string,postalCode: string) {
            //change the country code to an existing official country code
            if (countryCode.toLocaleUpperCase()==="USA") {
                countryCode="US";
            }
            let url=bindTemplateData("https://api.zippopotam.us/{countryCode}/{postalCode}",
{countryCode: countryCode,postalCode: postalCode});
            //do the http GET call asynchronous
            $.ajax({
                url: url,
                method: "GET",
                dataType: "json",
                success: (response: ZippopotamDTO)=> {
                    if (response && response.places.length>0) {
                        this.CityField.Value=response.places[0]["place name"];
                        //update the state code field with the data from Zippopotam
                        this.StateCodeField.Value=response.places[0]["state abbreviation"];
                    } else {
                        this.resetUIStateForNoZippopotamData();
                    }                },
                error: ()=> {
                    this.resetUIStateForNoZippopotamData();
                }
            });
        }
```

```
        private resetUIStateForNoZippopotamData() {
        }
```

Note that we also added 2 properties to easily get a reference to the city field and state code field. And we also already added a method for resetting the UI state if nothing was received (this is for later in this example).

Now we have a method that can be used to get data from zippopotam, and to update the city name field and state code field on the screen.

Next step: Add code to act on UI events and call the external system

Add code to act on UI events and call the external system

After adding the method to call the external system, we need to add code that will be triggered by UI events and calls the method we added in the previous step. Because this code needs to act on field change events, we need to add it to the form TS handler:

```
        /**
         * onFieldChange event handler
         */
        public onFieldChange(viewField: IViewField<any>, eventData: Communication.EventData.
QraView.FieldChangeEventData<any>, processEvent: (processIt?: boolean) => void): void{
            //update city information when zip code or Country code changes
            if (viewField.Name==Constants.FieldNames.ZipCodeAutoField) {
                this.getZippopotamDataAndUpdateCityAndState(this.NgData.suppliers[0].countryCode,
viewField.Value);
            }
            else if (viewField.Name==Constants.FieldNames.CountryCodeAutoField) {
                this.getZippopotamDataAndUpdateCityAndState(viewField.Value,this.NgData.suppliers
[0].zipCode);
            }
        }
```

Now we have code that is triggered by the UI events when a User changes the zip code or country code. This is not enough, we also need to call the external system when our UI loads new data. This can be done in the Main TS handler onBindData event:

```
        public onBindData(eventData: Communication.EventData.QraView.BindDataEventData<any>): void
{
            //update city information when data is loaded
            (<SupplierFormHandler>this.ViewFormTSHandler).updateCityAndState();
        }
```

Note that the above code calls a function in the view form TS handler. That's why we also have to add that method to that view form TS handler:

```
        /**
         * @function updateCityAndState : update the city and state code with data from
ZippopotamData. Use the current zip code and country code
         */
        public updateCityAndState() {
            this.getZippopotamDataAndUpdateCityAndState(this.NgData.suppliers[0].countryCode,this.
NgData.suppliers[0].zipCode);
        }
```

After adding all the above code, the city and state code fields should be automatically updated when the zip code and country code changes.

Next step: Add code to add link to google maps on UI

Add code to add link to google maps on UI

Zippopotam also returns location information for a postal code. We can use that information to add a link on the screen to a google map that shows that location. In order to do that, we have to add html for the link and update the href attribute from the link.

We add the html in the init method of the main TS handler. This method only fires once when the UI is loaded the first time:

```
    /**
     * SupplierMaintHandler : Maint TS handler class.
     *
     * Do not change this class name or the event handler will no longer run.
     *
     */
    export class SupplierMaintHandler extends QraViewTSHandlerWithViewFormTSHandler<DTO.
SupplierMaint, SupplierFormHandler> {
        private _appendedHtml1:JQuery;


        public init() {
            this.addHTML();
        }

        private addHTML() {
            //get the grid cell the city field is in
            let td=$("#"+Constants.FieldNames.CityAutoField).closest("td");
            //append a link for google maps to the grid cell where the city field is in
            this._appendedHtml1=td.append("<a id=\"cityGMLink\" target=\"_blank\">Show on google
maps</a>");
            //get the form field wrapper div
        }


        protected onDestroy() {
            //remove the html we added
            this._appendedHtml1.remove();
        }
```

Note that we also add code to cleanup the html we added. In this case, it is not strictly necessary, it will be automatically cleaned up, but it is good practice to do that.

Now that we have the link element in the html, we can update the href attribute when we receive data from Zippopotam:

```
        private getZippopotamDataAndUpdateCityAndState(countryCode:string,postalCode: string) {
            //change the country code to an existing official country code
            if (countryCode.toLocaleUpperCase()==="USA") {
                countryCode="US";
            }
            let url=bindTemplateData("https://api.zippopotam.us/{countryCode}/{postalCode}",
{countryCode: countryCode,postalCode: postalCode});
            //do the http GET call asynchronous
            $.ajax({
                url: url,
                method: "GET",
                dataType: "json",
                success: (response: ZippopotamDTO)=> {
                    if (response && response.places.length>0) {
                        this.CityField.Value=response.places[0]["place name"];
                        //update the state code field with the data from Zippopotam
                        this.StateCodeField.Value=response.places[0]["state abbreviation"];
                        //update the google maps link
                        this.CityGMLinkElement.attr("href","http://www.google.com/maps/place/"
+response.places[0].latitude+","+response.places[0].longitude);
                    } else {
                        this.resetUIStateForNoZippopotamData();
                    }               },
                error: ()=> {
                    this.resetUIStateForNoZippopotamData();
                }
            });
        }

        private resetUIStateForNoZippopotamData() {
            this.CityGMLinkElement.removeAttr("href");
        }
```

```
private get CityGMLinkElement():JQuery {
    return $("#cityGMLink",this.$element);
}
```

In the above code, the following was added to the existing method:

- new property CityGMLinkElement (bottom)
- to function getZippopotamDataAndUpdateCityAndState : this.CityGMLinkElement.attr("href","http://www.google.com/maps/place/"+response.places[0].latitude+","+response.places[0].longitude);
- to function resetUIStateForNoZippopotamData : this.CityGMLinkElement.removeAttr("href");

So, we add the href attribute when we received data, we remove the attribute if we do not receive anything.

Next step: Add code to replace city field with drop-down box

Add code to replace city field with drop-down box

In some cases, Zippopotam returns multiple city names for one zip code. This is the case in Belgium. In this case, we want to replace the input box with a drop down that gives the possibility to select one of these city names. We do that by hiding the original field and showing a drop down that was added to the html. What we also do, is in case there is only one city returned by Zippopotam, we hide the drop down and show the input again, but we disable the input because there is one correct city name found. Follow these steps to add the code:

1. Add the code to add the html (append to existing code):

```
    export class SupplierMaintHandler extends QraViewTSHandlerWithViewFormTSHandler<DTO.
SupplierMaint, SupplierFormHandler> {
        private _appendedHtml1:JQuery;
        private _appendedHtml2:JQuery;


        public init() {
            this.addHTML();
        }

        private addHTML() {
            //get the grid cell the city field is in
            let td=$("#"+Constants.FieldNames.CityAutoField).closest("td");
            //append a link for google maps to the grid cell where the city field is in
            this._appendedHtml1=td.append("<a id=\"cityGMLink\" target=\"_blank\">Show on
google maps</a>");
            //get the form field wrapper div
            let formFldWrap=$("#"+Constants.FieldNames.CityAutoField).closest(".
formFldWrap");
            //append a div for creating a Kendo dropdown to the form field wrapper div
            this._appendedHtml2=formFldWrap.append("<input required id=\"
cityCustomDropDown\" >");
        }
```

2. Also add an extra line to onDestroy (append to existing code):

```
        protected onDestroy() {
            //remove the html we added
            this._appendedHtml1.remove();
            this._appendedHtml2.remove();
        }
```

3. Now we added the html, but it is an input. So, we need to add code to convert it into a drop down. This is done by creating a method to create the drop down and call that form the constructor of the form view handler:

```
    export class SupplierFormHandler extends QraViewFormTSHandlerV2<DTO.SupplierMaint> {
        constructor(creator: Qad.QraView.TSHandler.TSHandlerCreator<DTO.SupplierMaint>) {
            super(creator);
            //create the Kendo dropdown
            this.createCityCustomDropDown();
        }


        // create the kendo dropdown
        private createCityCustomDropDown() {
            $("#cityCustomDropDown",this.$element).kendoDropDownList({
                change: ()=> { this.cityCustomDropDownOnChange(); }
            });
            this.CityCustomDropDown.element.hide();
        }


        //drop down change event, fires when value was changed and field looses focus
        private cityCustomDropDownOnChange(): void {
            //update the value in city field to make sure that it is in the data object
and saved
            this.CityField.Value = this.CityCustomDropDown.value();
            //update the state code field with the data from Zippopotam
            this.StateCodeField.Value=this.lastZippopotamResponse.places[this.
CityCustomDropDown.select()]["state abbreviation"];
        }
```

```
        //property to easyly get the kendo dropdown object
        private get CityCustomDropDown():kendo.ui.DropDownList {
            return $("#cityCustomDropDown",this.$element).data("kendoDropDownList");
        }

        protected onDestroy() {
            //cleanup the kendo dropdown when we are destroyed
            this.CityCustomDropDown.destroy();
        }
```

Note the code for adding the change event to the drop down, but also to destroy it when the TS handler is destroyed. This is very important to avoid memory leaks.

4. Now that we have a drop down, let's add the code to fill it with city names and to hide/show it and disable the city field (replace existing getZippopotamDataAndUpdateCityAndState method with below code):

```
        private lastZippopotamResponse:ZippopotamDTO;
        private getZippopotamDataAndUpdateCityAndState(countryCode:string,postalCode:
string) {
            //change the country code to an existing official country code
            if (countryCode.toLocaleUpperCase()==="USA") {
                countryCode="US";
            }
            let url=bindTemplateData("https://api.zippopotam.us/{countryCode}/
{postalCode}", {countryCode: countryCode,postalCode: postalCode});
            //do the http GET call asynchronous
            $.ajax({
                url: url,
                method: "GET",
                dataType: "json",
                success: (response: ZippopotamDTO)=> {
                    if (response && response.places.length>0) {
                        //save the response for later use when the item in the dropdown
changes
                        this.lastZippopotamResponse=response;
                        this.CityField.Value=response.places[0]["place name"];
                        if (response.places.length>1) {
                            // hide the city field and show the dropdown in stead
                            this.CityField.IsDisabled=false;
                            this.CityField.Input.hide();
                            this.CityCustomDropDown.wrapper.show();
                            //adjust the tabindex so that focus is set correctly to the
dropdown when tabbing
                            this.CityCustomDropDown.span.attr("tabindex",this.CityField.
Input.attr("tabindex"));
                            //copy the city names to an array suitable for the dropdown
                            let data:string[]=[];
                            response.places.forEach((value: ZippopotamPlaceDTO, index:
number, array: ZippopotamPlaceDTO[]) => {
                                data.push(value["place name"]);
                            });
                            this.CityCustomDropDown.dataSource.data(data);
                            //search the current city value in the dropdown and select it
                            if (this.NgData.suppliers[0].city) {
                                let selected=false;
                                data.forEach( (value: string, index: number, array: string
[]) =>{
                                    if (value.toLocaleUpperCase()===this.NgData.suppliers
[0].city.toLocaleUpperCase()){
                                        this.CityCustomDropDown.select(index);
                                        selected=true;
                                        return false;
                                    }
                                });
                                //if nothing is selected in the dropdown, select the
first item
                                if (!selected)
                                    this.CityCustomDropDown.select(0);
                                //update the state code field with the data from
Zippopotam
                                this.StateCodeField.Value=this.lastZippopotamResponse.
places[this.CityCustomDropDown.select()]["state
abbreviation"];
                            }
```

```
                        } else {
                            //when only one record was received from Zippopotam, show the
city field again and hide the dropdown. Also disable the city field because there is only
one choise
                            this.CityField.Input.show();
                            this.CityField.IsDisabled=true;
                            this.CityCustomDropDown.wrapper.hide();
                            //update the state code field with the data from Zippopotam
                            this.StateCodeField.Value=response.places[0]["state
abbreviation"];
                        }
                        //update the google maps link
                        this.CityGMLinkElement.attr("href","http://www.google.com/maps
/place/"+response.places[0].latitude+","+response.places[0].longitude);
                        this.StateCodeField.Input.change();

                    } else {
                        this.resetUIStateForNoZippopotamData();
                    }
                },
                error: ()=> {
                    this.resetUIStateForNoZippopotamData();
                }
            });
                }

        //when no data was received from Zippopotam, disable the google maps link and
enable the city field again and hide the dropdown
        private resetUIStateForNoZippopotamData() {
            this.CityGMLinkElement.removeAttr("href");
            this.CityField.Input.show();
            this.CityField.IsDisabled=false;
            this.CityCustomDropDown.wrapper.hide();
        }
```

5. This is all. Now you can test as follows: open a supplier, change country code to BE, and then use zip code 9120. Now you should see a drop down with city names. If you enter US country code and an existing zip code, you should see a disabled input with the city name.

Ref: complete event handler code for Suppliers BC

```
module com.qad.erp.base.EventHandler.Supplier.ComQadBasecustom.Maint_AFTER {
    "use strict";


    import QraViewTSHandlerWithViewFormTSHandler = Qad.QraView.TSHandler.
QraViewTSHandlerWithViewFormTSHandler;
    import QraViewFormTSHandlerV2 = Qad.QraView.TSHandler.QraViewFormTSHandlerV2;
    import IViewField = Qad.QraView.TSHandler.IViewField;
    import DTO = com.qad.erp.base.EventHandler.Supplier.DTO;
    import Constants = com.qad.erp.base.EventHandler.Supplier.Constants;


    /**
     * SupplierMaintHandler : Maint TS handler class.
     *
     * Do not change this class name or the event handler will no longer run.
     *
     */
    export class SupplierMaintHandler extends QraViewTSHandlerWithViewFormTSHandler<DTO.
SupplierMaint, SupplierFormHandler> {
        private _appendedHtml1:JQuery;
        private _appendedHtml2:JQuery;

        protected createViewFormTSHandler(): SupplierFormHandler {
            return new SupplierFormHandler(this);

        }
        public onBindData(eventData: Communication.EventData.QraView.BindDataEventData<any>): void
{
            //update city information when data is loaded
            (<SupplierFormHandler>this.ViewFormTSHandler).updateCityAndState();
        }

        public init() {
            this.addHTML();
        }

        private addHTML() {
            //get the grid cell the city field is in
            let td=$("#"+Constants.FieldNames.CityAutoField).closest("td");
            //append a link for google maps to the grid cell where the city field is in
            this._appendedHtml1=td.append("<a id=\"cityGMLink\" target=\"_blank\">Show on google
maps</a>");
            //get the form field wrapper div
            //get the form field wrapper div
            let formFldWrap=$("#"+Constants.FieldNames.CityAutoField).closest(".formFldWrap");
            //append a div for creating a Kendo dropdown to the form field wrapper div
            this._appendedHtml2=formFldWrap.append("<input required id=\"cityCustomDropDown\" >");
        }


        protected onDestroy() {
            //remove the html we added
            this._appendedHtml1.remove();
            this._appendedHtml2.remove();
        }
    }

    export class SupplierFormHandler extends QraViewFormTSHandlerV2<DTO.SupplierMaint> {
        private lastZippopotamResponse:ZippopotamDTO;
        constructor(creator: Qad.QraView.TSHandler.TSHandlerCreator<DTO.SupplierMaint>) {
            super(creator);
            //create the Kendo dropdown
            this.createCityCustomDropDown();
        }


        // create the kendo dropdown
        private createCityCustomDropDown() {
            $("#cityCustomDropDown",this.$element).kendoDropDownList({
                change: ()=> { this.cityCustomDropDownOnChange(); }
            });
```

```
            this.CityCustomDropDown.element.hide();
        }


        //drop down change event, fires when value was changed and field looses focus
        private cityCustomDropDownOnChange(): void {
            //update the value in city field to make sure that it is in the data object and saved
            this.CityField.Value = this.CityCustomDropDown.value();
            //update the state code field with the data from Zippopotam
            this.StateCodeField.Value=this.lastZippopotamResponse.places[this.CityCustomDropDown.
select()]["state abbreviation"];
        }

        //property to easyly get the kendo dropdown object
        private get CityCustomDropDown():kendo.ui.DropDownList {
            return $("#cityCustomDropDown",this.$element).data("kendoDropDownList");
        }

        protected onDestroy() {
            //cleanup the kendo dropdown when we are destroyed
            this.CityCustomDropDown.destroy();
        }


        private get CityField():IViewField<string> {
            return this.ViewController.getViewField(Constants.FieldNames.CityAutoField);
        }
        private get StateCodeField():IViewField<string> {
            return this.ViewController.getViewField(Constants.FieldNames.StateCodeAutoField);
        }

        /**
         * @function updateCityAndState : update the city and state code with data from
ZippopotamData. Use the current zip code and country code
         */
        public updateCityAndState() {
            this.getZippopotamDataAndUpdateCityAndState(this.NgData.suppliers[0].countryCode,this.
NgData.suppliers[0].zipCode);
        }

       /**
         * onFieldChange event handler
         */
        public onFieldChange(viewField: IViewField<any>, eventData: Communication.EventData.
QraView.FieldChangeEventData<any>, processEvent: (processIt?: boolean) => void): void{
            //update city information when zip code or Country code changes
            if (viewField.Name==Constants.FieldNames.ZipCodeAutoField) {
                this.getZippopotamDataAndUpdateCityAndState(this.NgData.suppliers[0].countryCode,
viewField.Value);
            }
            else if (viewField.Name==Constants.FieldNames.CountryCodeAutoField) {
                this.getZippopotamDataAndUpdateCityAndState(viewField.Value,this.NgData.suppliers
[0].zipCode);
            }
        }

        private getZippopotamDataAndUpdateCityAndState(countryCode:string,postalCode: string) {
            //change the country code to an existing official country code
            if (countryCode.toLocaleUpperCase()==="USA") {
                countryCode="US";
            }
            let url=bindTemplateData("https://api.zippopotam.us/{countryCode}/{postalCode}",
{countryCode: countryCode,postalCode: postalCode});
            //do the http GET call asynchronous
            $.ajax({
                url: url,
                method: "GET",
                dataType: "json",
                success: (response: ZippopotamDTO)=> {
                    if (response && response.places.length>0) {
                        //save the response for later use when the item in the dropdown changes
                        this.lastZippopotamResponse=response;
                        this.CityField.Value=response.places[0]["place name"];
                        if (response.places.length>1) {
                            // hide the city field and show the dropdown in stead
                            this.CityField.IsDisabled=false;
                            this.CityField.Input.hide();
                            this.CityCustomDropDown.wrapper.show();
                            //adjust the tabindex so that focus is set correctly to the dropdown
```

```
when tabbing
                            this.CityCustomDropDown.span.attr("tabindex",this.CityField.Input.attr
("tabindex"));
                            //copy the city names to an array suitable for the dropdown
                            let data:string[]=[];
                            response.places.forEach((value: ZippopotamPlaceDTO, index: number,
array: ZippopotamPlaceDTO[]) => {
                                data.push(value["place name"]);
                            });
                            this.CityCustomDropDown.dataSource.data(data);
                            //search the current city value in the dropdown and select it
                            if (this.NgData.suppliers[0].city) {
                                let selected=false;
                                data.forEach( (value: string, index: number, array: string[]) =>{
                                    if (value.toLocaleUpperCase()===this.NgData.suppliers[0].city.
toLocaleUpperCase()){
                                        this.CityCustomDropDown.select(index);
                                        selected=true;
                                        return false;
                                    }
                                });
                                //if nothing is selected in the dropdown, select the first item
                                if (!selected)
                                    this.CityCustomDropDown.select(0);
                                //update the state code field with the data from Zippopotam
                                this.StateCodeField.Value=this.lastZippopotamResponse.places[this.
CityCustomDropDown.select()]["state abbreviation"];
                            }
                        } else {
                            //when only one record was received from Zippopotam, show the city
field again and hide the dropdown. Also disable the city field because there is only one choise
                            this.CityField.Input.show();
                            this.CityField.IsDisabled=true;
                            this.CityCustomDropDown.wrapper.hide();
                            //update the state code field with the data from Zippopotam
                            this.StateCodeField.Value=response.places[0]["state abbreviation"];
                        }
                        //update the google maps link
                        this.CityGMLinkElement.attr("href","http://www.google.com/maps/place/"
+response.places[0].latitude+","+response.places[0].longitude);
                        this.StateCodeField.Input.change();

                    } else {
                        this.resetUIStateForNoZippopotamData();
                    }
                },
                error: ()=> {
                    this.resetUIStateForNoZippopotamData();
                }
            });
        }

        //when no data was received from Zippopotam, disable the google maps link and enable the
city field again and hide the dropdown
        private resetUIStateForNoZippopotamData() {
            this.CityGMLinkElement.removeAttr("href");
            this.CityField.Input.show();
            this.CityField.IsDisabled=false;
            this.CityCustomDropDown.wrapper.hide();
        }

        private get CityGMLinkElement():JQuery {
            return $("#cityGMLink",this.$element);
        }
    }

    //dto class for Zippopotam data
    class ZippopotamDTO {
        "post code": string;
        "country": string;
        "country abbreviation": string;
        "places": ZippopotamPlaceDTO[];
    }
    //dto class for Zippopotam city data
    class ZippopotamPlaceDTO {
        "place name": string;
        "longitude": number;
        "state": string;
        "state abbreviation": string;
```

```
        "latitude": number;
    }
}
```

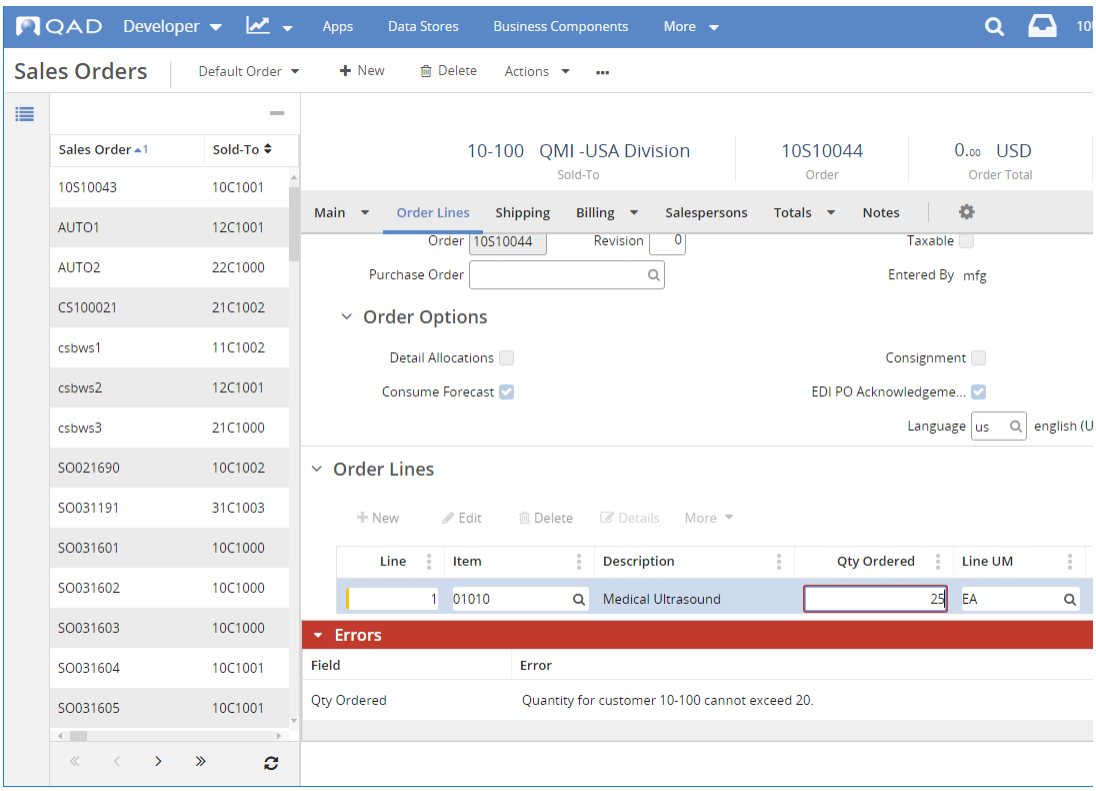
Example 3 - Extend the OOABL: Add extra validation to Sales Order

Table of Contents

Overview

This example is an example on extending the OOABL by adding an extra validation to the Sales Order.

The end result looks as follows:



We will add the following validation to Sales Order create and update: for customer 10-100, for any line item, if the quantity entered is greater than 20, send the message "Quantity for customer 10-100 cannot exceed 20." and reduce the quantity to 20.

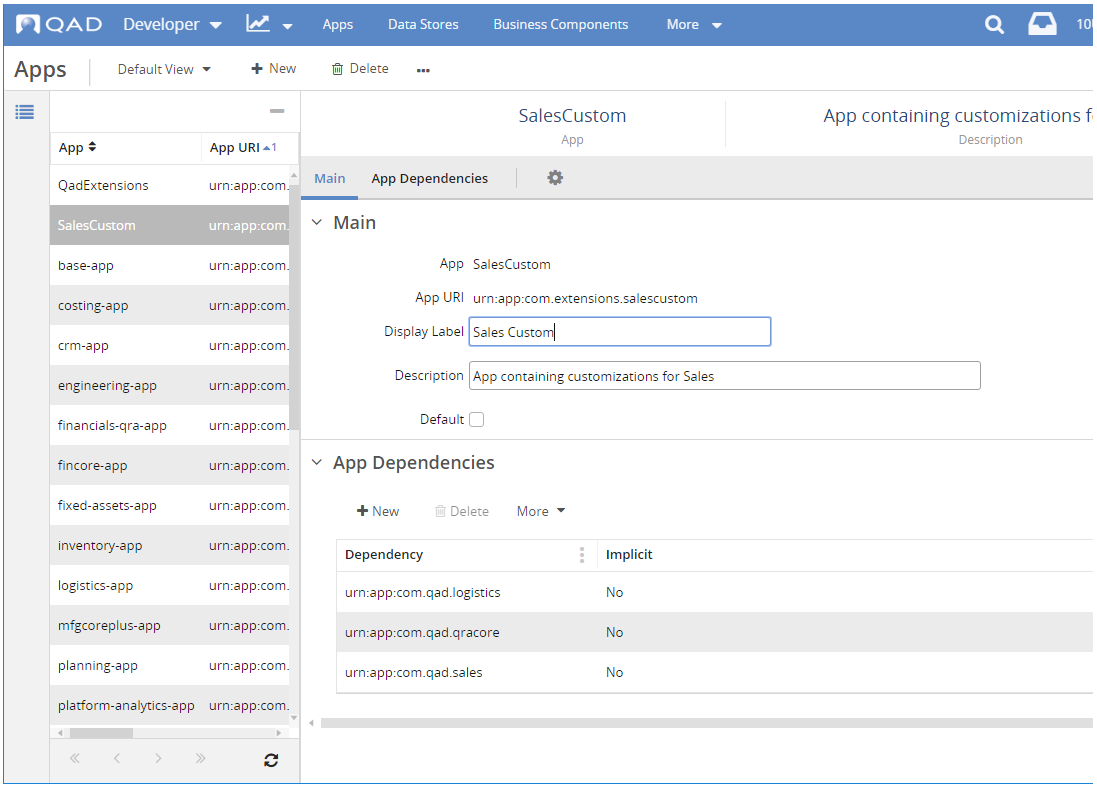
Steps

Step 1: Create a new app

Since we will be customizing the sales app, I'll create a new app named SalesCustom. If you already have an app you can use, you can skip this step.

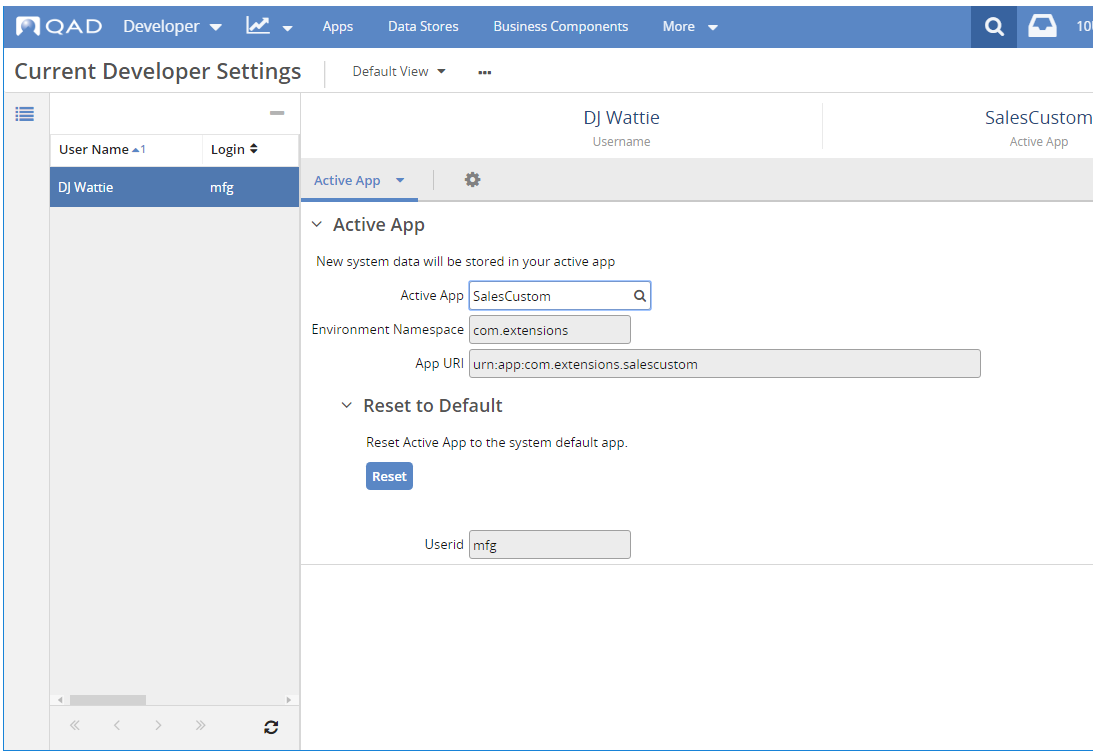
Our new app will have a dependency on the standard sales app, on the logistics app (because Sales Orders use definitions in logistics) and on qracore. The dependency on qracore will automatically be added when you save the app with the other dependencies in place.

Launch **Apps** from the menu and fill the necessary fields.



Step 2: Set the new app as default for yourself

Launch **Current Developer Settings** from the menu and select the app created in Step 1 as default.



Step 3: Export the app to make it ready for customizations

In your environment, run the following command:

yab app-export -dir:<the folder you want to export your app to> -appuri:<the uri of the app you want to export>

In our example, we move to the root folder of our environment and run this:

yab app-export -dir:./salescustom -appuri:urn:app:com.extensions.salescustom

```
[mfg@vmlymw002 systest]$ yab app-export -dir:./salescustom -appuri:urn:app:com.exte

                      app-export (8 tasks)                              [APPLY]
------------------------------------------------------------------------------
1/8 app-export                                              OK (0.006 s)
2/8 metadata-all-export                                     OK (2.183 s)
3/8 app-package-metadata-create                             OK (0.092 s)
4/8 app-export-dependency-update                            OK (2.174 s)
5/8 action-center-dashboard-extract                         OK (2.380 s)
6/8 action-center-kpi-files-extract                         OK (9.817 s)
7/8 action-center-kpi-extract                               OK (5.682 s)
8/8 app-schema-dump                                         OK (0.151 s)
------------------------------------------------------------------------------

BUILD SUCCESSFUL (23.050 s)
```

Step 4: Tell YAB about your new app

Edit configuration.properties in your environment and add the following line:

platform-extension.<appname>.dir=<folder where you exported the app>

In our example, that is:

platform-extension.salescustom.dir=/dr01/qadapps/systest/salescustom

```
#------------------------------------------------------------------
# configuration.properties
#
# This file can be used to overwrite any existing YAB configuration
# settings or to add/upgrade new packages to an environment.
#
# Example: Overwriting an existing property
# netui.telnet.user=odnetui
# appserver.fin.maxsrvrinstance=10
# netui.telnet.maxconnections=200
#
# Example: Add/Upgrade new packages to an environment
# packages.avataxeeconnector=1.2.3.4
#
# Note: To apply any changes to an environment execute:
# yab update
#------------------------------------------------------------------

packages.installation-service-app=1.0.0.0
packages.qad-enterprise-platform=3.4.0.44
packages.role-data-app=3.4.0.35
packages.yab-ee-app=1.8.0.206

appserver.qra.srvrstartupparam=${appserver._base.srvrstartupparam} -mmax 16384 -Bt 10000 -tmpbsize 4 -errorstack
datastore.extension.type=development
openedge.dir=/tech/progress/dlc1173

# @exclude
appserver._base.brkrlogthreshold=
# @exclude
appserver._base.brkrnumlogfiles=
# @exclude
appserver._base.srvrlogthreshold=
# @exclude
appserver._base.srvrnumlogfiles=

platform-extension.salescustom.dir=/dr01/qadapps/systest/salescusto
~
```

Step 5: Write the custom code

1. In your yab environment, go to the folder where your customizations will be created, in our case here, that is salescustom/src/impl/com/extensions/salescustom.
2. Create a subfolder for the customization you're going to do. Since I'm going to customize SalesOrderLine, I create a subfolder salesorder.
3. Write the custom code, in our case in SalesOrderLine.cls.

SalesOrderLine.cls

```
using com.qad.lang.List.
using com.qad.lang.Message.

using com.qad.sales.SalesError.
using com.qad.sales.SalesServices.
using com.qad.sales.salesorder.ISalesOrderHeaderDataObject.
using com.qad.sales.salesorder.ISalesOrderLine.
using com.qad.sales.salesorder.SalesOrderLineBase.
```

```
routine-level on error undo, throw.

class com.extensions.salescustom.salesorder.SalesOrderLine inherits SalesOrderLineBase implements
ISalesOrderLine:
    { com/qad/sales/salesorder/dsSalesOrderLine.i }
    { com/qad/sales/salesorder/dsSalesOrderLineConf.i }

    method public override void CreateWithConfirmation(
        input-output dataset-handle dsSalesOrderLine,
        input-output dataset dsSalesOrderLineConf):

        define variable soDO    as ISalesOrderHeaderDataObject no-undo.
        define variable valMsgs as List                        no-undo.
        define variable msg     as Message                     no-undo.

        /* Copy the sales order lines to the typed dataset */
        dataset dsSalesOrderLine:copy-dataset(dsSalesOrderLine, false, true, true).

        /* For customer 10-100, for any line item, if the quantity entered is greater than 20,
send message "Quantity for customer 10-100 cannot exceed 20." and reduce quantity to 20. */
        assign valMsgs = new List().

        for each ttSalesOrderLine
                 break by ttSalesOrderLine.DomainCode
                       by ttSalesOrderLine.SalesOrderNumber
                 on error undo, throw:
            if first-of(ttSalesOrderLine.SalesOrderNumber)
            then assign soDO = SalesServices:GetSalesOrder():GetSalesOrderByNumber(input
ttSalesOrderLine.DomainCode, input ttSalesOrderLine.SalesOrderNumber).

            if not soDO:available or
               soDO:SoldToCustomerCode <> "10-100"
            then next.

            if ttSalesOrderLine.QuantityOrdered > 20
            then do:
                assign msg = new Message(1, "Quantity for customer 10-100 cannot exceed 20.").
                msg:SetContext(buffer ttSalesOrderLine:buffer-field("QuantityOrdered")).
                valMsgs:Add(msg).
                assign ttSalesOrderLine.QuantityOrdered = 20.
            end.
        end.

        if not valMsgs:IsEmpty()
        then undo, throw new SalesError(valMsgs).

        /* Call the standard code */
        super:CreateWithConfirmation(
            input-output dataset-handle dsSalesOrderLine by-reference,
            input-output dataset dsSalesOrderLineConf by-reference).
    end method.

    method public override void UpdateWithConfirmation(
        input-output dataset-handle dsSalesOrderLine,
        input-output dataset dsSalesOrderLineConf):

        define variable soDO    as ISalesOrderHeaderDataObject no-undo.
        define variable valMsgs as List                        no-undo.
        define variable msg     as Message                     no-undo.

        /* Copy the sales order lines to the typed dataset */
        dataset dsSalesOrderLine:copy-dataset(dsSalesOrderLine, false, true, true).

        /* For customer 10-100, for any line item, if the quantity entered is greater than 20,
send message "Quantity for customer 10-100 cannot exceed 20." and reduce quantity to 20. */
        assign valMsgs = new List().

        for each ttSalesOrderLine
                 break by ttSalesOrderLine.DomainCode
                       by ttSalesOrderLine.SalesOrderNumber
                 on error undo, throw:
            if first-of(ttSalesOrderLine.SalesOrderNumber)
            then assign soDO = SalesServices:GetSalesOrder():GetSalesOrderByNumber(input
ttSalesOrderLine.DomainCode, input ttSalesOrderLine.SalesOrderNumber).

            if not soDO:available or
               soDO:SoldToCustomerCode <> "10-100"
            then next.
```

```
            if ttSalesOrderLine.QuantityOrdered > 20
            then do:
                assign msg = new Message(1, "Quantity for customer 10-100 cannot exceed 20.").
                msg:SetContext(buffer ttSalesOrderLine:buffer-field("QuantityOrdered")).
                valMsgs:Add(msg).
                assign ttSalesOrderLine.QuantityOrdered = 20.
            end.
        end.

        if not valMsgs:IsEmpty()
        then undo, throw new SalesError(valMsgs).

        /* Call the standard code */
        super:UpdateWithConfirmation(
            input-output dataset-handle dsSalesOrderLine by-reference,
            input-output dataset dsSalesOrderLineConf by-reference).
    end method.
end class.
```

Step 6: Compile the code

In your environment, run the following command:

yab dev-<extractfolder>-update

In our case, that is:

yab dev-salescustom-update

```
                dev-salescustom-update (5 tasks)                    [APPLY]
-------------------------------------------------------------------------------
1/5 dev-salescustom-init-check                              OK (0.003 s)
2/5 code-dev-salescustom-db-start-stop                      SKIPPED (0.001 s)
3/5 code-dev-salescustom-update                             OK (0.747 s)
4/5 code-dev-salescustom-db-start-stop                      SKIPPED (0.001 s)
5/5 prolib-dev-salescustom-create                           OK (0.021 s)
-------------------------------------------------------------------------------

BUILD SUCCESSFUL (5.379 s)
```

Step 7: Add this new implementation of ISalesOrderLine to module-config.xml

Go to the folder salescustom/config and add the new ISalesOrderLine to module-config.xml.

module-config.xml

```
<Module>
    <ModuleInfo
            Key="extensions.salescustom"
            Version="@module.version@"
            Vendor="@module.vendor@"
            VendorUrl="@module.vendorurl@"
            DisplayName="Sales Custom"
            Description="App containing customizations for Sales"
            Date="@module.date@"
            ClassName="com.extensions.salescustom.Module"
            Uri="urn:app:com.extensions.salescustom"/>

    <Service
        ServiceClass="com.qad.sales.salesorder.ISalesOrderLine"
        ServiceKey=""
        ImplClass="com.extensions.salescustom.salesorder.SalesOrderLine"
        LifetimePolicy="factory" />
</Module>
```

Step 8: Register the new code in your environment

In your environment, run the following command:

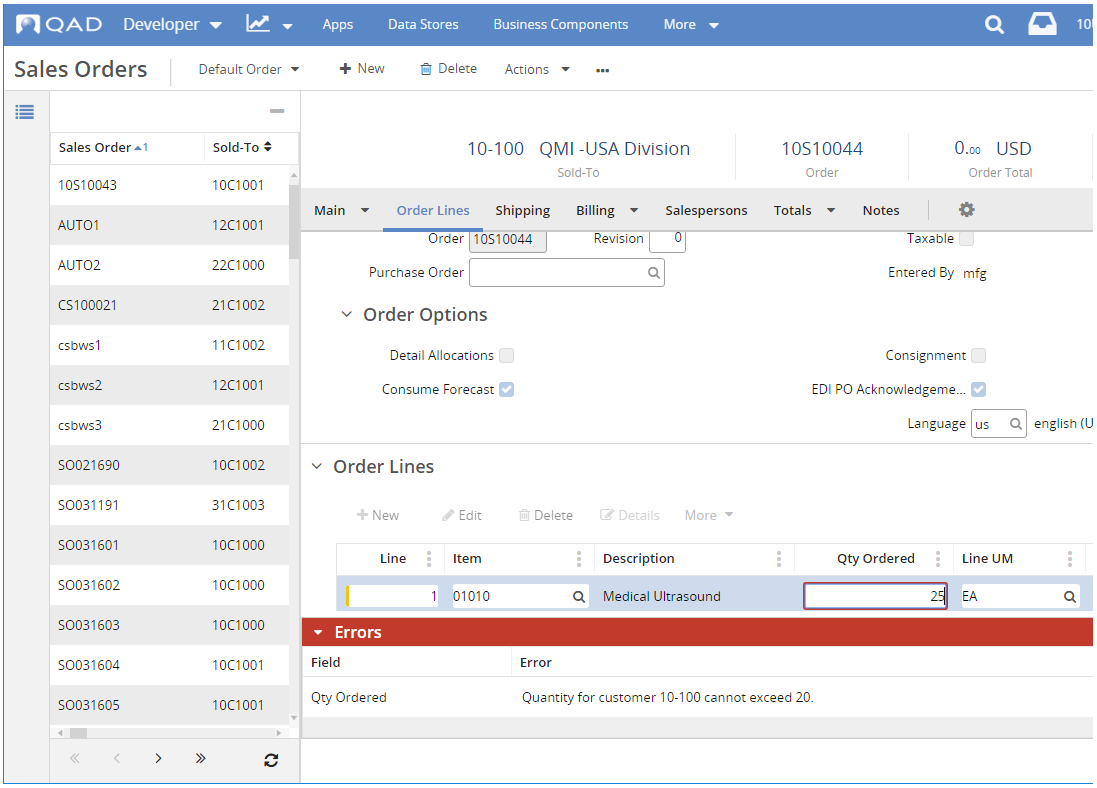
yab dev-<extractfolder>-code-register

In our case, that is:

yab dev-salescustom-code-register

```
             dev-salescustom-code-register (7 tasks)                  [APPLY]
------------------------------------------------------------------------------
1/7 dev-salescustom-code-register                                OK (0.080 s)
2/7 appserver-fin-trim                                           OK (0.863 s)
3/7 appserver-mfg-trim                                           OK (0.612 s)
4/7 appserver-qra-trim                                           OK (0.714 s)
5/7 appserver-qxosi-trim                                         OK (0.417 s)
6/7 appserver-qxoui-trim                                         OK (0.359 s)
7/7 appserver-qxtnative-trim                                     OK (0.358 s)
------------------------------------------------------------------------------

BUILD SUCCESSFUL (3.970 s)
```

Step 9: Test



Example 4: Add data extension to Countries to store Capital, Population, and Currency Code

Table of Contents

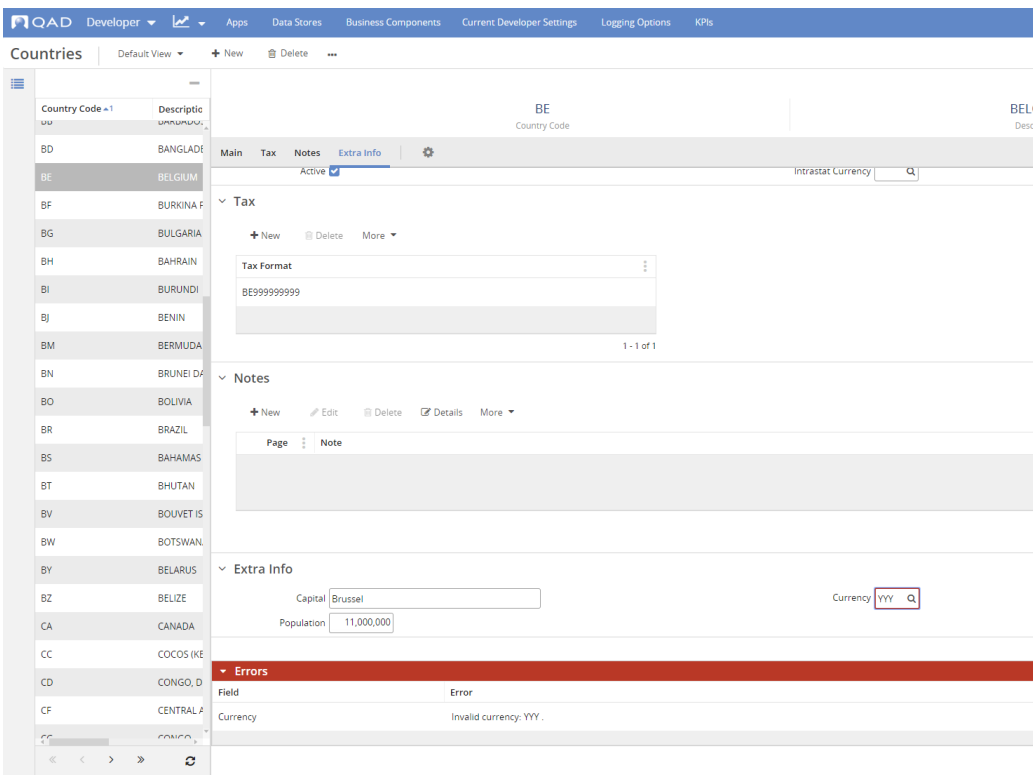
Overview

In this example, we will:

- add extra information to "Countries" (data extension to ICountry)
- have a lookup on Currencies (lookup relation)
- have a validation on the currency code when saving (OOABL extension)

This is what the screen will look like. You can see:

- a new panel named **Extra Info** having 3 fields
- a lookup on the currency field
- a validation error message to validate the currency when saving



"Country" in the standard product is owned by the Base app (**com.qad.base.address.ICountry.cls)**. So in this exercise, we will need to extend/customize the Base App.

We want to store the following extra information:

- Capital
- Population
- Currency Code

We will also make sure Currency Code has a lookup and is validated to be a code existing in the system.

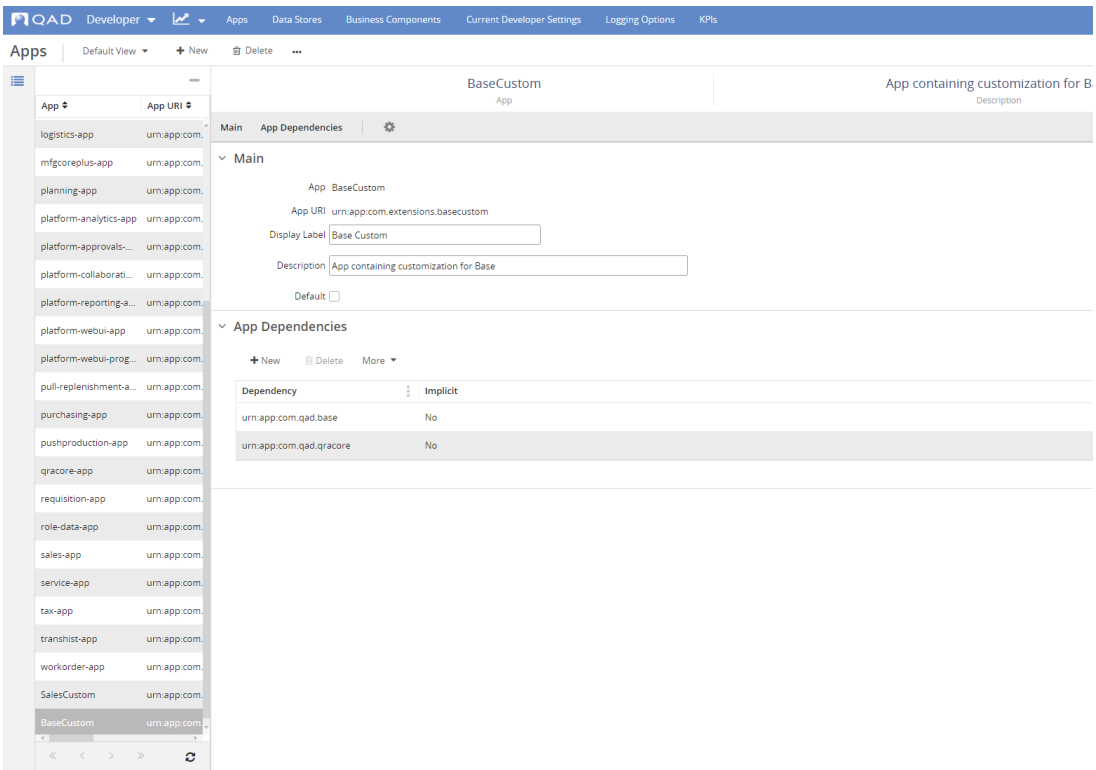
Steps

Step 1: Create a new app

Since we will be customizing the base app, I'll create a new app named BaseCustom. If you already have an app you can use, you can skip this step.

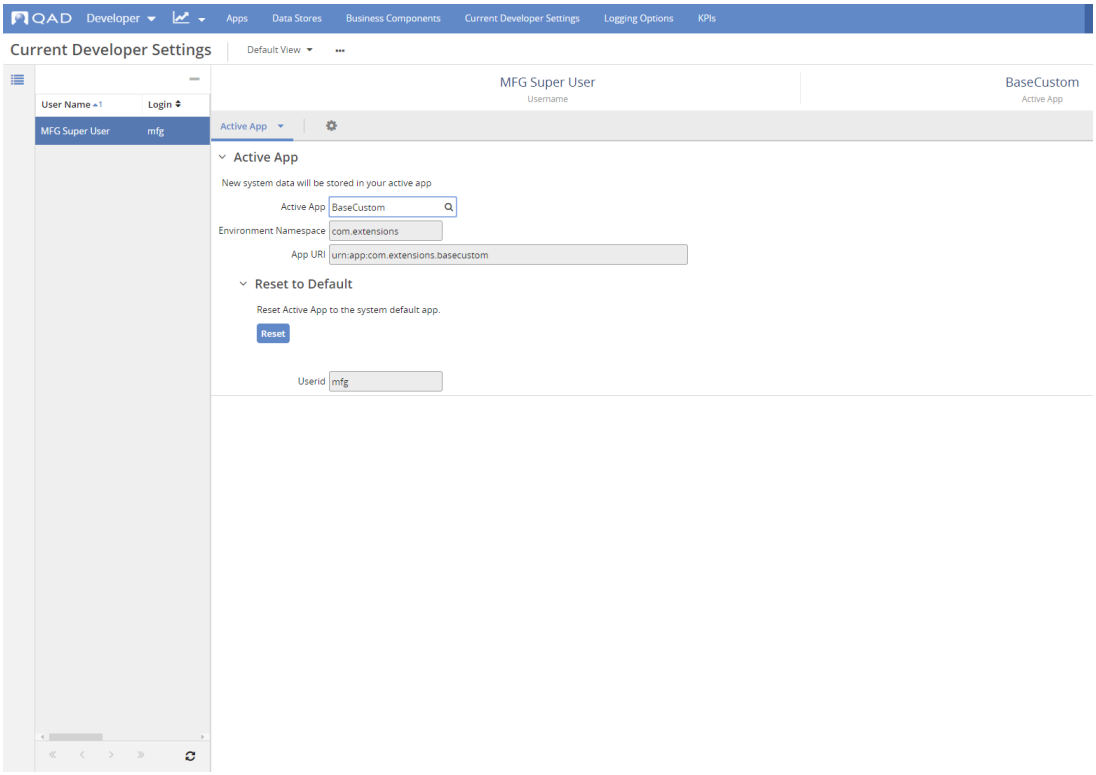
Our new app will have a dependency on the standard base app and on qracore. Note the dependency on qracore will automatically be added when you save the app with the other dependencies in place.

Launch **Apps** from the menu and fill the necessary fields.



Step 2: Set the new app as default for yourself

Launch **Current Developer Settings** from the menu and select the app created in Step 1 as default.



Step 3: Export the app to make it ready for customizations

In your environment, run the following command:

yab app-export -dir:<the folder you want to export your app to> -appuri:<the uri of the app you want to export>

In our example, we move to the root folder of our environment and run this:

yab app-export -dir:./basecustom -appuri:urn:app:com.extensions.basecustom

```
                    app-export (8 tasks)                              [APPLY]
-------------------------------------------------------------------------------
1/8 app-export                                              OK (0.008 s)
2/8 metadata-all-export                                     OK (1.224 s)
3/8 app-package-metadata-create                             OK (0.077 s)
4/8 app-export-dependency-update                            OK (3.044 s)
5/8 action-center-dashboard-extract                         OK (2.017 s)
6/8 action-center-kpi-files-extract                         OK (7.753 s)
7/8 action-center-kpi-extract                               OK (4.912 s)
8/8 app-schema-dump                                         OK (0.136 s)
-------------------------------------------------------------------------------

BUILD SUCCESSFUL (19.978 s)
```

Step 4: Tell YAB about your new app

Edit configuration.properties in your environment and add the following line:

platform-extension.<appname>.dir=<folder where you exported the app>

In our example, that is:

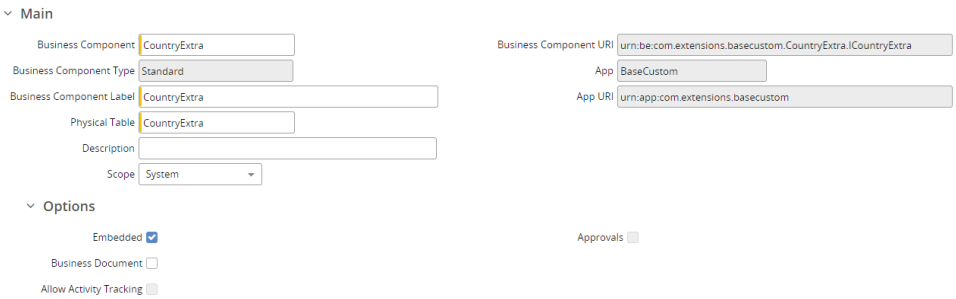
platform-extension.basecustom.dir=/dr01/qadapps/systest/basecustom

```
#----------------------------------------------------------------------
# configuration.properties
#
# This file can be used to overwrite any existing YAB configuration
# settings or to add/upgrade new packages to an environment.
#
# Example: Overwriting an existing property
```

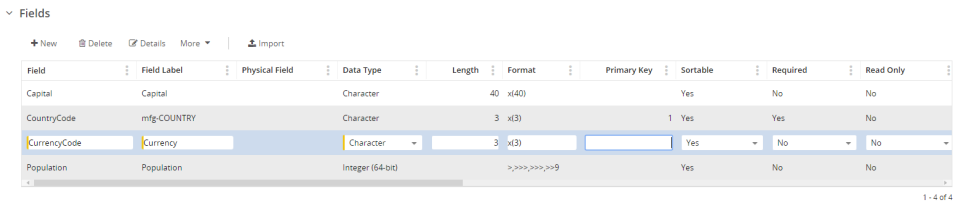
```
# netui.telnet.user=odnetui
# appserver.fin.maxsrvrinstance=10
# netui.telnet.maxconnections=200
#
# Example: Add/Upgrade new packages to an environment
# packages.avataxeeconnector=1.2.3.4
#
# Note: To apply any changes to an environment execute:
# yab update
#-----------------------------------------------------------------------

platform-extension.basecustom.dir=/dr01/qadapps/systest/basecustom
```

Step 5: Create the data extension

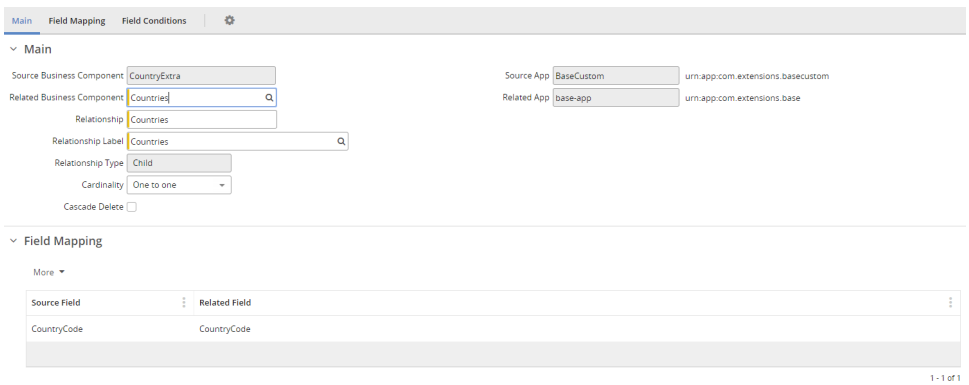
1. Open **Business Components** and create a new one.
2. **Main** panel: fill in the Main panel like this (make sure the **Embedded** checkbox is selected).



3. **Fields** panel: fill in the Fields as listed in the overview.
   Note we also need to add CountryCode or we won't be able to link this extension to the main component.



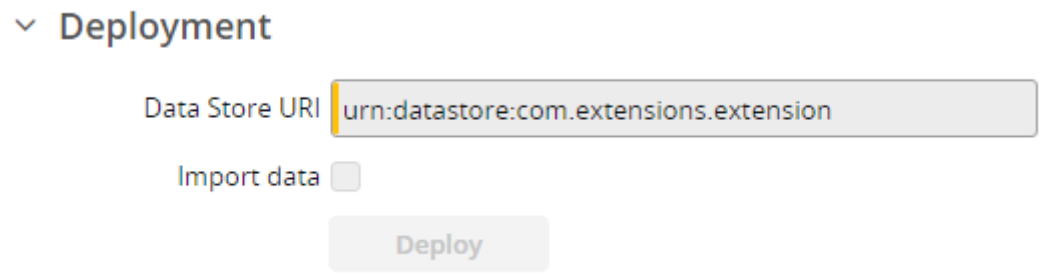
4. **Save**.
5. **Relationships** panel: now define the extension relation between the extension and Countries.



6. **Save** and **Close** the relationship pop-up window and **Save** the business component.

Step 6: Deploy the data extension

In the business component form, locate the **Deployment** panel, select a data store, and then click **Deploy**.

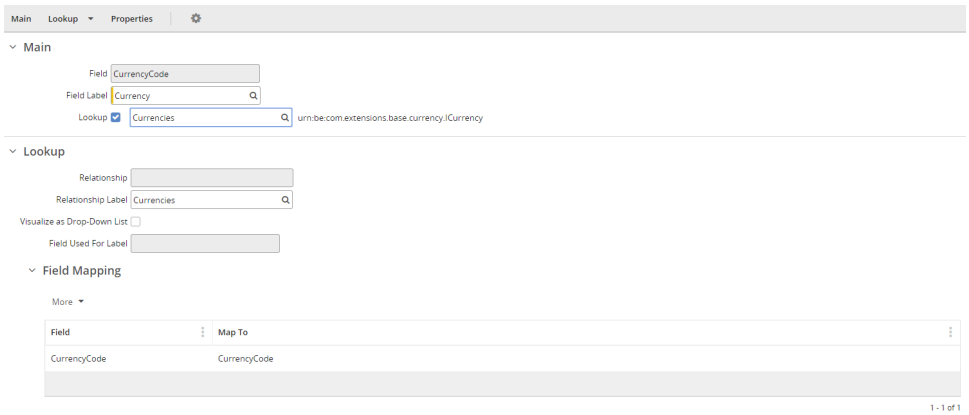


Step 7: Check it out

Run **Countries** from the menu, scroll down and see the extra fields.

Step 8: Define the lookup for Currency

1. Run **Business Components** and open your extension.
2. Locate the **Fields** panel, select the CurrencyCode field, and then click **Details**.
3. Select the **Lookup** checkbox and define the lookup relation to Currencies like this.



4. Click **Save**, **Close**, and **Save** again on the Business Components form.

Step 9: Check it out again

Run **Countries** from the menu, scroll down and see the lookup on Currency. If you do not see it the first time, click **Refresh** in your browser.

Step 10: Write a validation on Currency. Only valid Currencies should be entered.

1. In your yab environment, go to the folder where your customizations will be created, in our case here, that is base custom/src/impl/com/extensions/basecustom.
2. All data extensions are handled by **VirtualBusinessEntity** so we will have to customize this class.
3. Create a subfolder for the customization you're going to do. Since I'm going to customize VirtualBusinessEntity, I create a subfolder be.

4. Write the custom code, in our case in VirtualBusinessEntity.cls.

VirtualBusinessEntity.cls

```
using com.qad.lang.List.
using com.qad.lang.Message.

using com.qad.qra.be.IVirtualBusinessEntity.
using com.qad.qra.be.VirtualBusinessEntityBase.

using com.qad.base.BaseError.
using com.qad.base.BaseMessageKey.
using com.qad.base.BaseServices.
using com.qad.base.currency.ICurrencyDataObject.

routine-level on error undo, throw.

class com.extensions.basecustom.be.VirtualBusinessEntity inherits VirtualBusinessEntityBase
implements IVirtualBusinessEntity:
    method public override void Create(
        input entityURI as character,
        input dataset-handle dsEntity):

        define variable buf     as handle              no-undo.
        define variable qry     as handle              no-undo.
        define variable currDO  as ICurrencyDataObject no-undo.
        define variable valMsgs as List                no-undo.
        define variable msg     as Message             no-undo.

        /* Only valid currency codes for our extension on Countries */
        if entityURI = "urn:be:com.extensions.basecustom.CountryExtra.ICountryExtra"
        then do on error undo, throw:
            assign valMsgs = new List()
                   buf     = dsEntity:get-buffer-handle("ttCountryExtra").

            create query qry.
            qry:set-buffers(buf).
            qry:query-prepare("for each ttCountryExtra").
            qry:query-open().
            qry:get-first().

            do while not qry:query-off-end:
                assign currDO = BaseServices:GetCurrency():GetCurrencyByCode(input buf::
CurrencyCode).

                if not currDO:Available
                then do:
                    assign msg = new Message(BaseMessageKey:INVALID_CURRENCY, buf::CurrencyCode).
                    msg:SetContext(buf:buffer-field("CurrencyCode")).
                    valMsgs:Add(msg).
                end.

                qry:get-next().
            end.

            if not valMsgs:IsEmpty()
            then undo, throw new BaseError(valMsgs).

            finally:
                if qry:is-open
                then qry:query-close().

                delete object qry.
                assign qry = ?.
            end.
        end.

        /* Call the standard code */
        super:Create(
            input entityURI,
            input dataset-handle dsEntity by-reference).
    end method.

    method public override void Update(
        input entityURI as character,
        input dataset-handle dsEntity):
```

```
        define variable buf     as handle             no-undo.
        define variable qry     as handle             no-undo.
        define variable currDO  as ICurrencyDataObject no-undo.
        define variable valMsgs as List               no-undo.
        define variable msg     as Message            no-undo.

        /* Only valid currency codes for our extension on Countries */
        if entityURI = "urn:be:com.extensions.basecustom.CountryExtra.ICountryExtra"
        then do on error undo, throw:
            assign valMsgs = new List()
                   buf     = dsEntity:get-buffer-handle("ttCountryExtra").

            create query qry.
            qry:set-buffers(buf).
            qry:query-prepare("for each ttCountryExtra").
            qry:query-open().
            qry:get-first().

            do while not qry:query-off-end:
                assign currDO = BaseServices:GetCurrency():GetCurrencyByCode(input buf::
CurrencyCode).

                if not currDO:Available
                then do:
                    assign msg = new Message(BaseMessageKey:INVALID_CURRENCY, buf::CurrencyCode).
                    msg:SetContext(buf:buffer-field("CurrencyCode")).
                    valMsgs:Add(msg).
                end.

                qry:get-next().
            end.

            if not valMsgs:IsEmpty()
            then undo, throw new BaseError(valMsgs).

            finally:
                if qry:is-open
                then qry:query-close().

                delete object qry.
                assign qry = ?.
            end.
        end.

        /* Call the standard code */
        super:Update(
            input entityURI,
            input dataset-handle dsEntity by-reference).
    end method.
end class.
```

Step 11: Compile the code

In your environment, run the following command:

yab dev-<extractfolder>-update

In our case, that is:

yab dev-basecustom-update

```
                dev-basecustom-update (14 tasks)                    [APPLY]
------------------------------------------------------------------------------
1/14  database-dev-basecustom-extension-structure-list      OK (0.036 s)
2/14  database-dev-basecustom-extension-structure-file-upda OK (0.014 s)
3/14  database-dev-basecustom-extension-structure-validate  OK (0.005 s)
4/14  database-dev-basecustom-extension-create              OK (0.003 s)
5/14  database-dev-basecustom-extension-structure-update    OK (0.017 s)
6/14  database-dev-basecustom-extension-schema-update       OK (0.112 s)
7/14  database-dev-basecustom-extension-schema-snapshot     SKIPPED (0.003 s)
8/14  database-dev-basecustom-extension-data-xml-update     OK (0.013 s)
9/14  database-dev-basecustom-extension-data-dotd-update    OK (0.035 s)
10/14 dev-basecustom-init-check                             OK (0.003 s)
11/14 code-dev-basecustom-db-start-stop                     SKIPPED (0.001 s)
12/14 code-dev-basecustom-update                            OK (0.374 s)
```

```
13/14 code-dev-basecustom-db-start-stop                    SKIPPED (0.001 s)
14/14 prolib-dev-basecustom-create                         OK (0.008 s)
--------------------------------------------------------------------------------

BUILD SUCCESSFUL (1.306 s)
```

Step 12: Add this new implementation of IVirtualBusinessEntity to module-config.xml

Go to the folder basecustom/config and add the new IVirtualBusinessEntity to module-config.xml.

module-config.xml

```
<Module>
    <ModuleInfo
            Key="extensions.basecustom"
            Version="@module.version@"
            Vendor="@module.vendor@"
            VendorUrl="@module.vendorurl@"
            DisplayName="Base Custom"
            Description="App containing customization for Base"
            Date="@module.date@"
            ClassName="com.extensions.basecustom.Module"
            Uri="urn:app:com.extensions.basecustom"/>

    <Service
        ServiceClass="com.qad.qra.be.IVirtualBusinessEntity"
        ServiceKey=""
        ImplClass="com.extensions.basecustom.be.VirtualBusinessEntity"
        LifetimePolicy="factory" />
</Module>
```

Step 13: Register the new code in your environment

In your environment, run the following command:

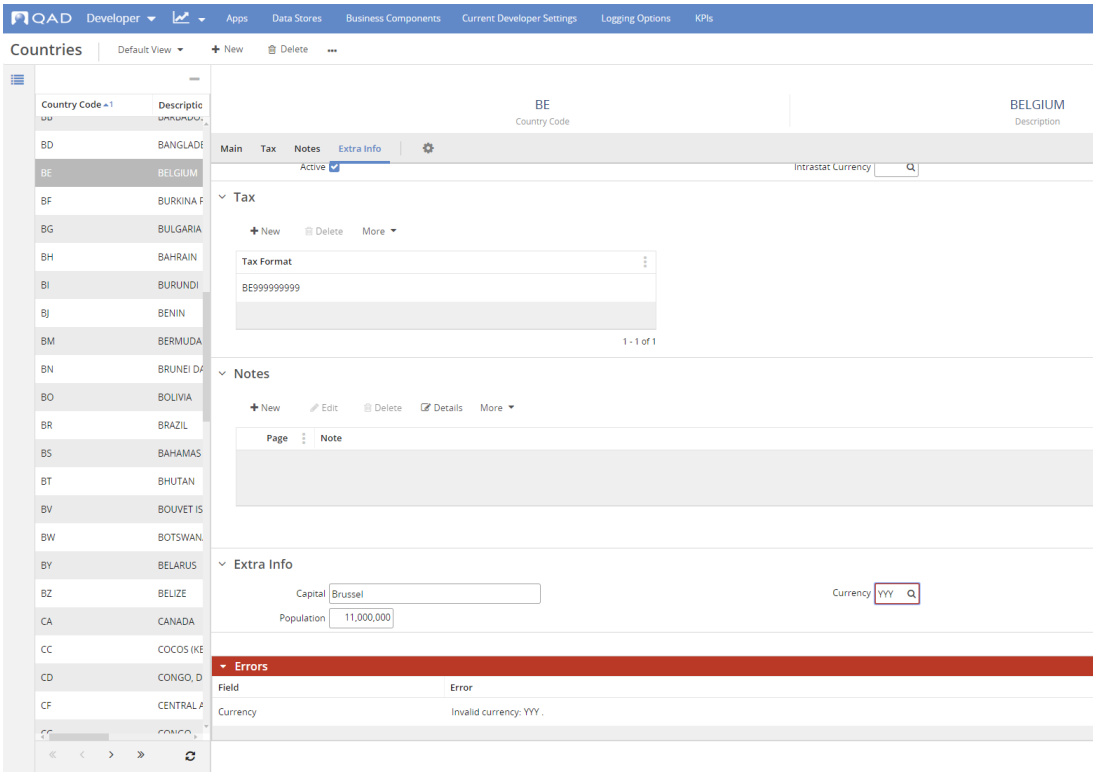
yab dev-<extractfolder>-code-register

In our case, that is:

yab dev-basecustom-code-register

```
             dev-basecustom-code-register (7 tasks)                  [APPLY]
--------------------------------------------------------------------------------
1/7 dev-basecustom-code-register                           OK (0.074 s)
2/7 appserver-fin-trim                                     OK (0.340 s)
3/7 appserver-mfg-trim                                     OK (0.862 s)
4/7 appserver-qra-trim                                     OK (0.610 s)
5/7 appserver-qxosi-trim                                   OK (0.356 s)
6/7 appserver-qxoui-trim                                   OK (0.307 s)
7/7 appserver-qxtnative-trim                               OK (0.308 s)
--------------------------------------------------------------------------------

BUILD SUCCESSFUL (3.378 s)
```

Step 14: Test


QAD Developer
Apps
Data Stores
Business Components
Current Developer Settings
Logging Options
KPIs
Countries
Default View
New
Delete
Country Code
Descriptic
BD BANGLADE
BE BELGIUM
BF BURKINA F
BG BULGARIA
BH BAHRAIN
BI BURUNDI
BJ BENIN
BM BERMUDA
BN BRUNEI DA
BO BOLIVIA
BR BRAZIL
BS BAHAMAS
BT BHUTAN
BV BOUVET IS
BW BOTSWAN
BY BELARUS
BZ BELIZE
CA CANADA
CC COCOS (KE
CD CONGO, D
CF CENTRAL A
BE
Country Code
BELGIUM
Description
Main
Tax
Notes
Extra Info
Active
Intrastat Currency
Tax
New
Delete
More
Tax Format
BE999999999
1 - 1 of 1
Notes
New
Edit
Delete
Details
More
Page
Note
Extra Info
Capital
Brussel
Currency
YYY
Population
11,000,000
Errors
Field
Error
Currency
Invalid currency: YYY .

App Security

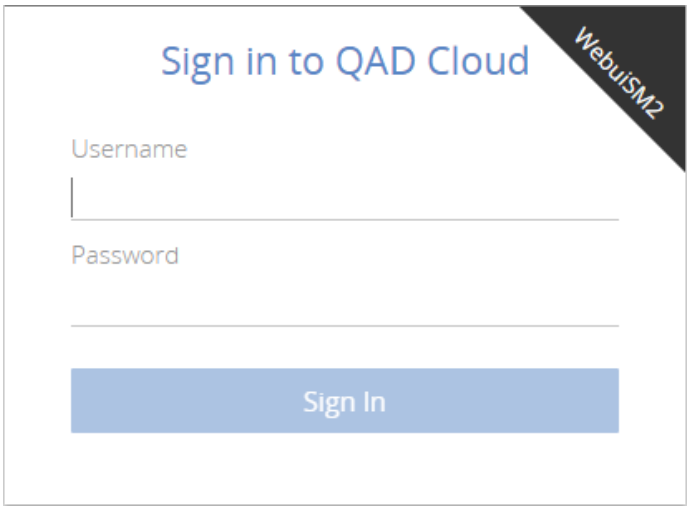
This section covers a brief overview of QAD Enterprise Platform security features as it relates to App development.

Security Model

QAD Enterprise Platform provides a core set of security services that come as part of the platform. Here is a quick overview of what some of these services do and how the security model works.

Authentication

Authentication validates the identity of the user. For Channel Islands this is provided when signing in to the application



Authentication types supported by QAD Enterprise Platform:

- UI - Form based authentication
- API - OAuth2 and HTTP Basic

QAD Enterprise platform integrates with the providers

- Built in DB Users
- LDAP/Active Directory
- SAML SSO
- OAuth2
- X.509 Certificates (Smart Card authentication)

Authorization

Authorization is the process of granting a user access to a system resource (menu, browse, view, fields, etc). Authorization is granted through permissions granted to a set of Roles that are assigned to a given user.

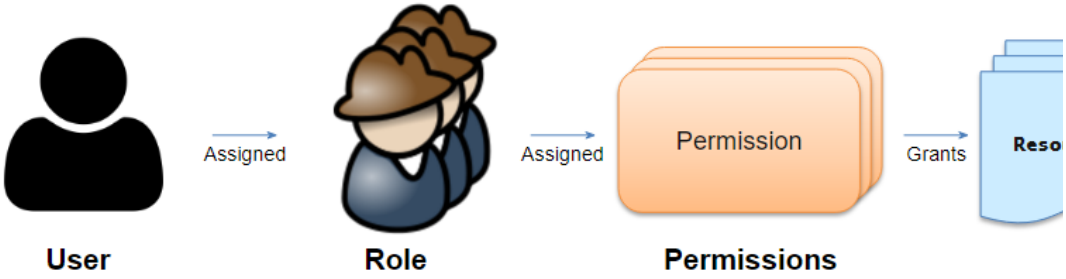
Session Management

QAD Enterprise Platform provides a session management service. This service should be transparent to App developers.

Role Based Access Control

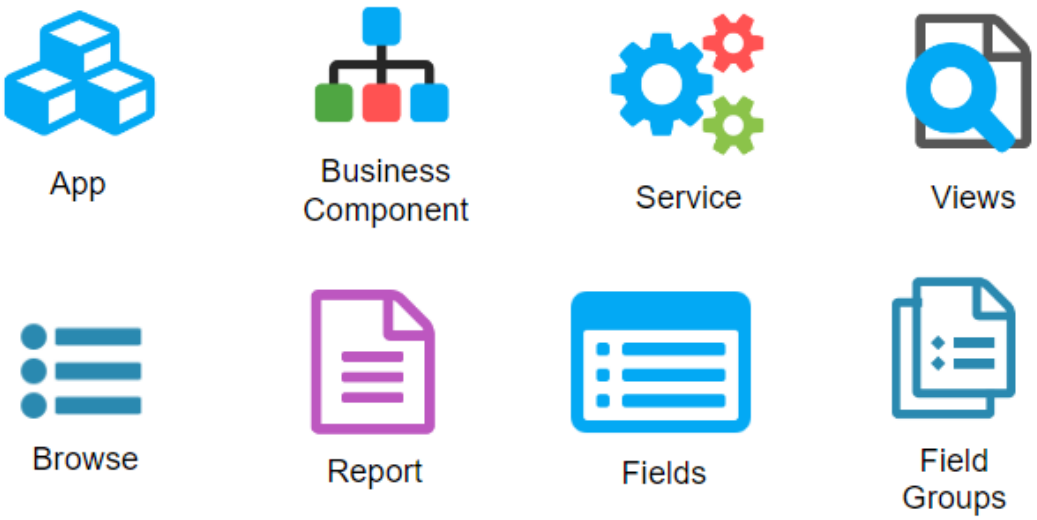
QAD Enterprise Platform supports a Role based access control model. In it Roles are defined at the system level and users can be assigned multiple Roles.

At a high level a User is assigned a number of Roles. These Roles are assigned Permissions which grant or deny access to resources in the system



Resources

A Resource is anything that can be uniquely identified in the system (URI) and has been designated to be secured.



Users

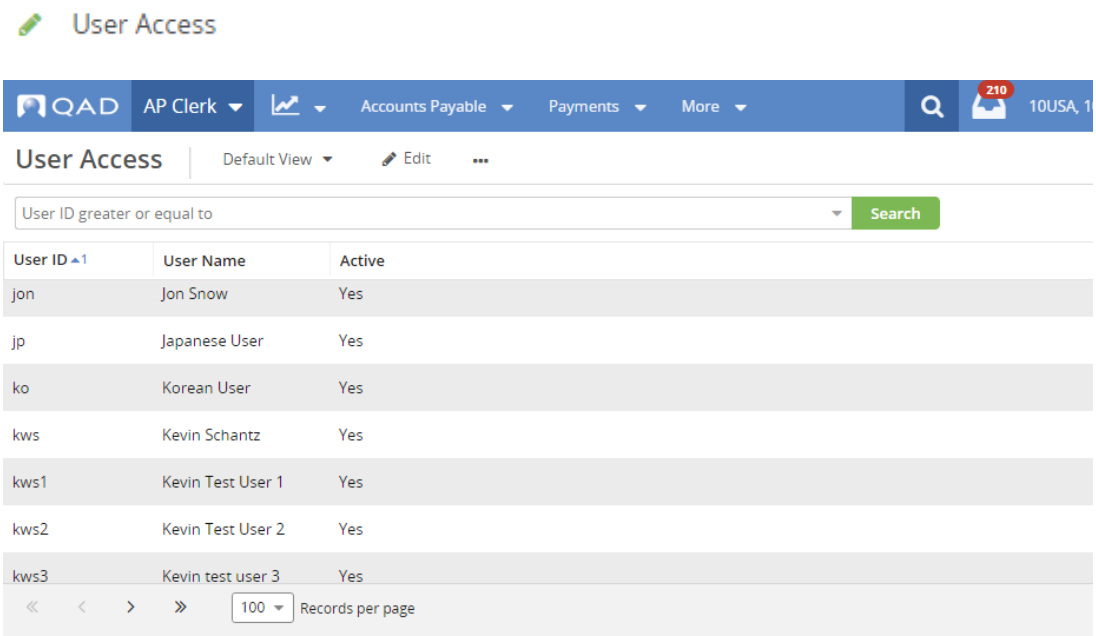
Users can be managed by a system administrator through the .NetUI application or externally defined in a directory service such as LDAP or Active Directory.

- Users defined by administrator
- Must have webui_user role for CI access
- Must be assigned roles for their specific job function

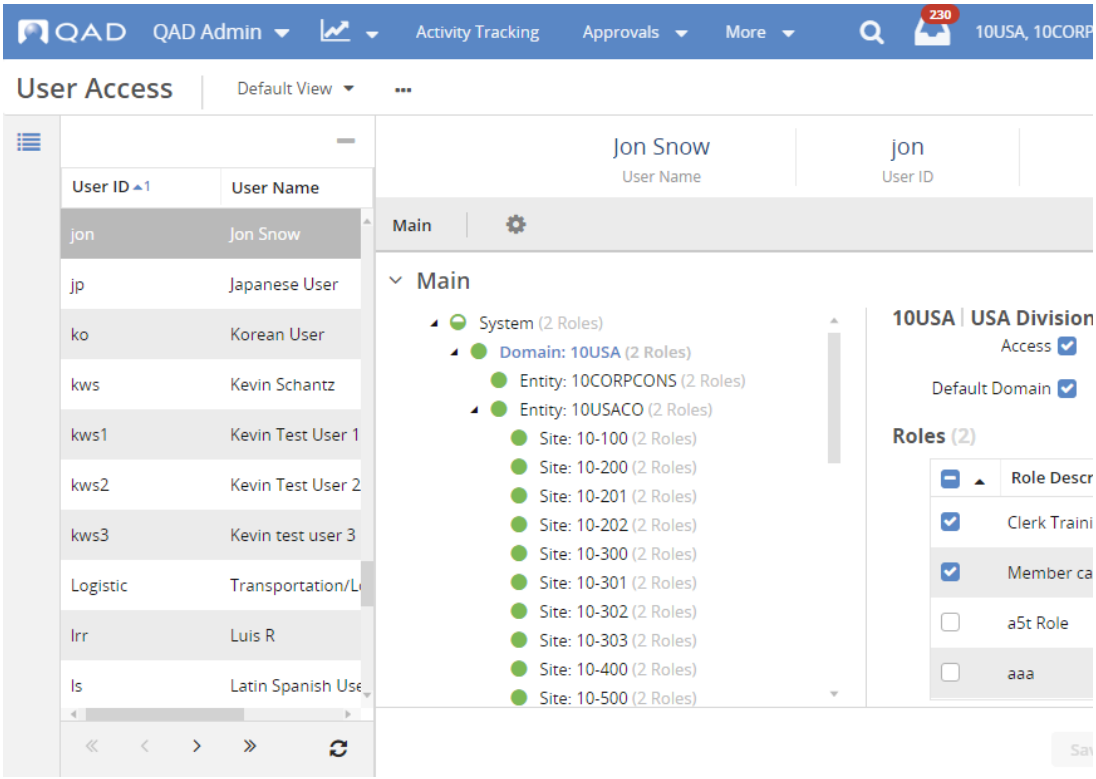
User Access

Roles are defined at a system level but the assignment of these roles to a user depends on the Security Context (System, Domain, Entity, Site).

This maintenance screen combines user domain/entity access for users familiar with .NetUI from Enterprise Edition environments.



- Allows admin to configure user access
  - System
  - Domain
  - Entities
  - Sites
  - Role assignment
- Combines User domain/entity access maintain and role membership from Enterprise Edition for administrators familiar it.

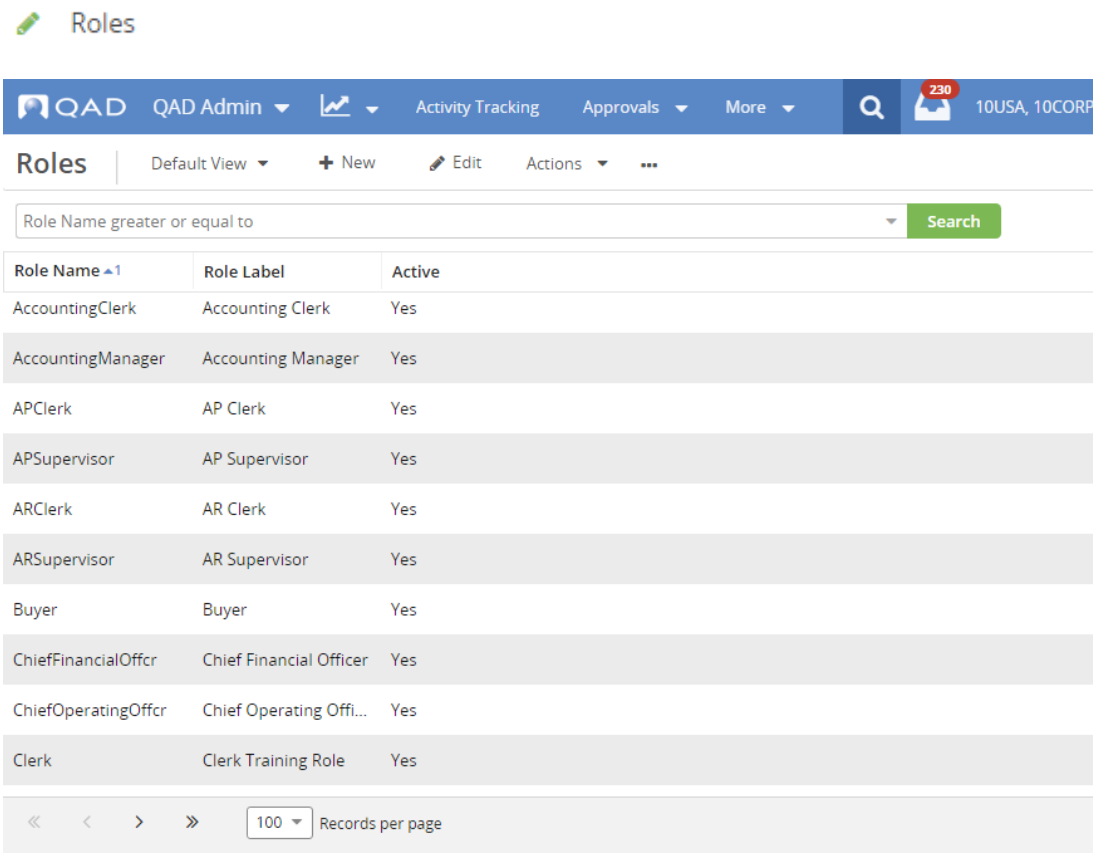

QAD
QAD Admin
Activity Tracking
Approvals
More
230
10USA, 10CORP
User Access
Default View
User ID
User Name
jon
Jon Snow
jp
Japanese User
ko
Korean User
kws
Kevin Schantz
kws1
Kevin Test User 1
kws2
Kevin Test User 2
kws3
Kevin test user 3
Logistic
Transportation/L
lrr
Luis R
ls
Latin Spanish Use
Jon Snow
User Name
jon
User ID
Main
Main
System (2 Roles)
Domain: 10USA (2 Roles)
Entity: 10CORPCONS (2 Roles)
Entity: 10USACO (2 Roles)
Site: 10-100 (2 Roles)
Site: 10-200 (2 Roles)
Site: 10-201 (2 Roles)
Site: 10-202 (2 Roles)
Site: 10-300 (2 Roles)
Site: 10-301 (2 Roles)
Site: 10-302 (2 Roles)
Site: 10-303 (2 Roles)
Site: 10-400 (2 Roles)
Site: 10-500 (2 Roles)
10USA | USA Division
Access
Default Domain
Roles (2)
Role Descr
Clerk Traini
Member ca
a5t Role
aaa

Roles

All permissions are assigned to a given Role. A user can move through an organization, wear many hats and be assigned multiple roles.

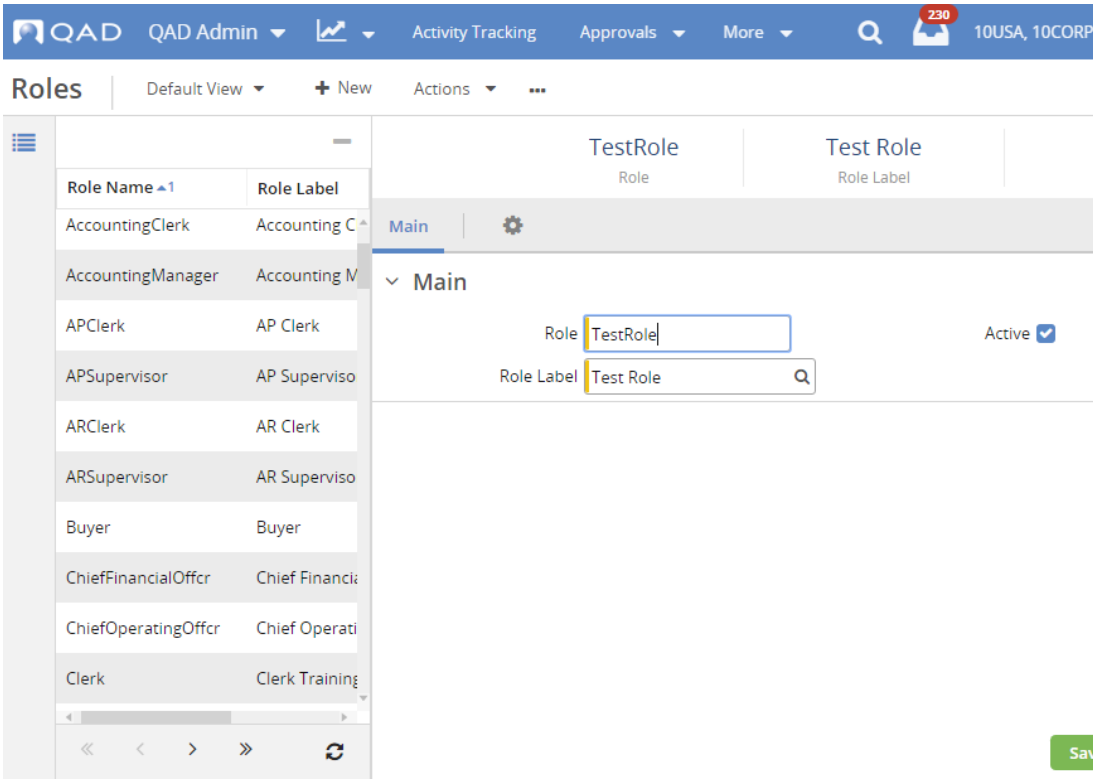
- Roles are defined at the system level
- Roles & role menus delivered by QAD are samples only
- QAD reserves the right to change sample role menus or permissions
- Customers should create their own roles
- Customers may copy existing role menus

To define a new role use the Roles maintenance screen



To define a new Role

- Click on New
- Enter RoleName and
- Enter a description or label for the role.
- Click Save.

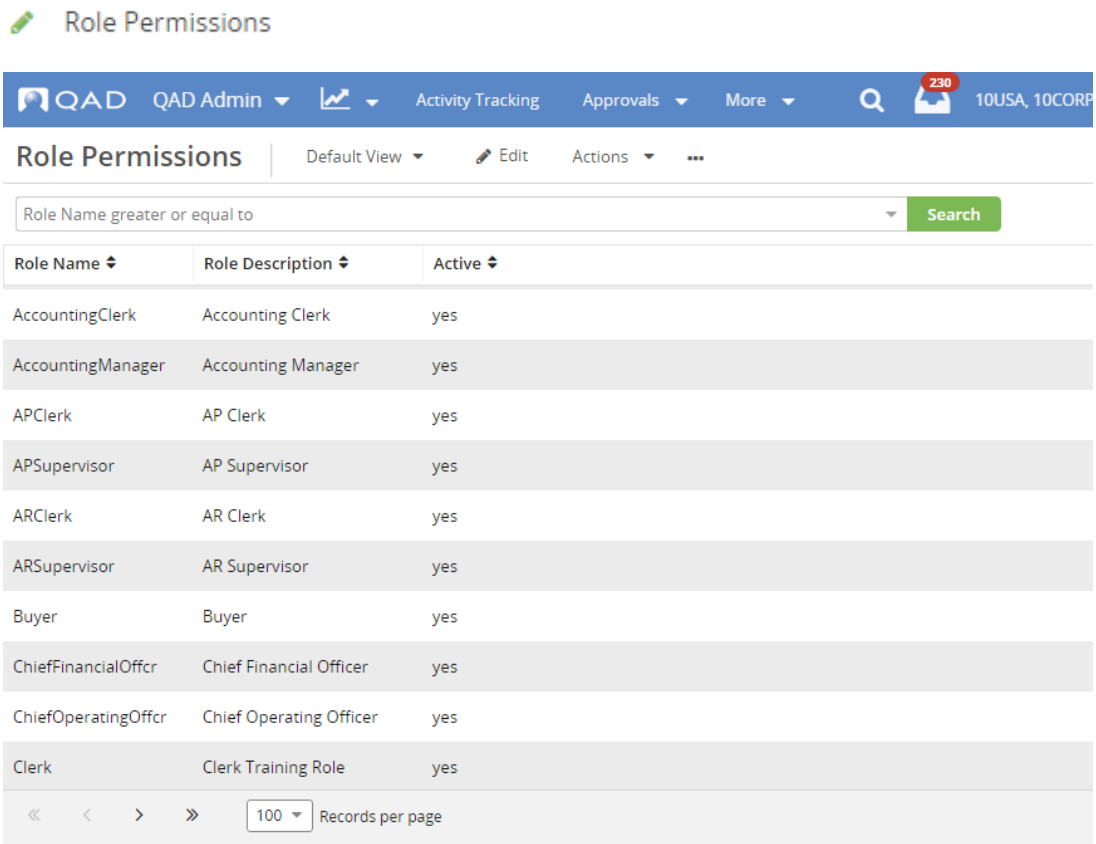

QAD
QAD Admin
Activity Tracking
Approvals
More
230
10USA, 10CORP
Roles
Default View
New
Actions
Role Name
Role Label
AccountingClerk
Accounting C
AccountingManager
Accounting M
APClerk
AP Clerk
APSupervisor
AP Superviso
ARClerk
AR Clerk
ARSupervisor
AR Superviso
Buyer
Buyer
ChiefFinancialOffcr
Chief Financia
ChiefOperatingOffcr
Chief Operati
Clerk
Clerk Training
TestRole
Role
Test Role
Role Label
Main
Main
Role
TestRole
Active
Role Label
Test Role

Role Permissions

Permissions depend on the type of resource being granted.

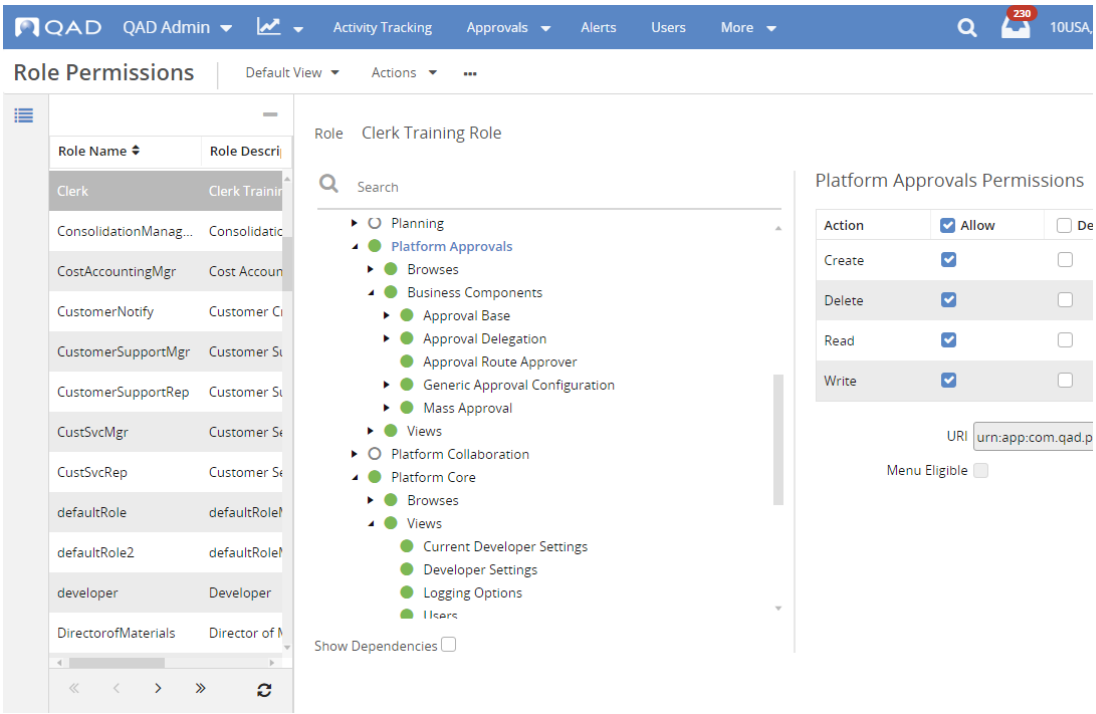
Resource Type	Permissions
Apps	Approve, Create, Delete, Read,
Business Components	Approve, Create, Delete, Read,
Browses	Read
Dashboard & KPIs	Read, Write, Delete
Reports	Read
Services	Create, Delete, Read, Write
Fields	Read, Write
Field Groups	Read, Write
Views	Read

Roles have permissions assigned through Role Permissions screen.

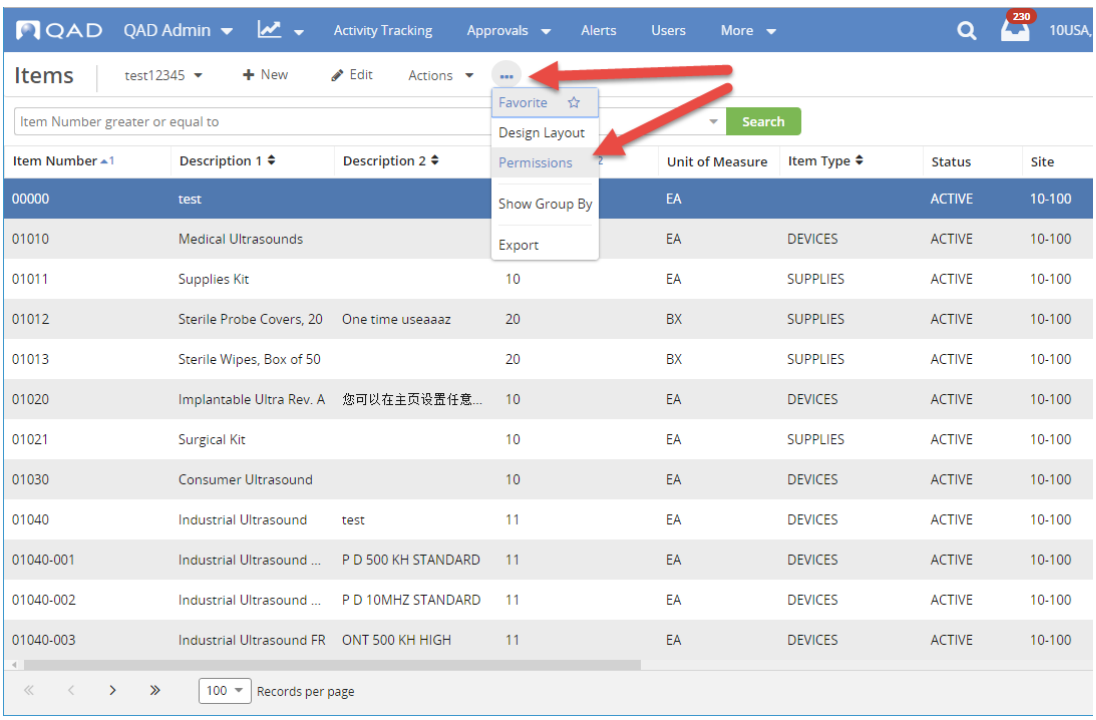


To assign/change permissions

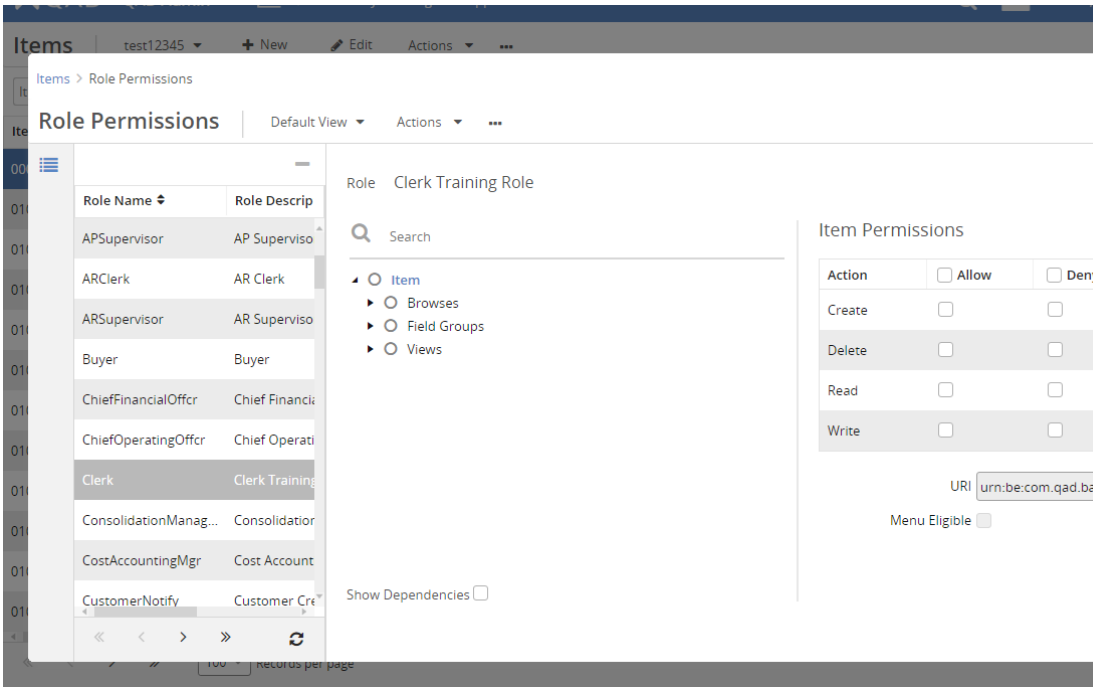
- Select Role (double click)
- Expand tree view
- Set desired permission
- Save changes



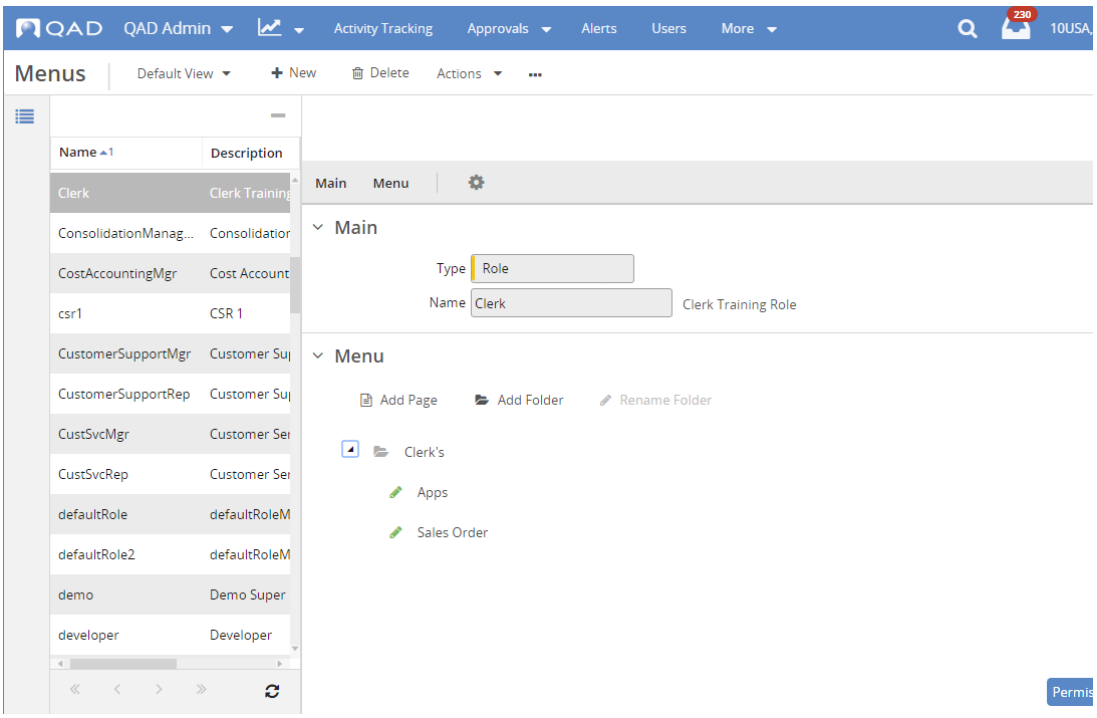
Role Permissions should be granted by a system administrator. There are two other was to launch Role Permissions depending on the context, from the drop down menu:



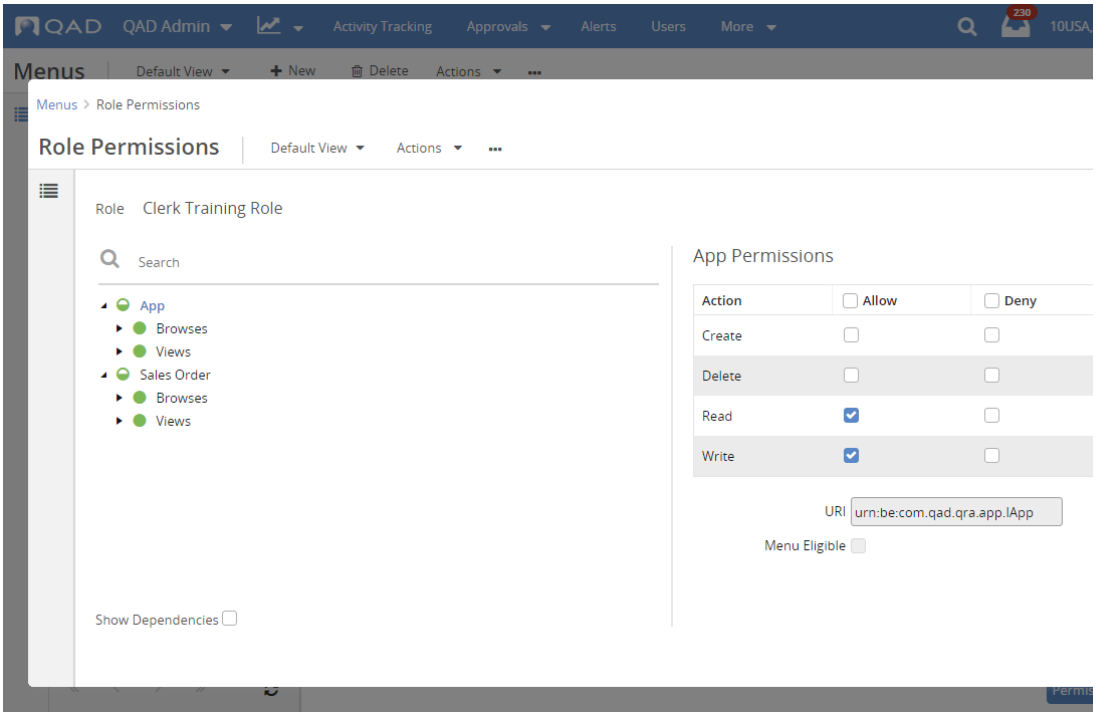
Only the resources in that context will be available to assign permissions



Finally through defining a Role Menu in the "Menus" screen. Click on Permissions

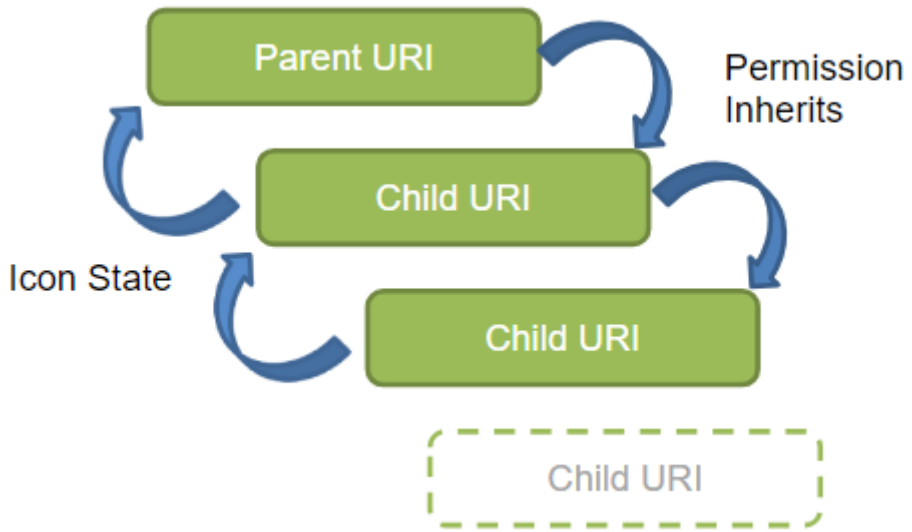


Permissions are only for the items related to the menu

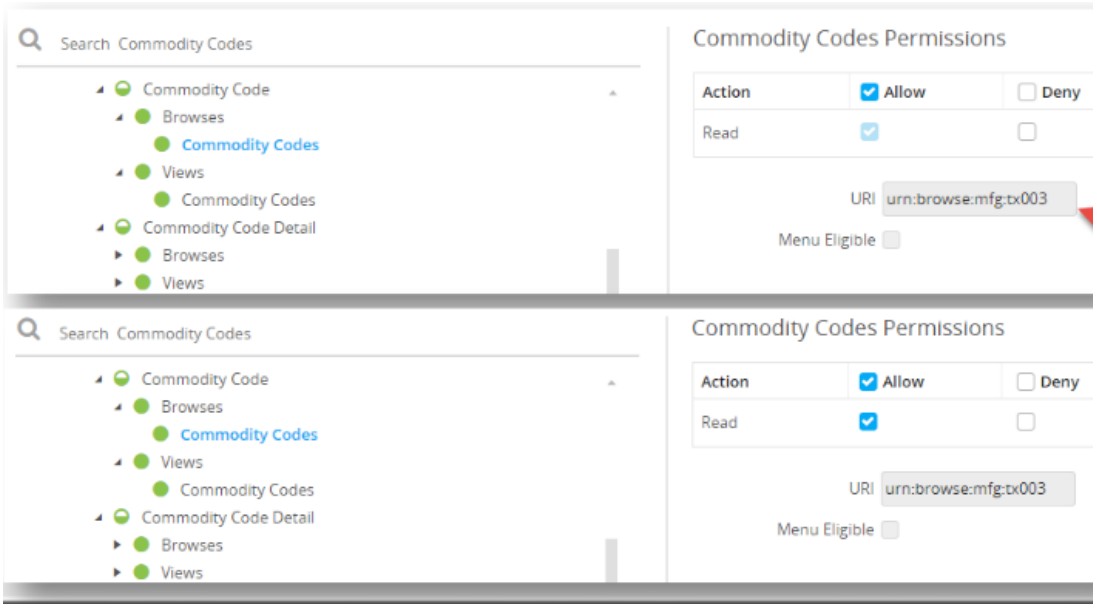


Effective and Explicit Permissions

Permissions assigned at a top level in the permission tree will be inherited by children elements. Effective permissions are determined by traversing the tree upstream to check if the permission is granted upstream (unless a deny permission has been setup). Explicit permission is when a given URI is explicitly granted or denied permissions (see figure above, and notice "Inherited From" column contains "not inherited").



The following example illustrates the effective permission as its inherited from the parent business component. The second permission is equivalent but has been set explicitly to the URI.


Search Commodity Codes
Commodity Code
Browses
Commodity Codes
Views
Commodity Codes
Commodity Code Detail
Browses
Views
Commodity Codes Permissions
Action
Allow
Deny
Read
URI
urn:browse:mfg:tx003
Menu Eligible
Search Commodity Codes
Commodity Code
Browses
Commodity Codes
Views
Commodity Codes
Commodity Code Detail
Browses
Views
Commodity Codes Permissions
Action
Allow
Deny
Read
URI
urn:browse:mfg:tx003
Menu Eligible

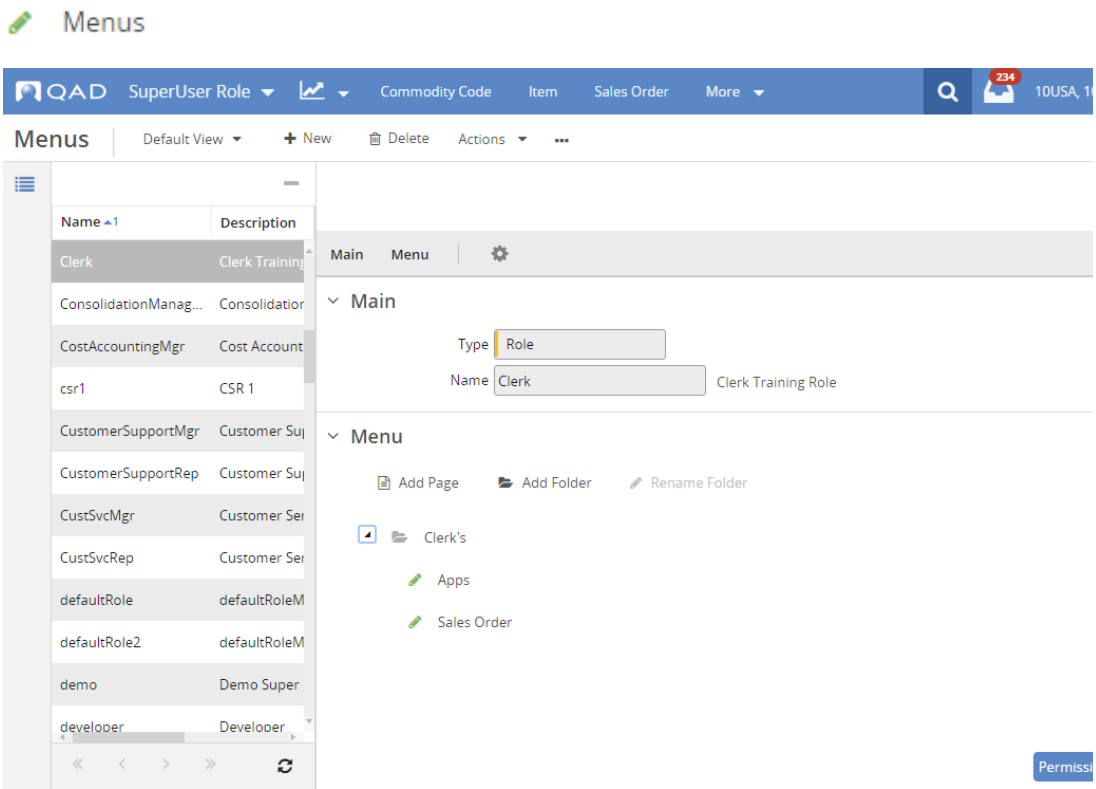
Role Menus

The Menu system in QAD Enterprise Platform has been redesigned and in some cases it consolidates multiple menu entries into a single screen. There are two types of Menus defined, your default role menu and favorites (defined per user).

- Role Menu
- Favorite Menu

QAD Enterprise Platform Applications ship with a default set of role menus

- QAD role menus may be copied but not modified by customers
- QAD role menu permissions MAY be modified
- Development teams are responsible for creating and maintaining the role menus required for their modules



Some important notes about Menus

- Menu navigation linked to a Role
- A user must be a member of the role to see the menu
- Menu items are filtered based on permissions
- Important to align permissions with role menu
- Only resources marked as menu eligible can be added
- Menu eligible resources are identified in Role Permissions
- Menu eligible resources are available in Menus
- Set permissions directly from Menus screen

Creating a Menu

- Make sure a "Role" is available. Or Create a Role by launching "Roles" screen.
- Open "Menus" screen
- Click on New
- Type: Select Role or Favorites depending on the desired menu type
- Click lookup to locate the new Role or type the name in

- If you select an existing Role Menu name then an "Existing Menu" warning will be displayed
- The role name and role menu name must match

Copy an existing Role Menu

A Role Menu can be copied and modified accordingly. These steps highlight how to perform a copy:

1. In Menus, double click on the role to copy. For example "ChiefOperatingOffcr"
2. Click Actions>Copy to copy the "ChiefOperatingOffcr" role menu to the new Role Menu you are defining
    a. Type: Role
    b. Name: Use lookup and select the new Role that you are copying to.
3. Click "Save"
4. Click "Permissions"
5. Click Actions > Allow All Permissions (or selectively grant limited permissions if desired).

Field Security

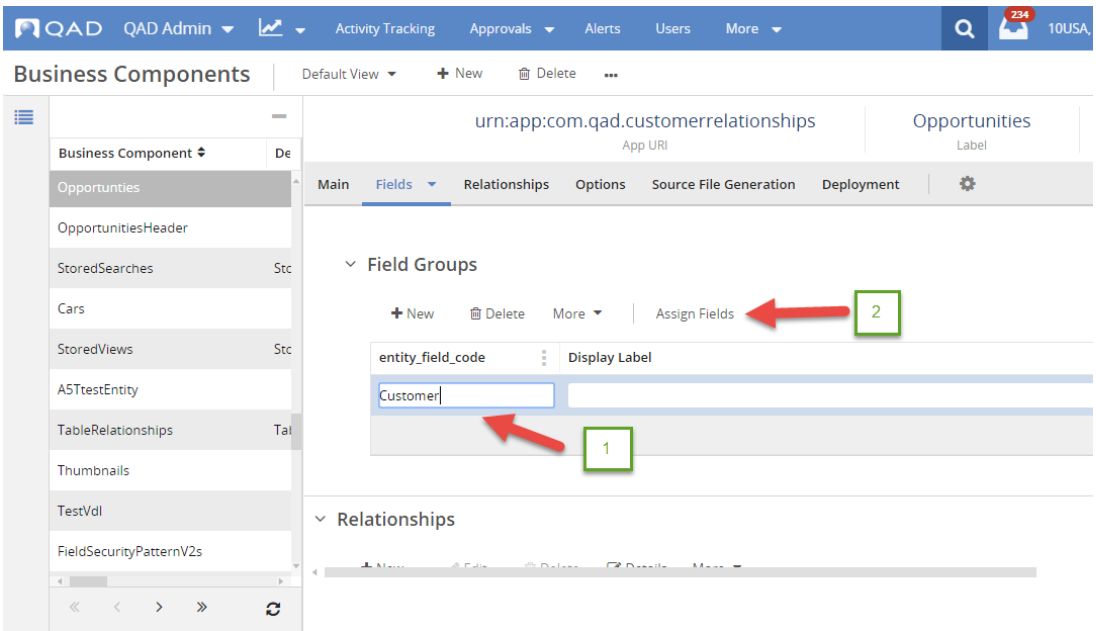
QAD Enterprise Platform Applications makes it possible to control permissions at the field level. Fields are hidden to an end users that don't have permissions to read it. Field security is enforced at the UI and API level. Calling APIs may return null values, developers are expected to handle it and avoid saving "null" values for users that don't have read or write privileges.

Field Groups

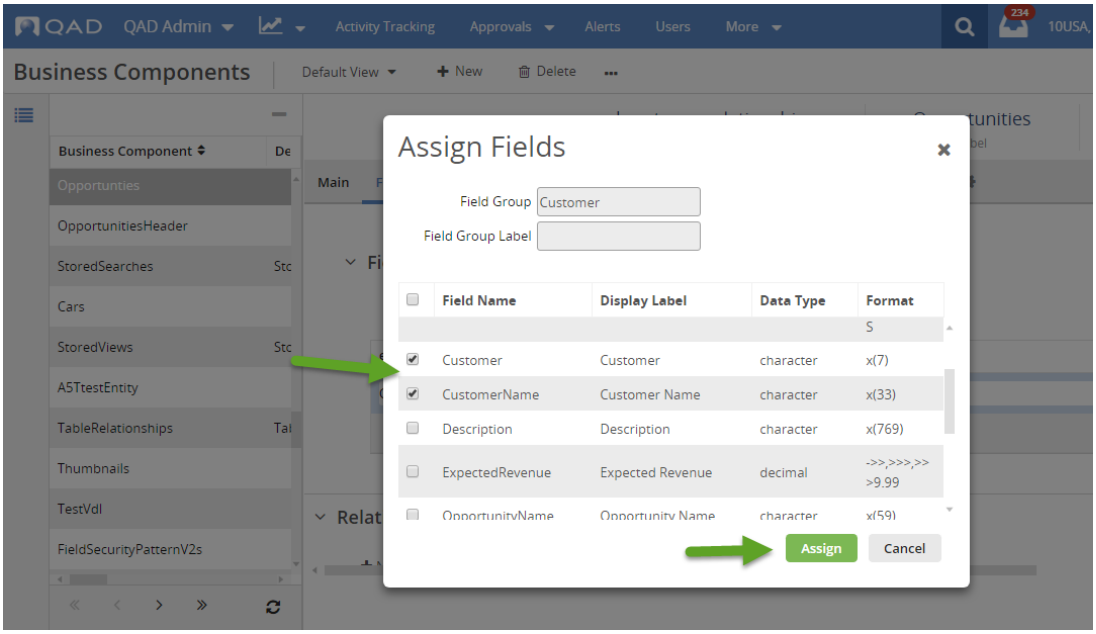
Field Group definitions are not required but are important for system administrators to quickly identify and set permissions to a group of fields. It is important to define "Field Groups" as part of the Business Component definition.

- A field group is a set of fields that logically belong together
- Mostly align with UI panels but not required
- Security can be defined to the group rather than individual fields

- A field group applies permission inheritance to each field
- Developers are responsible for defining appropriate Field Groups
- Customers expect these to be defined for convenience when maintaining field permissions



Then Assign Fields



Then finally save. Permissions can now be set in Role Permissions to these fields or field group.

Deployment

App development can deliver security related settings such as:

- Roles
- Role Menu
- Permissions

Once a developer is done setting up security related configuration the export process will include those settings.

Refer to Platform Development in YAB for process to export App.

Limitations

Introduction

This section provides an overview of the limitations of the platform, based on the current version.

Known limitations

Limitation	Comments
Formula fields can only reference other formula fields that do not contain references to other formula fields.	Formula fields can reference any field from the business component they get defined in, including formula fields. But there is a limitation in that the formula field it references should not reference other formula fields. This is done to prevent performance issues with nested calculations for formula fields.
UI event handler TypeScript code cannot be shared between different business components.	You cannot write classes nor functions that can be called from different event handlers.

Minimizing security risks

To avoid possible security risks with extending Business Components UI Event Handlers or Formulas:

- Separate production and development environments
- Disable all development UI's via security settings

Collaboration Administration

Collaboration administration includes managing activity tracking and alerts.

Activity Tracking

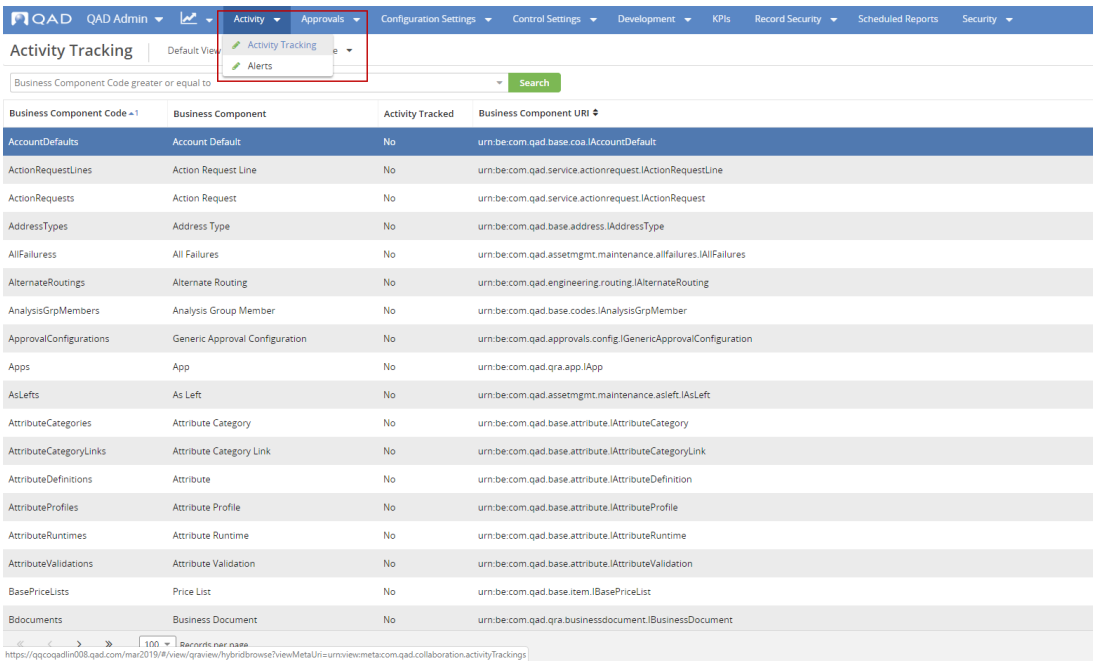
Introduction

In the Activity Tracking panel, you can see a history of transactions and follow the things you care about most. You can comment on activity in a way similar to how you use other kinds of collaborative media. The system can track field changes, comments, and attachment uploads for the current record (such as a field) and displays them in the Activity Tracking panel.

By default, not all field activity can be tracked. As an administrator, you can specify which fields can be tracked using Activity Tracking. From the Activity Tracking view, you can configure what users can track based on the business component. Each business component is listed along with a description, whether the entity is activity-tracked, and then the Business Component URI is included for reference. The Activity Tracked column indicates whether the business component is being tracked (Yes or No).

Example Screen Shots

To open the **Activity Tracking** panel, select the BC and click **Edit**. Or, you can open the panel from a form on the right side of the page by clicking the **Activity** icon.

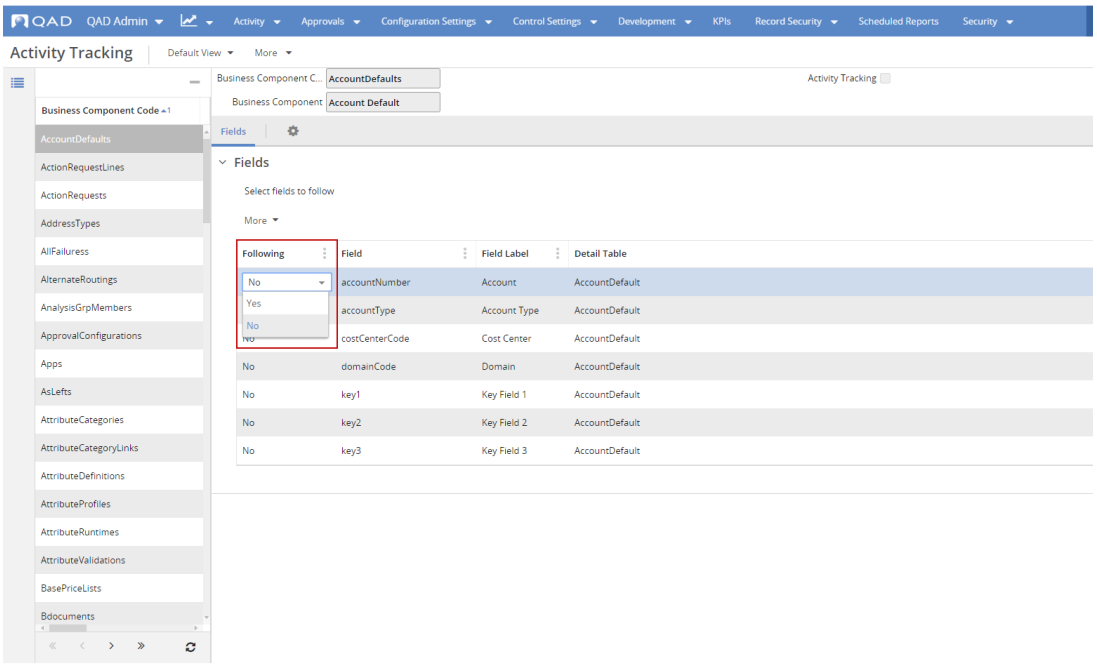


UI Quick Reference

Fields panel

On the **Fields** panel, you can choose which fields of the specific BC you want to track. Select **Yes** or **No** for the field in the **Following** column and save changes.

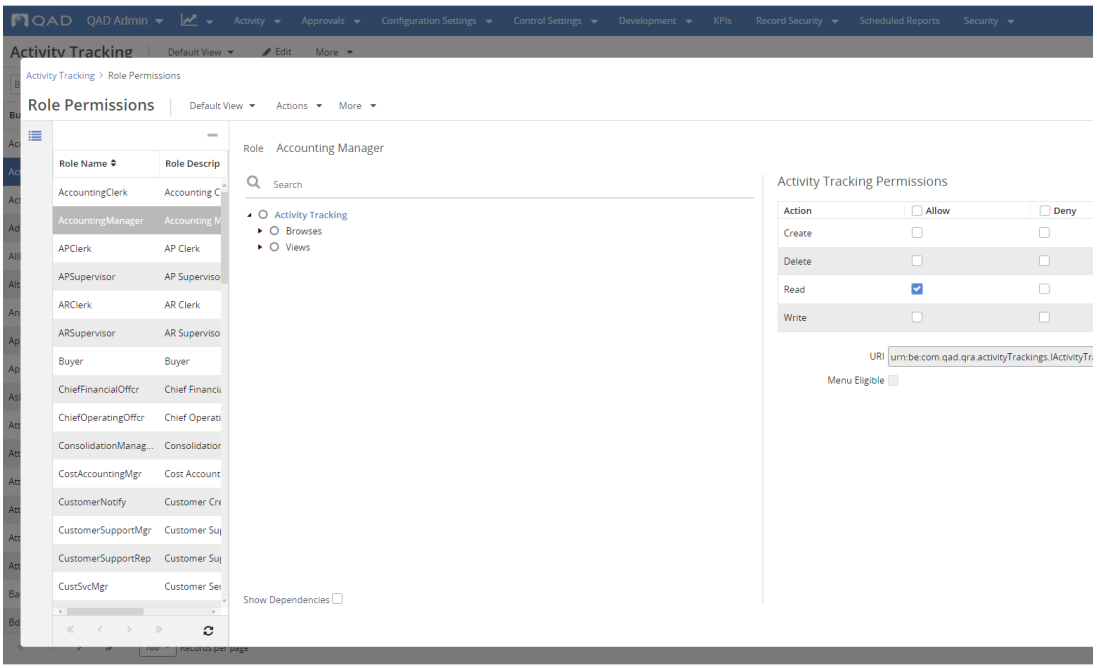
Note that users will only receive notifications on fields they have chosen to follow; these Activity Tracking administration settings only control which fields users can follow.



Setting up Permissions for Activity Tracking

You can set up role permissions for Activity Tracking for the business components fields.

1. On the **Activity Tracking** panel, click the **More** drop-down menu, and then select **Permissions**.
2. Find the role name for which you want to manage permissions and click the **Edit** button.
3. In the **Activity Tracking Permissions** table, choose the needed actions and checkboxes.
4. Click **Save**.



Alerts

Alerts provide the capability to notify users when certain conditions are met. For example, you can create an alert so that if a purchase order's revision number changes, a notification is automatically sent to a user's Inbox or email address.

Alerts can be created by users and by administrators, who can define alerts for users.

Before you can set alerts about fields on a hybrid view form, you must enable activity tracking for those fields on that hybrid view's business component. Activity tracking must be enabled so that the system is following the activity of the fields of interest. For more information about activity tracking, see Activity Tracking.

Users can create their own alerts for hybrid view field activity if activity tracking is enabled for that view's business component. A bell icon in the upper-right corner of the form is displayed if activity tracking is enabled for the business component. A user can click the bell icon to open the Alerts pop-up window for creating alerts.

Users can choose how they receive alert notifications from their user profile settings. Notifications can be sent to the QAD Inbox and the user's email address.

In addition, an Alerts view is available for administrators. Administrators can access the Alerts view to create, modify, or delete alerts, and to add users for specific alerts. In this way, administrators can help users have the alerts they need. For more information on the Alerts view, see Alerts View.

The Alerts pop-up window for users is identical to the Alerts view for administrators, except the Alerts view includes additional settings to select users for alerts.

In the Alerts pop-up windows, users can see both the alerts that they have defined and the alerts defined for them by an administrator. Note that alerts defined by an administrator can only be changed by an administrator.

Creating Alerts - Step by Step

Before you can set alerts about fields on a hybrid view form, you must enable activity tracking for those fields on that hybrid view's business component. For example, on Purchase Orders, to have alerts about changes to the Revision field, activity tracking must be enabled so that the system is following the activity of the Purchase Order Header business component's Revision field. For more information about activity tracking, see Activity Tracking.

As an administrator, to create an alert:

1. Open the Alerts view.
   Note: Users can define alerts by clicking the bell icon located on a hybrid view form whose business component has activity tracking enabled.
2. Click **New**.
3. On the **Main** panel, in the **Alert** field, enter a description of the alert that you want to create.
4. In the Business Component field, select the business component for which you want to define this alert.
5. In the **Conditions** grid, set up the conditions that trigger the alert.
   If you do not set up the conditions, whenever the tracked field is changed, the system delivers an alert notification to users.
   To add a condition, click **New**. To delete an existing condition, choose the record, and then click **Delete**. If you leave either the **Alert** field or the **Business Component** field blank, clicking **New** does not work.
   When setting up a condition, you can click the toggle button to enable field comparison in alert condition definition. When field comparison functionality is on, the **Value 1** field is offered with field options. You can choose a field as the value. For example, for the Item entity, you can define: Description contains Item Type.
   The rules for combinations of conditions are standard. Conditions with different Conditions Fields are joined with the AND operation. Conditions with the same Conditions Fields are joined with the OR operation.
6. Define the alert type.
   a. You can choose to send alerts about changes to fields. See Choosing to Send Alerts about Changes to Fields.
   b. Optionally, you can choose to send alerts when conditions are met. See Choosing to Send Alerts when Conditions are Met.
7. On the **Users** panel, add users to subscribe them to the alert.
   To add a user, click **New**. In the **User ID** field, choose a user ID and the system displays its user name. To delete an existing user, choose the record, and then click **Delete**.

Choosing to Send Alerts about Changes to Fields

If you want the system to send alerts when changes are made to specific fields, choose the **Send alerts about changes to fields** option under the **Conditions** grid. When conditions are met, the system sends alerts that notify users of the changes to the tracked fields.

1. Choose the **Send alerts about changes to fields** option.
   The system displays the **Field Changes** panel.
2. On the **Field Changes** panel, select the fields that you want to track.

Choosing to Send Alerts when Conditions are Met

If you want the system to send alerts when the conditions to trigger the alert are met, choose the **Send alert when conditions are met** option under the **Conditions** grid.

1. Choose the **Send alert when conditions are met** option.
   The system displays the **Message** panel and the **Notification Options** panel.
2. In the **Message** box, define the alert message. You can click **Include Field** to include fields in the alert message. The added fields are automatically added to the end of the text. You can adjust the positions of the added fields manually.
3. On the **Notification Options** panel, configure the **Delivery** setting.
   a. **Immediate (when conditions are first met)**: Let the system deliver alert messages when conditions are first met.
      When you choose this delivery option, the first time the conditions are met, the system sends the defined alert message to users.
      Take price, for example: if the condition is that price is greater than 1,000 and the price value is changed to 1,100 first and then to 1,200, users will only receive one alert about the first change. Under the same condition, if the price is changed to 800 first and then to 1,200, users will also only receive one alert. But this time the alert is to notify users of the price change to 1,200, because the price change to 1,200 is the first time the condition is met.
   b. **Delayed (after conditions are still true)**: Let the system deliver alert messages only when the defined delay time has passed.
      If you choose this delivery option, you are required to specify a time delay in days, hours, and minutes. In this way, you let the system deliver alert messages when the delay time has passed, as long as the business event remains in a triggered state.
      If the alert conditions are not true any more during the delay period, the system does not send the alert message in the end.
   c. **Relative (before or after a variable date)**: Let the system deliver alert messages at a defined time relative to a variable date.
      If you choose this delivery option, you are required to specify the relative conditions.
4. Optionally, on the **Notification Options** panel, select the **Repeat** checkbox to define periodical reminders of alerts to let the system send alert notifications periodically. See Defining Periodical Reminders of Alerts.

Defining Periodical Reminders of Alerts

When you choose to send alerts when the conditions to trigger the alert are met, you can define periodical reminders of alerts at the same time.

On the **Notification Options** panel, specify the Repeat for every days, hours, and minutes as the repetition interval. The system sends alert notifications at specified intervals about the most recent event that happens during each interval, as long as the conditions are satisfied.

For example, you set to trigger the alert when item price is over 1,000 and repeat notifications every two weeks. Now you change the price from 900 to 1,100. The system sends a notification about the price change to 1,100. Then within two weeks, you change the price from 1,100 first to 1,200 and then to 1,400. The system sends a second notification about the price change to 1,400 two weeks after the first notification. If you do not change the price later, every two weeks, the system sends the reminder of the same notification as the second one.

Alerts View

Introduction

With Alerts, you can create and manage alerts for users. The Alerts view is available for administrators. Administrators can access the Alerts view to create, modify, or delete alerts, and to add users for specific alerts. In this way, administrators can help users have the alerts they need.

UI Quick Reference

Main panel

Alert

Enter a name for the alert.

Alert Label

Select a label for the alert name.

Business Component

Select the business component for which you want to define an alert. Only business components with activity tracking enabled are listed.

Conditions

You can have alerts about changes to fields, and alerts for when specific conditions are met. On the Conditions panel, define the conditions to use for getting alerts when conditions are met.

Click **New** to define a condition.

- **Field**: Select a field from the business component.
- **Operator**: Select from the following: equals, does not equal, contains, not contains, range, greater than, greater or equal to, less than, less or equal to, in list, not in list, starts with, ends with, is null, is not null. For operators that require one value, use the Value1 field. For operators that require two values, use the Value1 and Value2 fields.
- **Value1**: Select from available values or click the toggle icon to select available fields.
- **Value2**: For operators that require two values, select from available values or click the toggle icon to select available fields.
- **Send alerts about changes to fields**: From the drop-down, select Send alerts about changes to fields to get notifications about any changes to field values. On the Field Changes panel, select the fields for which you want notifications.
- **Send alert when conditions are met**: From the drop-down, select Send alert when conditions are met to define a message and notification delivery options for when the specified conditions are met. On the Message and Notification Options panels, define the message content and delivery options.

Field Changes panel

This panel is displayed if Send alerts about changes to fields is selected.

In the grid, select the fields for which you want alerts for field value changes. Note that only fields that are enabled for activity tracking are available for alerts.

Message panel

This panel is displayed if Send alert when conditions are met is selected.

Enter the message text. To include field values, click Include Field and select from the business component fields. The field is included in the text in the ${field_name} format, indicating that the field value will be included in the text.

Notification Options panel

This panel is displayed if Send alert when conditions are met is selected.

Delivery

- Select Immediate (when conditions are first met) to have the notification sent as soon as conditions are met. Select Repeat to repeat sending the notification while the conditions are met for the specified Days, Hours, and Minutes.
- Select Delayed (after conditions are still true) to have the message sent after a time delay (in Days, Hours, and Minutes). Select Repeat to repeat sending the notification while the conditions are still true for the specified Days, Hours, and Minutes.
- Select Relative (before or after a variable date) to have the notification sent within a specified time (in Days, Hours, and Minutes) before or after a date provided by selected date field. Select Repeat to repeat sending the notification for the specified Days, Hours, and Minutes.

Repeat

Depending on the Delivery option selected, specify how often to repeat the notification in Days, Hours, and Minutes.

Users panel

An administrator can select which users will receive the notifications for this alert. Once selected to receive alerts by an administrator, a regular user must contact an administrator if they do not want to receive the notifications.

Note: This panel is included on the Alerts view for use by administrators, but is not included in the Alerts pop-up window for regular users.

Click **New** to select active users who will receive this alert.

User ID

Select a user from the lookup.

User Name

Displays the name of the user who will receive alerts.

Analytics

This section provides training materials about QAD Enterprise Platform Analytics.

These are the topics covered:

- Platform Analytics Introduction
- Action Centers Overview
- Create Action Center and Use Action Center
- Create a KPI
- Create Visuals: charts, gauges, tables
- Securing and sharing action centers
- Predefined Action Centers
- Query Service

Also see: Create a KPI based on financial report writer (.pptx download)

Please note that most screenshots are based on the September 2017 version of Action Centers. The overall functionality remains the same but screens might slightly differ with the latest September 2018 release.

Platform Analytics Introduction

Objectives

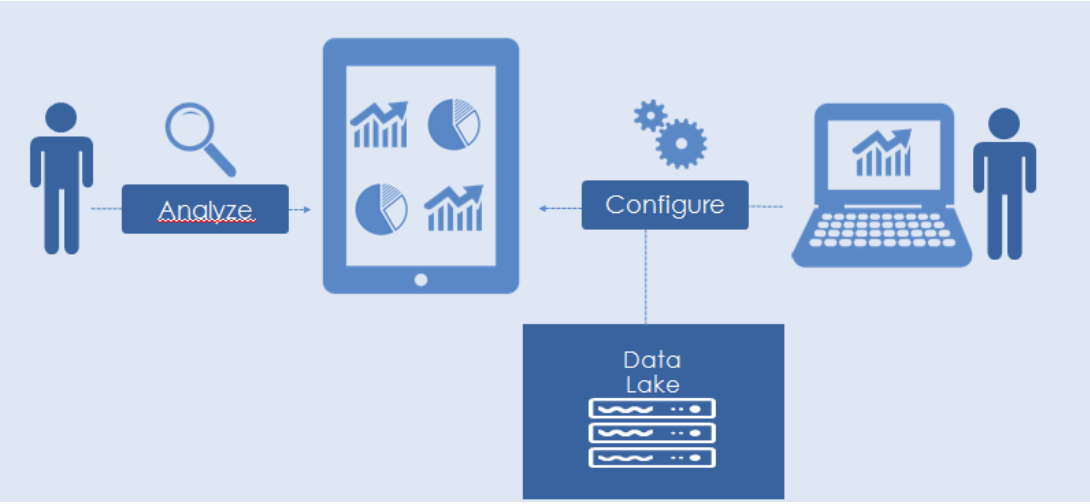
QAD Platform Analytics is QAD's embedded self-service analytics solution at the scale of the enterprise.
It provides managers at all levels of the organization actionable and deep insights in the performance of their business.

The main UX are the Action Center dashboards containing visual Key Performance Indicators (KPI)
From the Action Center the users can further analyse the data and drill down into the underlying details

Roles and processes in analytics

There are two types of users that work with Action Centers.

1. Managers/Decision makers
2. App Builders: Power Users, IT , Services, Partners



The first category are the managers who need visibility on the data going around their business. Managers interact with the Action Center dashboards, analyze the KPIs, slice them by various dimensions, apply filters to focus on segments of the business, drill down to the details and optionally export the details to spreadsheets.

The other category are power users who know the details of the data structures and who can prepare the queries that are the foundation of analytics.

In Action Centers all the legacy standard browse queries and custom browse queries can be used to transform the raw data into powerful KPIs. Identifying and configuring those KPI queries is the task for the power users. To give it a head-start, QAD ships the Action Centers with hundreds of predefined KPIs across all processes of the ERP solution.

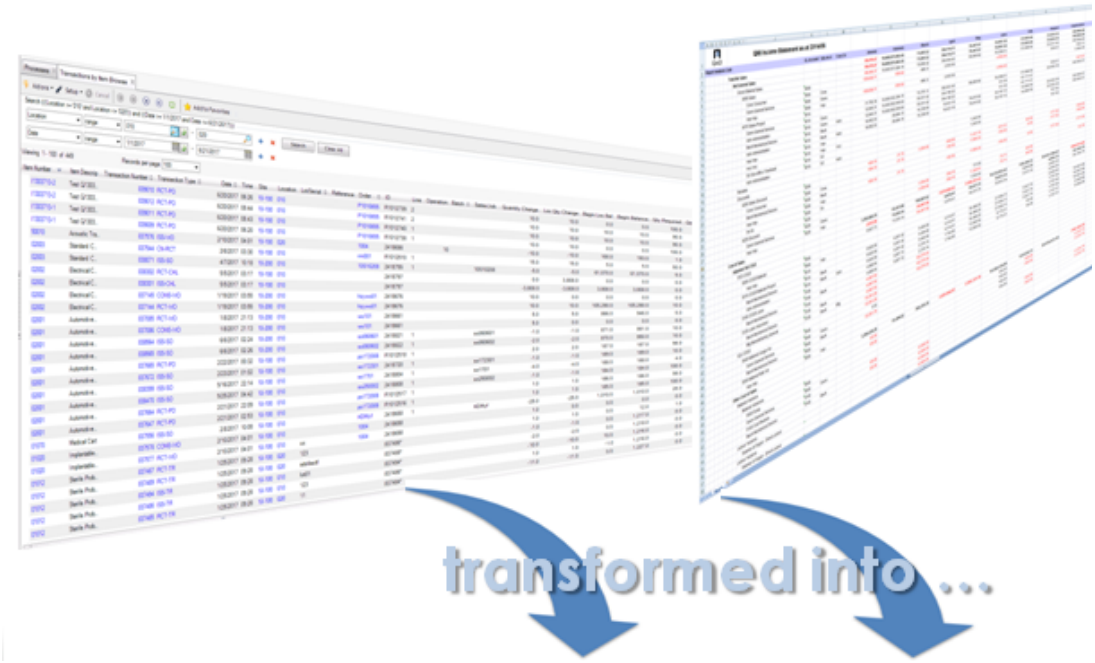
Action Centers Overview

See also demo video and PPT

Main features

Users can create Action Centers and populate those with the KPIs of their choice.

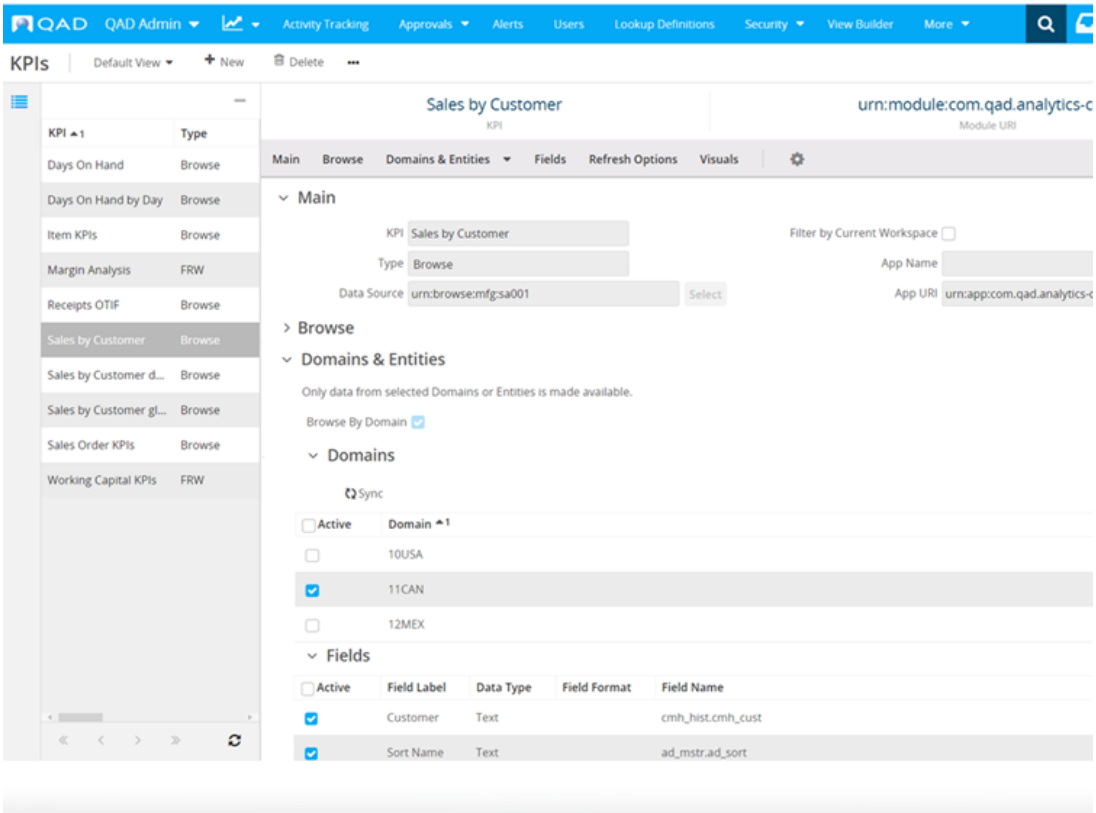
The process starts with the creation of KPIs. Users can transform ANY browse or ANY financial report into a KPI.



This process consists of four steps:

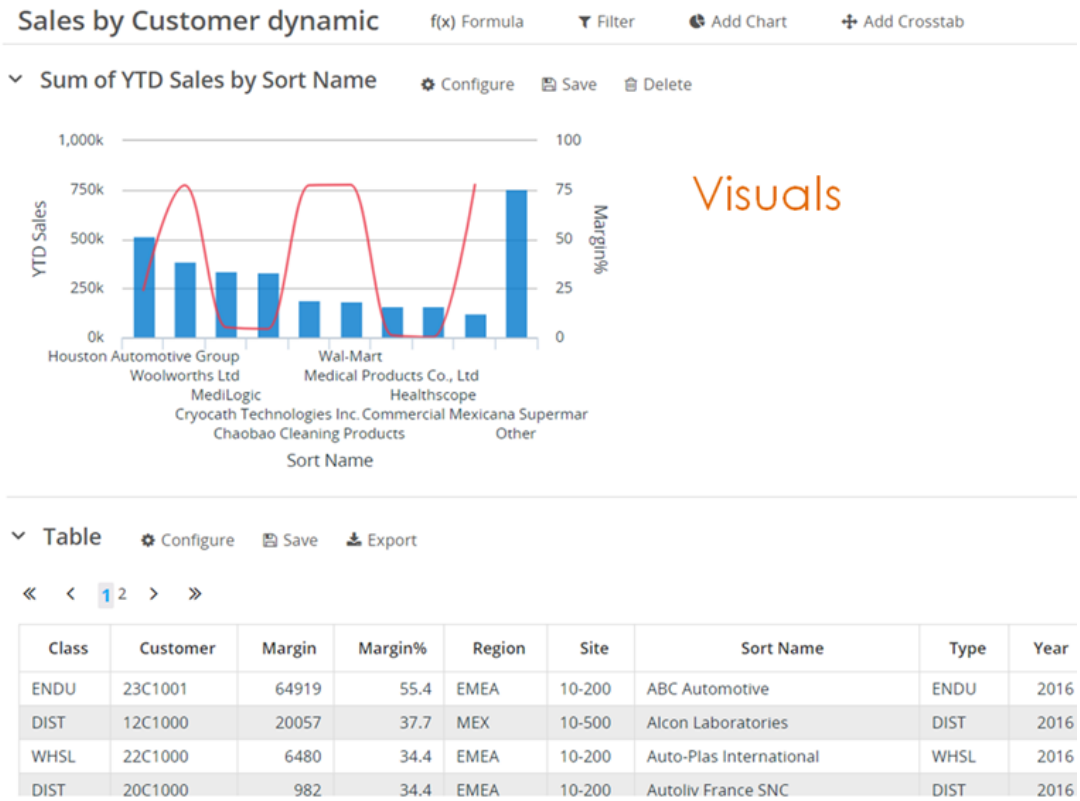
KPI maintenance

Use the KPIs menu to define the KPI. This defines the data source (browse) and the data selection (filter, domains, columns) for the KPI.



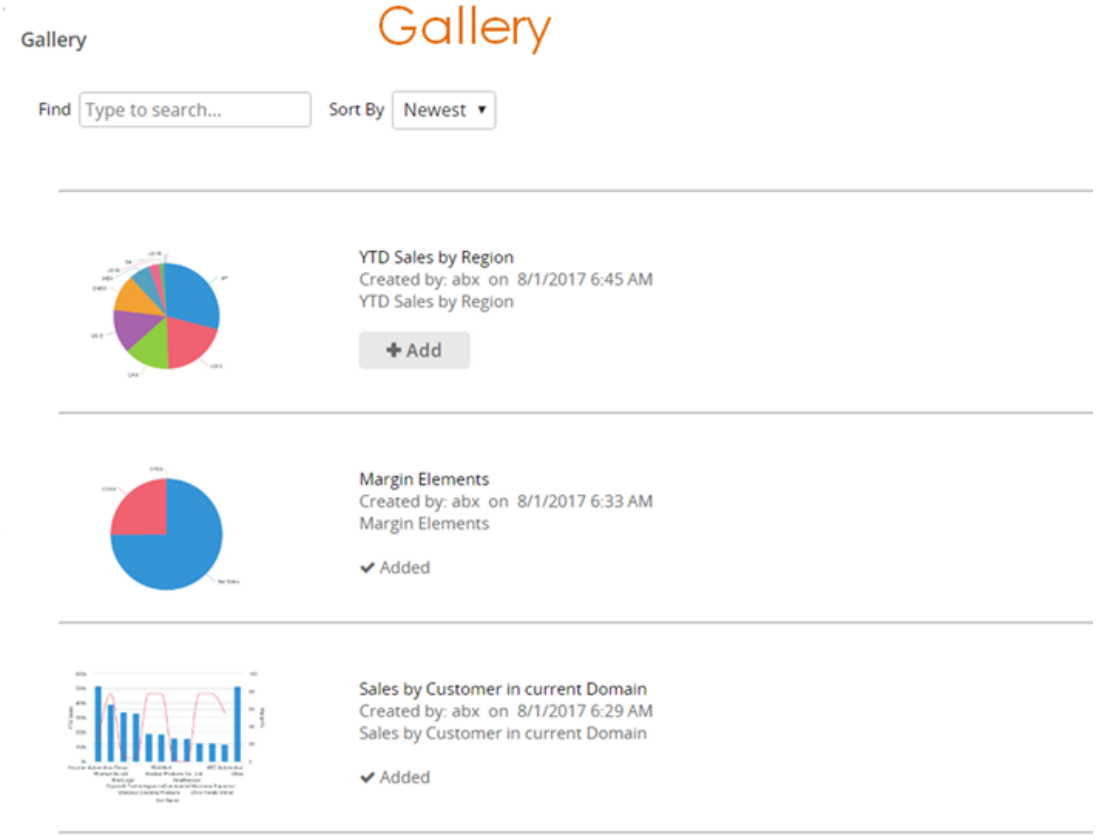
Visuals

From the KPI screen you can proceed to the visual creation (charts, gauges, tables) with calculated columns and filters where needed.



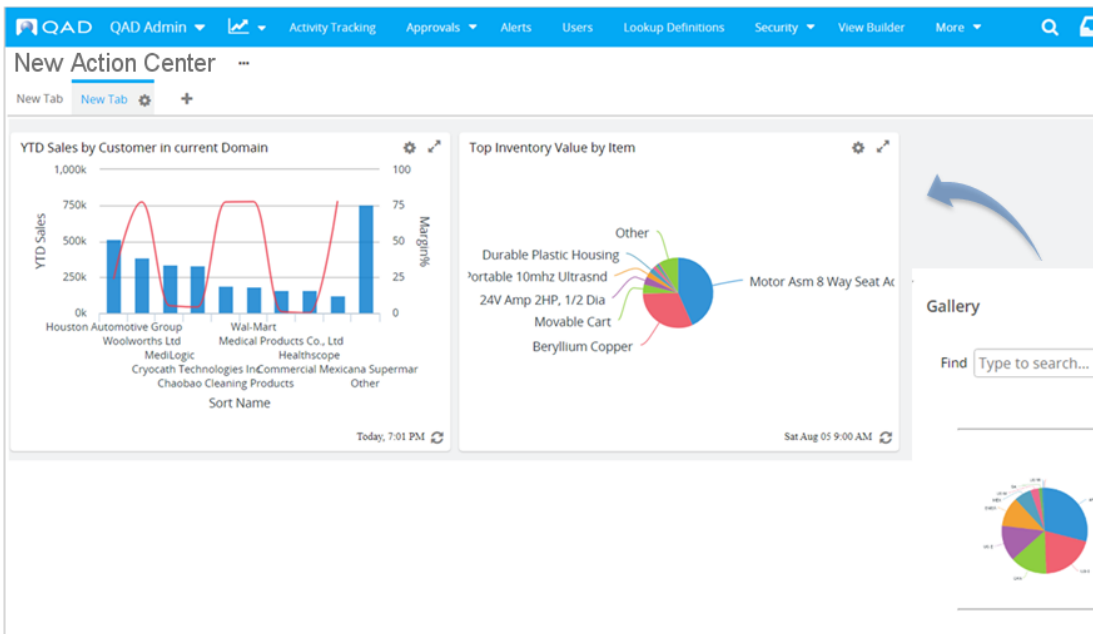
Publish

Visuals are then made available for Action Centers by publishing them in a shared Gallery.



Use in Action Center

The last step is to create an Action Center and populate that by selecting the Visuals from Gallery



Now the Action Center is ready to be used for data analysis. In the next pages we will look further into the details of each steps and many other aspects.

Create Action Center and Use Action Center

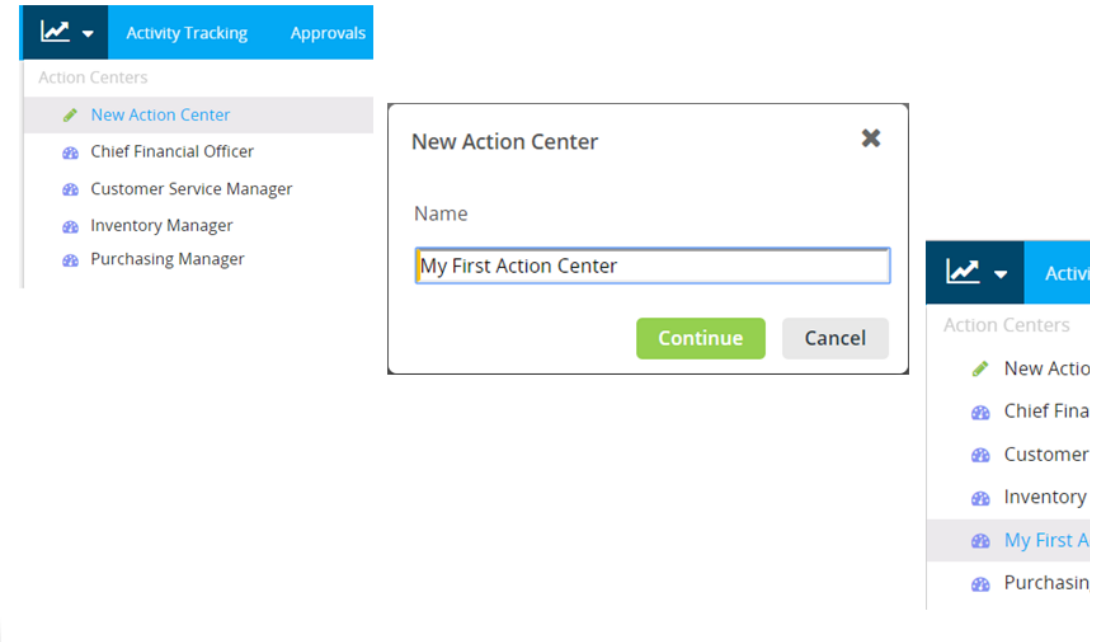
See also demo video and PPT

Create new Action Center

You can create your own action center and the KPIs that you want on it.

From the main menu under the dashboard icon, you select 'New Action Center'.

A dialog box will pop up where you enter the name of the new action center. Click continue and the new action center is added to the menu.



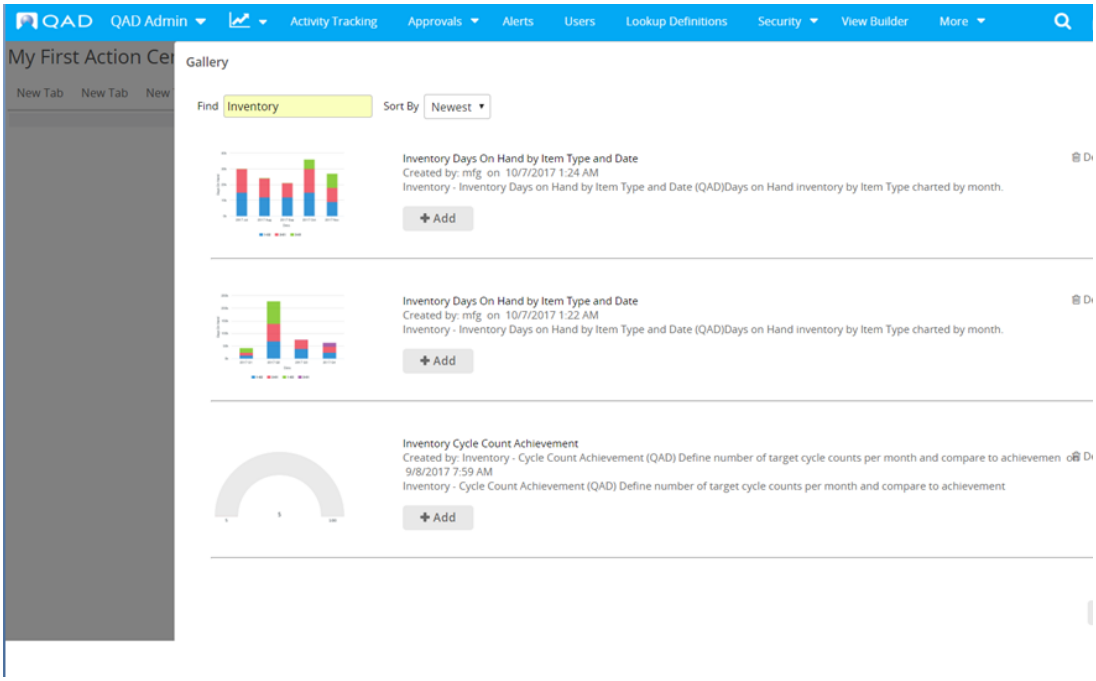
Select KPIs from Gallery

When you open the new action center, it is still empty and a dialog will pop up with a Gallery of all published KPIs.

Assume that you are looking for Inventory KPIs, so you type Inventory in the search box at the top.

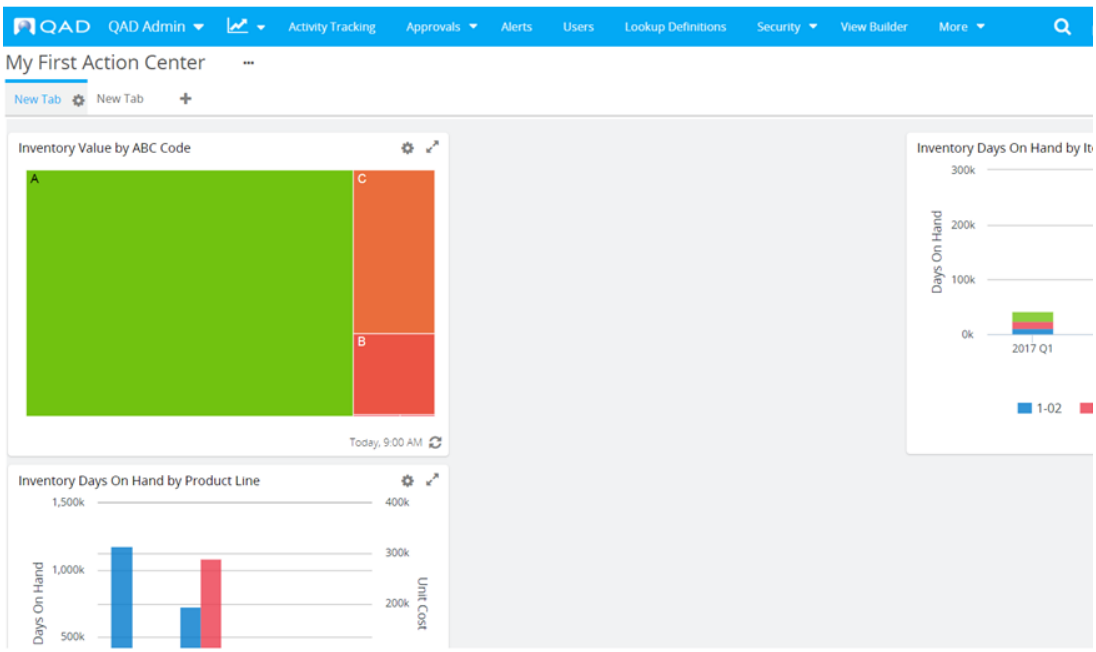
Now all KPIs that have Inventory in their name will show.

By clicking the Add button next to each KPI, you can select the KPIs that you want to add to the action center.



When you complete the selection, click Done and the selected KPIs will be displayed on the action center.

The selected KPI visuals are initially all displayed in the most left part of the screen, stacked upon each other. You can just drag and drop the visuals around the screen to arrange them as you like.



You can also add more tab-pages to the action center. You click the + sign at the top to add more tabs.

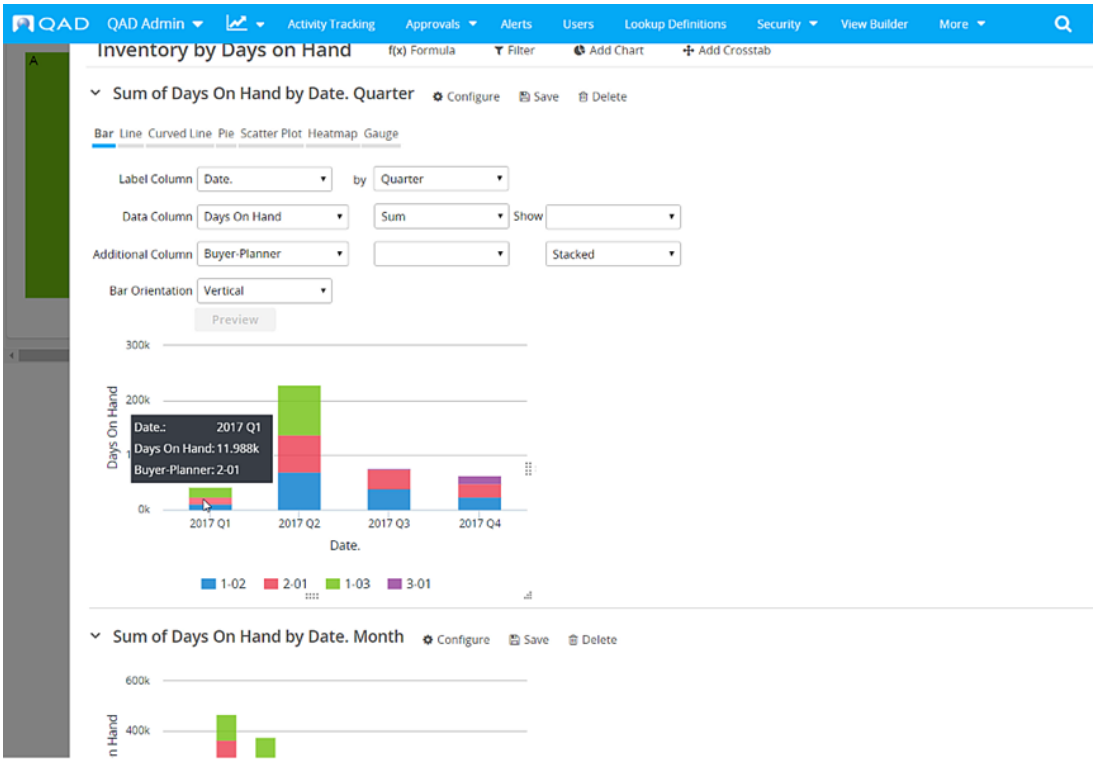
Again the gallery is opened and you can select more of the predefined KPIs. You can add KPIs from the gallery to your action center at any time.

Analyzing a KPI

You can analyze the KPIs in the action center by hovering over the graphs to see the values of the data points.
You can also click on the legend to show and hide the details of the chart.

When you click on the double arrow at the top of the visual, a dialog box opens with the setting details of the visual.
Here you can fine tune the visual. In the example of the screenshot below we change the appearance of the Days On Hand chart and change it into a quarterly chart instead of a monthly chart.
This can be done in the Configure options of the chart. After previewing the chart, you can decide to Save the new chart as a copy next to the original or to replace the original chart with the new one.

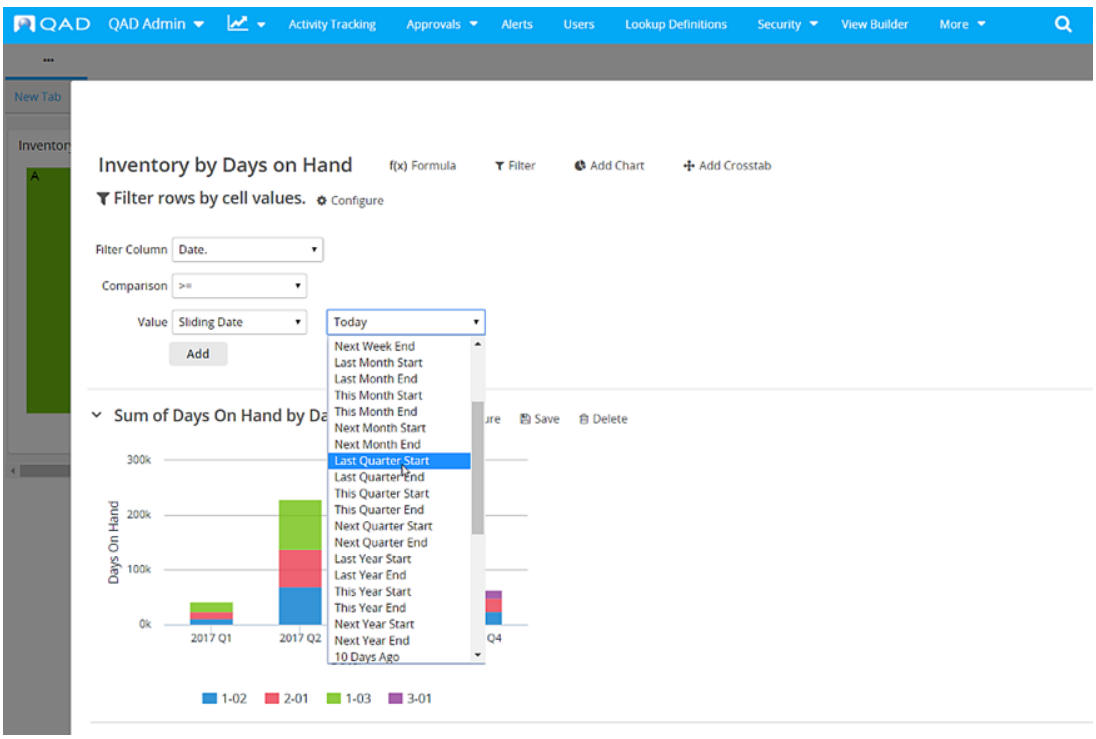
Note that the new version of a visual is always positioned in the upper left corner of the action center. From there you can drag it to the original position or to any other desired position.



Filtering a KPI

In many cases, you want to filter the data of the chart to analyze a specific slice of the data.

Again you expand the visual and in the details dialog you click on the Filter icon at the top.



In this example we will filter on the Date field and use a sliding date. That is a variable date. For example Date is greater than or equal to the “Last Quarter Start”.

With this filter you will always see the data of two quarters: the last (or previous) quarter and the current quarter.

Together with this filter you could for example also change the chart to a Monthly chart, so it shows the months of the last two quarters.

After changing the filter you can again chose to save this new chart on in the action center either replacing the existing chart or as a new chart with a new name.

Related topics

Moving visuals (3 columns)
Renaming Tabs
Renaming Action Center
Removing visuals from action centers
Removing tabs from action centers
Tab order (issue)

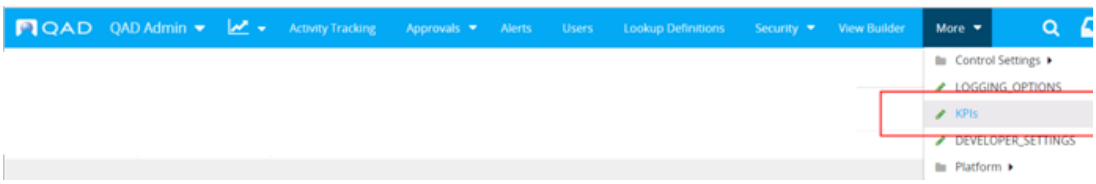
Covered in the Visuals section:
Analyzing the data in the table
Hiding columns, sorting, grouping, aggregates
Exporting the data

Create a KPI

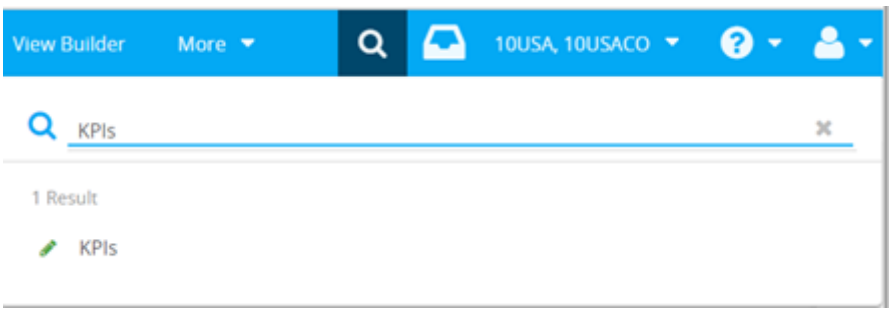
See also demo video and PPT

Menu access

There are two ways to access the KPI maintenance screen. If you have the QAD Admin role then the role menu has KPIs as an option that you can select under the More menu.



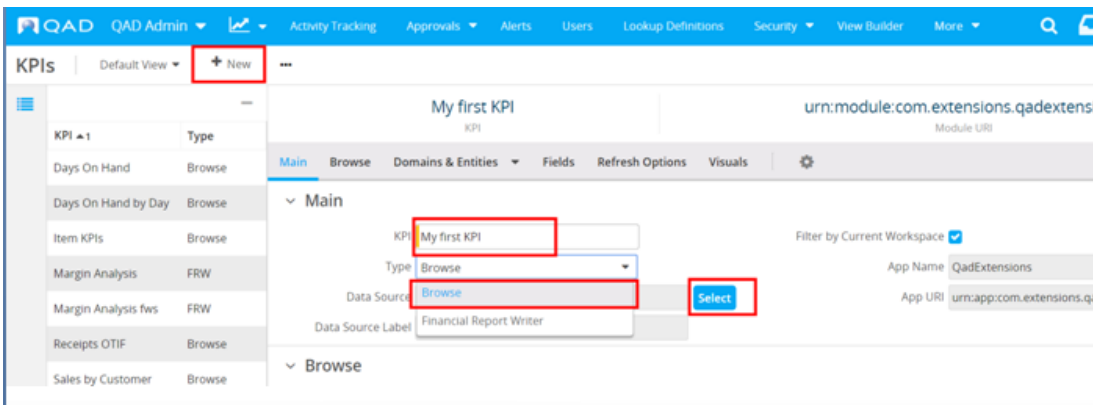
Even if you don't have the QAD Admin menu can still search the KPI maintenance by typing KPIs in the menu Search.



Main section

When the KPIs screen opens, you will see a list of existing KPIs in a browse on the left hand side. Clicking on any of these existing KPIs bring up the settings of that KPI in a maintenance form on the right hand side.

If you want to create a new KPI then click the +New icon at the top of the screen. A new empty KPI form opens.



In the KPI field on Main panel you enter a meaningful name for the new KPI.

For the Type, you can choose between Browse and Financial Report Writer. The type indicates the type of data source that will be used for the KPI.

In this example we choose Browse. Another page explains the Financial Report Writer type that has slightly different settings.

After selecting the Type = Browse, click the blue Select button.

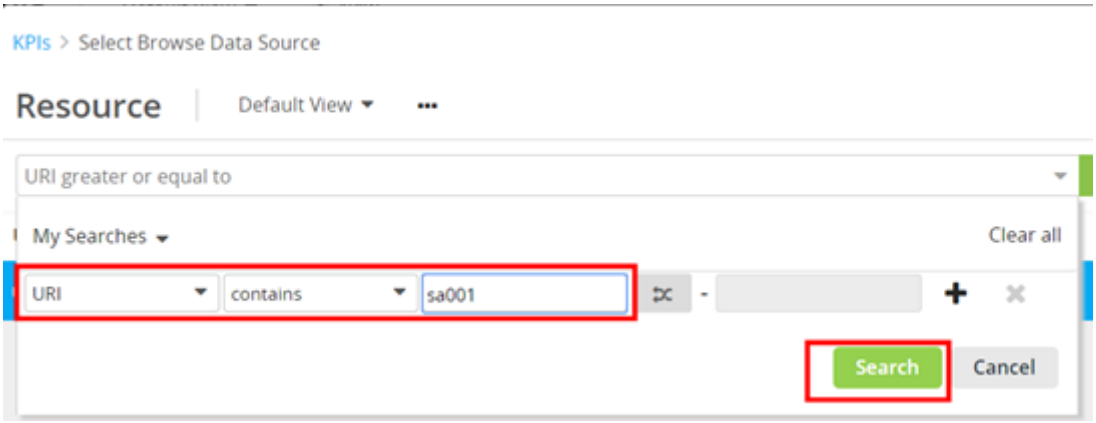
Selecting a browse

Clicking the Select button at the right hand side of the Data Source field opens a dialog box with a list of Browses. That is the list of ALL browses that exist in the ERP system and that were made available in the Channel Islands webUI.

You can search a browse by its name in the menu or you can also search by the URI.

For example, you can search a browse with Name Equals “Sales by Customer”. This returns the ‘Sales by Customer’ browse.

If you know the browse code, for example “sa001”, then you can search it with "URI Contains sa001". This also returns the ‘Sales by Customer’ browse.

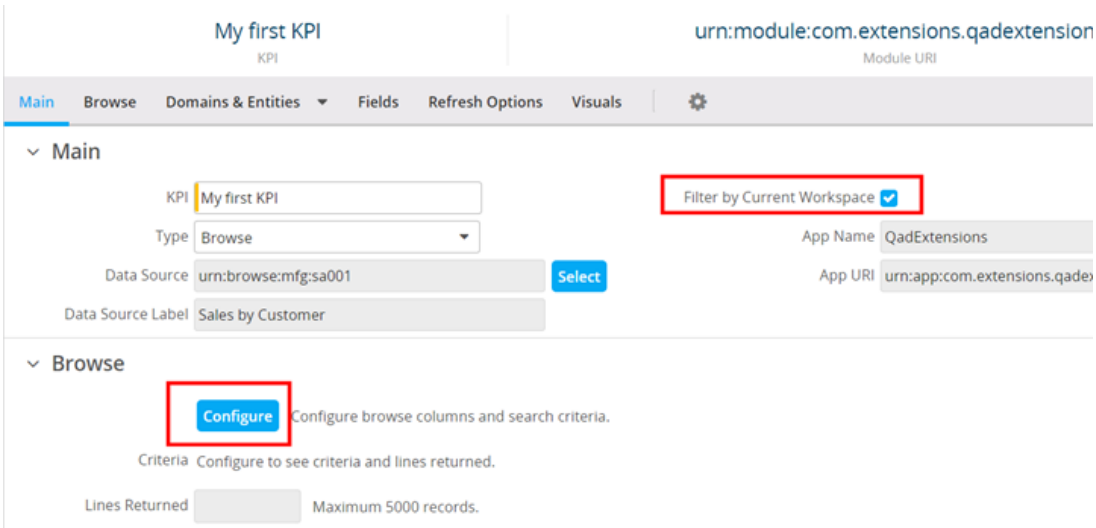


Once you located the browse, you select it and click OK.

This brings you back to the KPI screen and shows the selected browse.

Configuring the browse

Note that there is a checkbox on the right hand top of the form labeled Filter by Current Workspace.
If this is checked, it means the KPI will read the data according to the workspace that the user is logged on to. So typically you will see the data for the current domain (or entity for financials KPIs). When the box is unchecked, the KPI can show the data of multiple domains and entities at the same time. Which domains you will get data from depends on the domains or entities selection list further on in this screen and it will also be constrained by your user role permissions to read data of the selected browse. There is a separate page that explains the security for action centers in more detail.

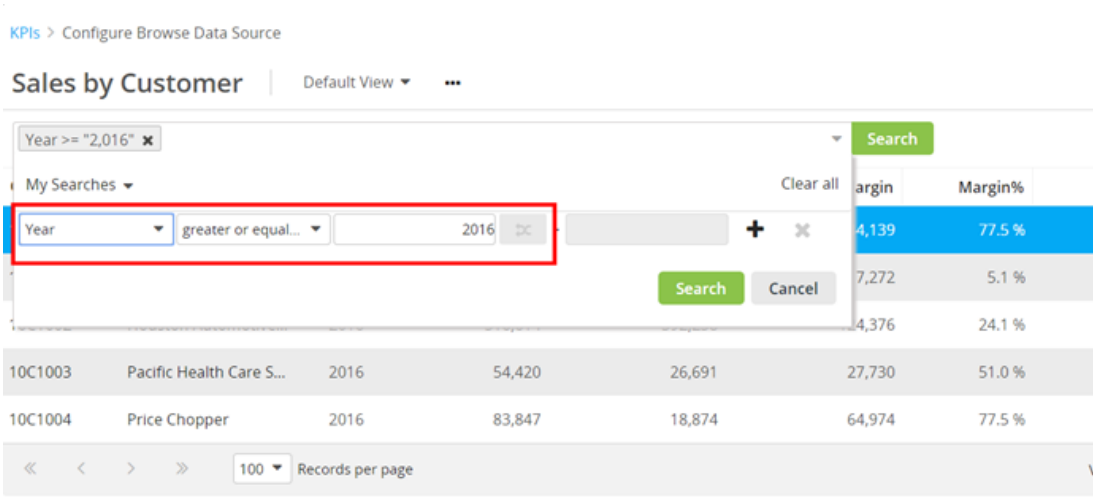


Next optional step is to configure the browse columns and search criteria. Search criteria are important to limit the number of records that is returned from the browse.

Therefore you click the Configure button. This opens the actual browse that you selected earlier.

In this example the Sales by Customer browse. If for example you only want to see the data of last year and this year, you enter a search condition Year greater or equal than 2017.

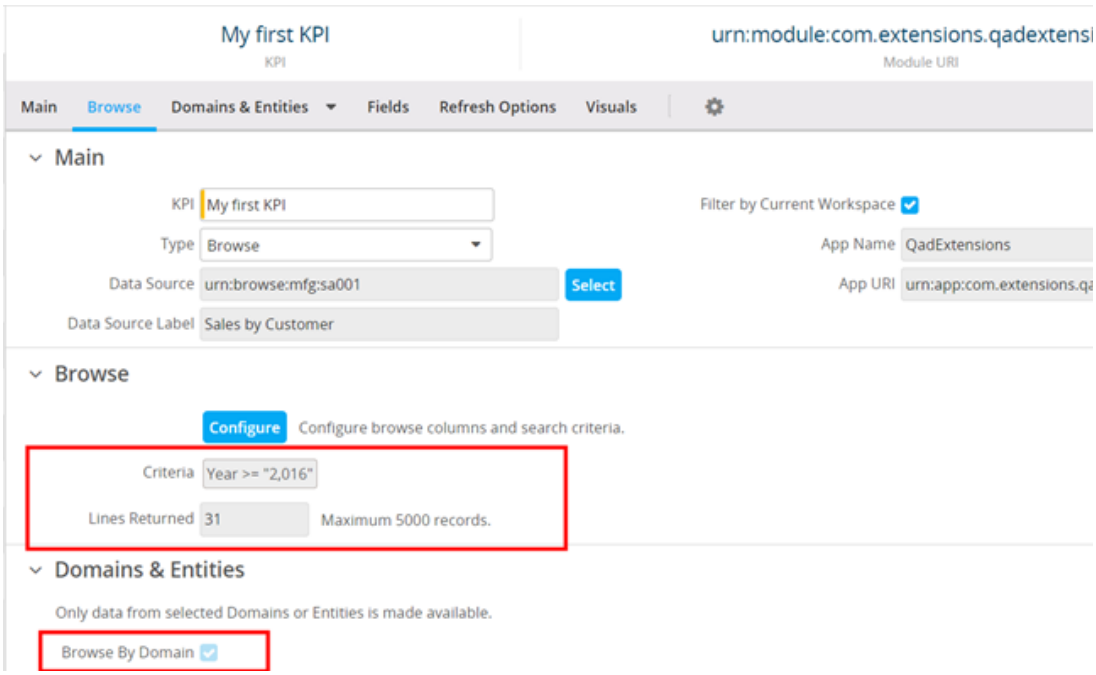
Note that if the search field is a date, you can also use a sliding date like ‘This year’ ‘This quarter’ etc.



Click Search to verify that the browse returns the expected records. Then click OK.

This brings you back to the KPI screen and shows the search conditions and the number of records that it currently returns.

You can always change the search criteria later.

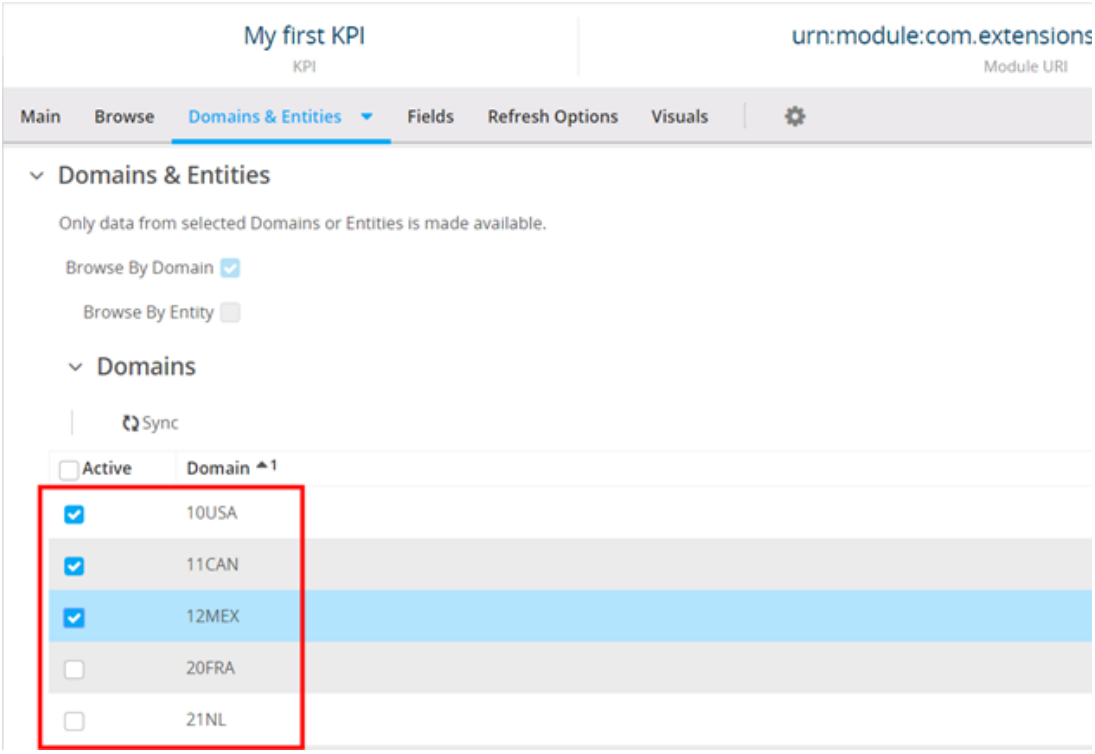


Select Domains

In the Domains and Entities section the Browse By Domain checkbox is checked. This field is read-only and for information purpose only.

It means that the selected browse always returns data per domain. So if you have set the Filter by Current Workspace checked, you will see the results of the current domain, one domain at a time. If Filter by Current Workspace is unchecked, then the browse will read the combined data from all domains selected in the domains list. That list allows selecting the domains that this KPI retrieves data from. You must select the extensive list of all domains that all users of this KPI can need.

The smaller the selection, the faster the KPI will read the data. So make sure that you are not selecting domains that are not expected to be used in the data analysis. You can always change the selection of the domains later.

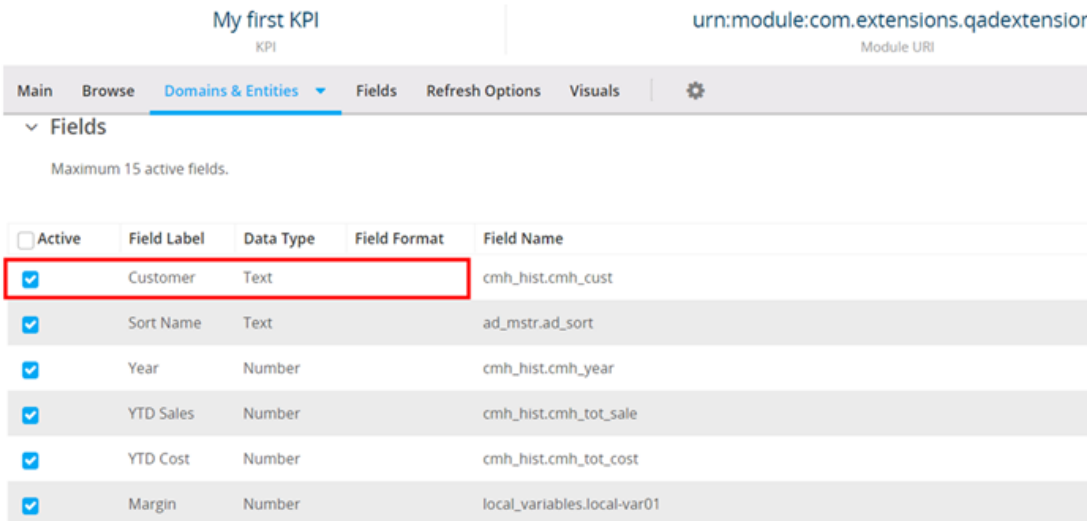


Select Fields

In the Fields section there is a grid showing all the fields that are returned from the browse. Here you select the fields that you need for the KPI by checking the boxes in the Active column.
It is important that you only select those fields that you need for the KPI data analysis. That typically are the key dimension fields like customer, supplier, item, the amounts and quantities that you want to measure and the time dimensions, transaction dates, due dates, etc.

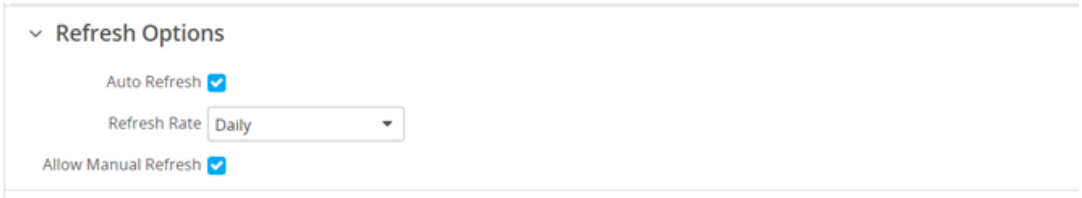
20 is the recommended maximum number of fields that you can select for a KPI. This is for performance reasons. If you need more fields then you are probably looking for more than a single KPI to analyze and you can create two or more KPIs for the same browse as data source.
In the grid you can also change the field labels if that is needed. For Date and Number type fields you can also set the Field Format. That is a display mask used for the fields in the visuals.
Important: while you can change the field selection at any time later, doing so will break KPI visuals that you have created in the mean time and you will have to recreate the visuals. So it is important to carefully make your selection of fields.

Data Type and the original browse Field Name columns are read-only and for information purpose only.



Refresh Options

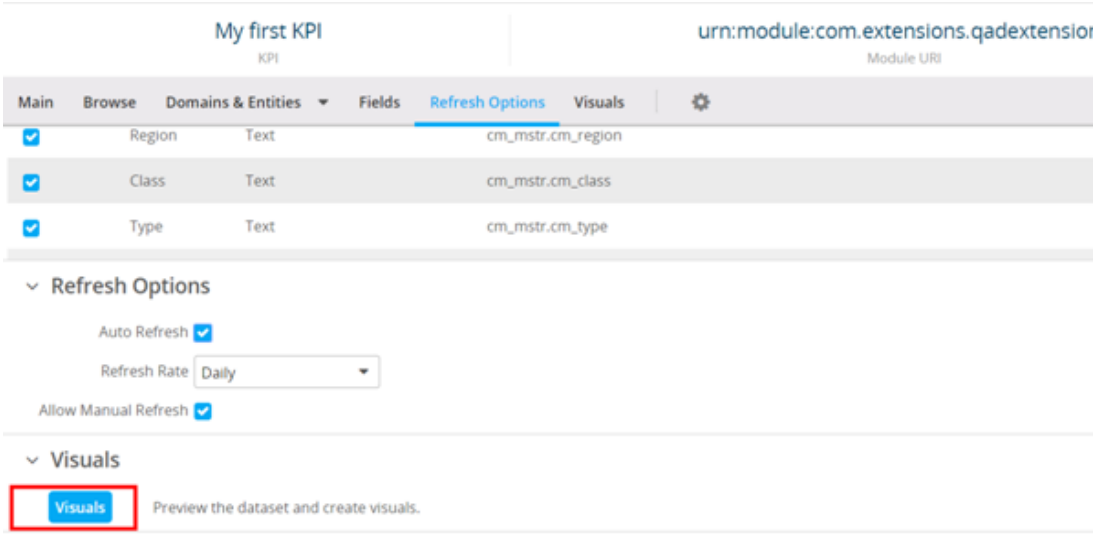
In the Refresh options section, you can indicate the frequency of automatic refresh of the data for the KPI. That can be daily, weekly or monthly.
If you leave the Allow Manual Refresh checked then users can refresh the data on the spot at any time so they get the real time data.
Since the refreshed data will be visual for all users of the KPI, in some scenarios you may not want users to do such a manual refresh so that all users are seeing the same KPI values during the day and have the KPIs refreshed automatically overnight.



Data refresh is discussed in more detail further on.

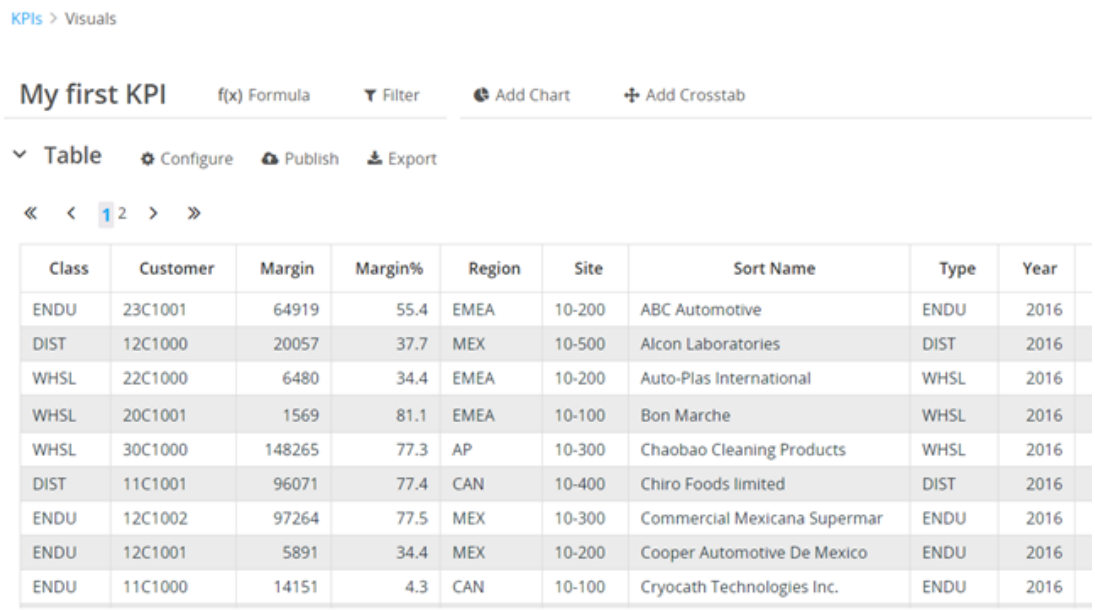
Save and Visuals button

Now we have completed the KPI definition and we can save it by clicking the Save button.
This also enables the blue Visuals button.



Visuals Preview

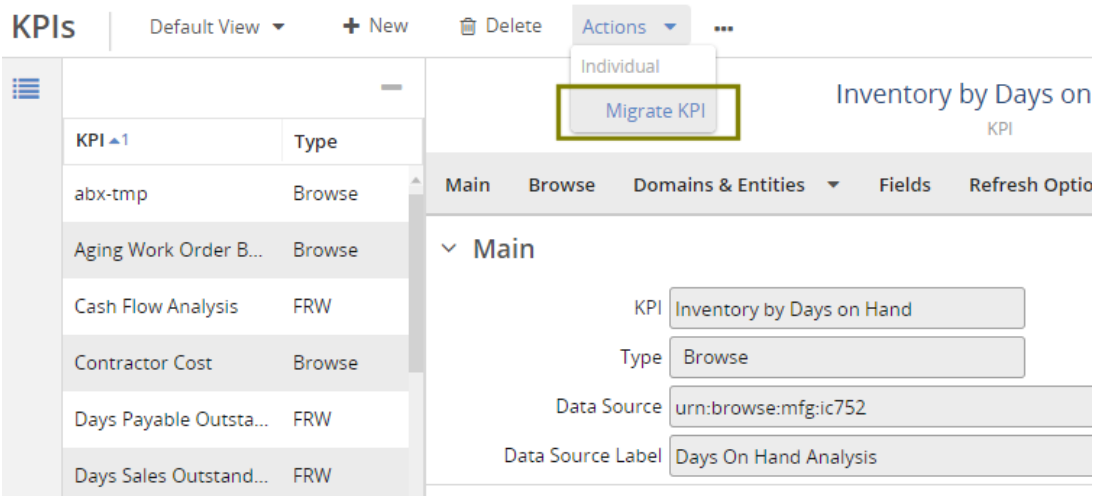
Clinking the Visuals button brings you to the Visuals screen where you create the graphical representations of the KPI : charts, gauges, tables, etc. The creation of visuals is explained on another page.
Note that there is always a table available on this screen that shows the data returned from the KPI data source (in this case the Sales by Customer browse).



Class	Customer	Margin	Margin%	Region	Site	Sort Name	Type	Year
ENDU	23C1001	64919	55.4	EMEA	10-200	ABC Automotive	ENDU	2016
DIST	12C1000	20057	37.7	MEX	10-500	Alcon Laboratories	DIST	2016
WHSL	22C1000	6480	34.4	EMEA	10-200	Auto-Plas International	WHSL	2016
WHSL	20C1001	1569	81.1	EMEA	10-100	Bon Marche	WHSL	2016
WHSL	30C1000	148265	77.3	AP	10-300	Chaobao Cleaning Products	WHSL	2016
DIST	11C1001	96071	77.4	CAN	10-400	Chiro Foods limited	DIST	2016
ENDU	12C1002	97264	77.5	MEX	10-300	Commercial Mexicana Supermar	ENDU	2016
ENDU	12C1001	5891	34.4	MEX	10-200	Cooper Automotive De Mexico	ENDU	2016
ENDU	11C1000	14151	4.3	CAN	10-100	Cryocath Technologies Inc.	ENDU	2016

Migrate KPI

This option was added in the September 2018 release.
If you want to migrate a KPI from one environment to another, the first select the KPI from the browse. Then select the Migrate KPI option from the menu. You will be prompted to select Export or Import.



For Export you can specify the .zip file name that will be downloaded in your web browser when you click OK.
For Import you can select the .zip file from your local drive that will be uploaded, and the corresponding KPI created / updated, when you click OK.

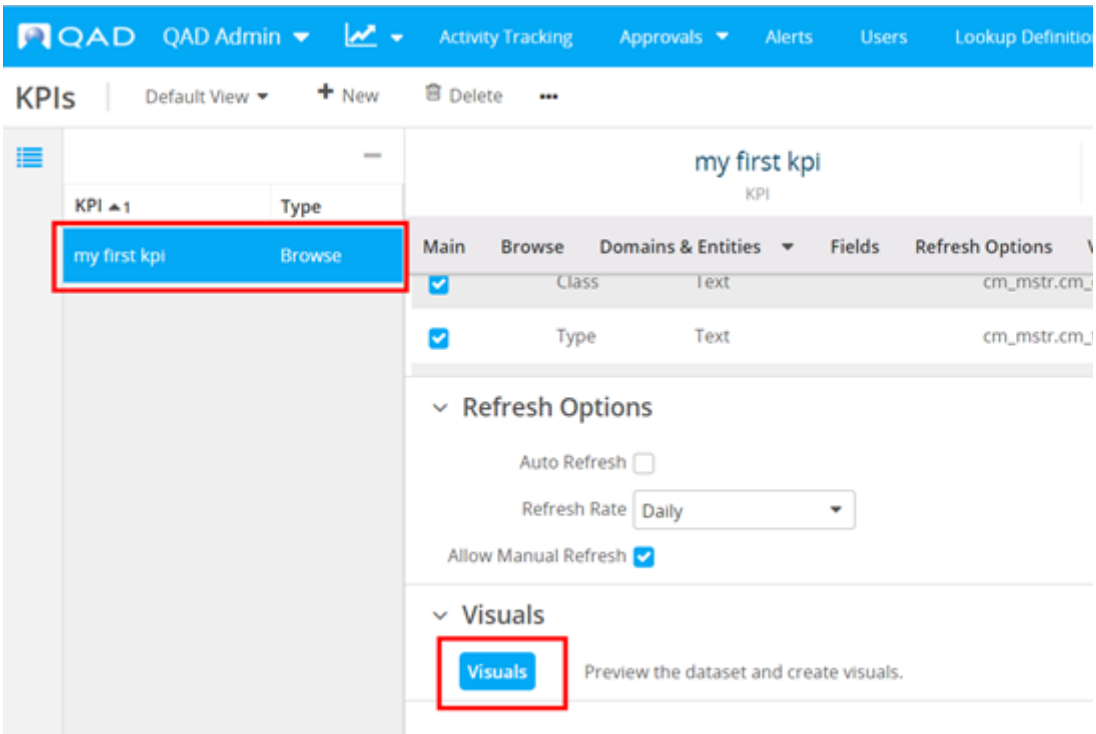
Create Visuals: charts, gauges, tables

See also demo video and PPT

Accessing the Visuals page

KPI visuals are visual representations of a KPI in the form of a chart, table or a cross-tab that you can add on an Action Center.

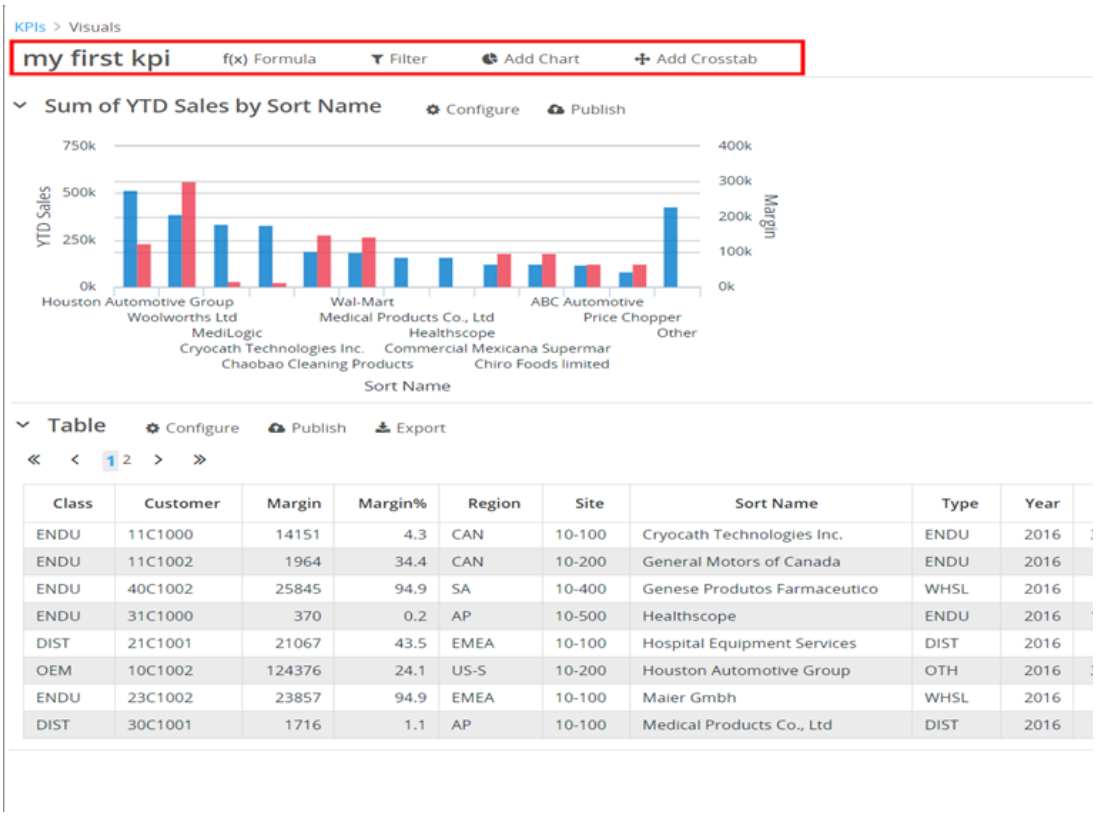
You can create visuals starting from the KPI maintenance screen and you can also create visuals starting from the Action Center itself.
In a first example we start from a KPI in the KPI maintenance screen. On another page it was explained how you create a KPI in that maintenance screen.
Now we select a KPI from the browse and go straight to the Visuals button at the bottom of the screen.



Visuals Header section

The Visuals page consists of header section with five elements.
•At the top is a read-only name of the KPI
•Next to it is the Formula icon that gives access to a Formula panel for adding columns to the dataset and making

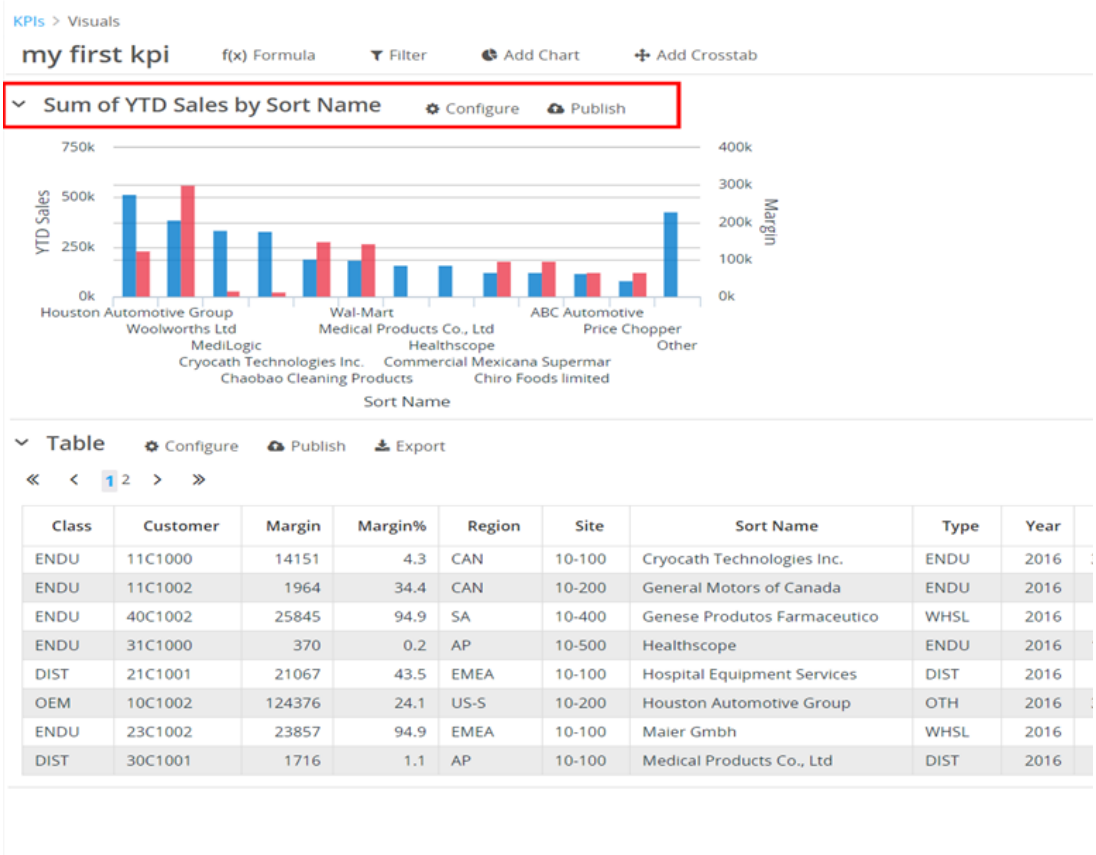
calculations with data from other columns
•Next to it is the Filter icon for specifying filter criteria to narrow down the data used for the visuals. The filter is applied on all the visuals.
•The Add Chart icon is for adding new charts on a visual page. Here we see a chart that was already added during a previous visit. Many charts can be added.
•The Add Cross-tab icon is for adding a cross-tab summary of the data.
•Group/Ungroup As of the September 2018 release there is an icon Group/Ungroup added. This allows to retrieve the data in an aggregated mode or in full detail. With large data sets the full data can only be charted when it's retrieved in Grouped mode (more details around this you can find in the Query Service section of the documentation.



Class	Customer	Margin	Margin%	Region	Site	Sort Name	Type	Year
ENDU	11C1000	14151	4.3	CAN	10-100	Cryocath Technologies Inc.	ENDU	2016
ENDU	11C1002	1964	34.4	CAN	10-200	General Motors of Canada	ENDU	2016
ENDU	40C1002	25845	94.9	SA	10-400	Genese Produtos Farmaceutico	WHSL	2016
ENDU	31C1000	370	0.2	AP	10-500	Healthscope	ENDU	2016
DIST	21C1001	21067	43.5	EMEA	10-100	Hospital Equipment Services	DIST	2016
OEM	10C1002	124376	24.1	US-S	10-200	Houston Automotive Group	OTH	2016
ENDU	23C1002	23857	94.9	EMEA	10-100	Maier Gmbh	WHSL	2016
DIST	30C1001	1716	1.1	AP	10-100	Medical Products Co., Ltd	DIST	2016

Charts section

Below the header section there are charts. Initially there are no charts here, but new charts are added each time you click the Add Chart button. So on a single visuals screen there can be multiple charts. Each of these charts has three elements above it.
•The first element is the chart name. This is read-only text and contains a proposed name for the chart. You can change this name when you publish the chart.
•Next to the chart name we have the Configure icon. Clicking this icon opens the settings that determine the type and the content of the chart.
•Next to the Configure we have the Publish icon. Once you are happy with the chart that you configured, you can publish it in a shared Gallery, so that it can be used on the action centers.



Class	Customer	Margin	Margin%	Region	Site	Sort Name	Type	Year
ENDU	11C1000	14151	4.3	CAN	10-100	Cryocath Technologies Inc.	ENDU	2016
ENDU	11C1002	1964	34.4	CAN	10-200	General Motors of Canada	ENDU	2016
ENDU	40C1002	25845	94.9	SA	10-400	Genese Produtos Farmaceutico	WHSL	2016
ENDU	31C1000	370	0.2	AP	10-500	Healthscope	ENDU	2016
DIST	21C1001	21067	43.5	EMEA	10-100	Hospital Equipment Services	DIST	2016
OEM	10C1002	124376	24.1	US-S	10-200	Houston Automotive Group	OTH	2016
ENDU	23C1002	23857	94.9	EMEA	10-100	Maier Gmbh	WHSL	2016
DIST	30C1001	1716	1.1	AP	10-100	Medical Products Co., Ltd	DIST	2016

Table Section

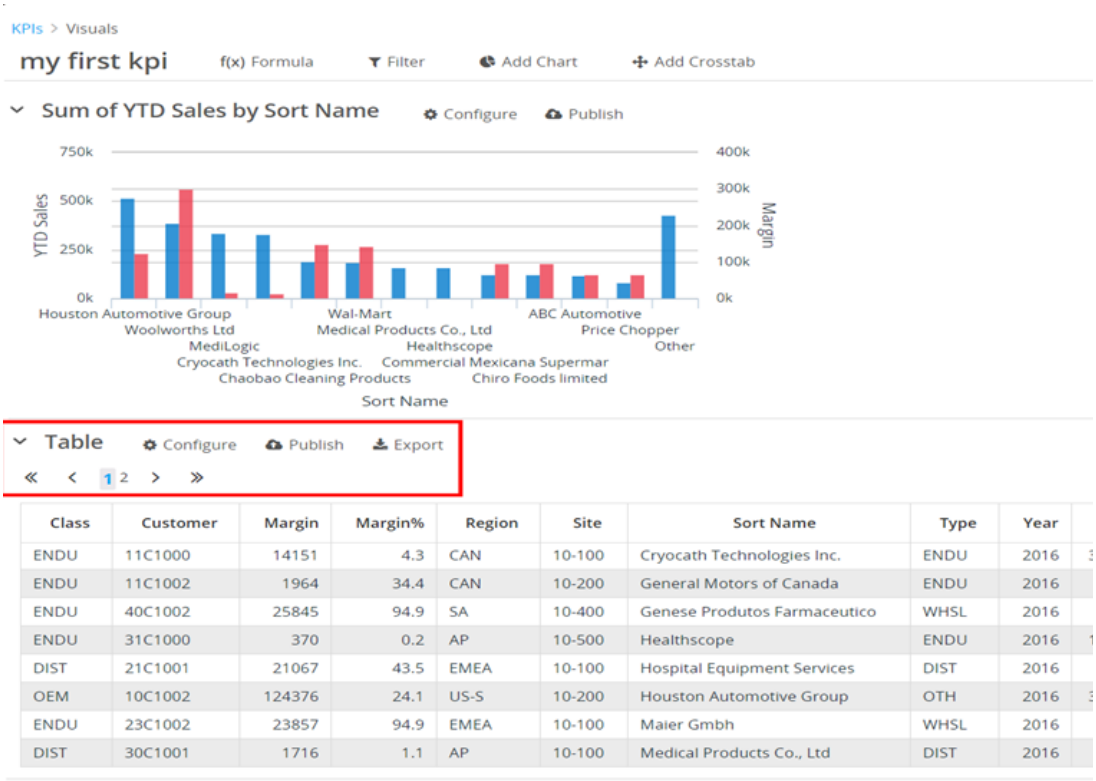
Under the charts there is the Table section. On the visuals screen there is always one table that shows the data returned from the data source.

The table is useful to verify that you have indeed the data that you expect. It can also be used to further analyze the data. When you apply filters on the data, then those will be applied on the charts and also on the table. When you add calculated columns with the Formula option, those additional calculated columns will also be visible in the table.

Above the table we have a Configure icon. When you click that, you get access to options to hide or show columns in the table, to sort the data in the table, to create groups and to aggregate data (adding up, counting, min, max, average, etc) and to control the paging in the table display.

Next there is a Publish icon allowing to publish the table to the shared Gallery so that it can be used on the action center. In the QAD predefined action centers we did not publish the complete data tables, because those typically take a lot of screen space.

Next to the Publish there is also an Export icon that allows you to export the data of the table to Excel, CSV format, or in a PDF file.



Class	Customer	Margin	Margin%	Region	Site	Sort Name	Type	Year
ENDU	11C1000	14151	4.3	CAN	10-100	Cryocath Technologies Inc.	ENDU	2016
ENDU	11C1002	1964	34.4	CAN	10-200	General Motors of Canada	ENDU	2016
ENDU	40C1002	25845	94.9	SA	10-400	Genese Produtos Farmaceutico	WHSL	2016
ENDU	31C1000	370	0.2	AP	10-500	Healthscope	ENDU	2016
DIST	21C1001	21067	43.5	EMEA	10-100	Hospital Equipment Services	DIST	2016
OEM	10C1002	124376	24.1	US-S	10-200	Houston Automotive Group	OTH	2016
ENDU	23C1002	23857	94.9	EMEA	10-100	Maier Gmbh	WHSL	2016
DIST	30C1001	1716	1.1	AP	10-100	Medical Products Co., Ltd	DIST	2016

Save or Reset Visual

At the bottom of the Visuals screen there is a Save and Reset button.

The Save button saves all the configuration you made in this screen so that next time when you open the screen for the KPI, it will remember all the settings.

But this does not make the visuals available for action centers. Therefore you have to Publish the visuals.

The Reset button is the opposite of the Save. Reset will wipe out all the settings you made in the visuals screen and you can start over from a pure data table with the raw data as returned from the data source. A warning is displayed when you hit Reset to avoid accidental resets.

Chart types - Bar Chart

Depending on chart type also the properties you can set vary.

Bar chart is the first and most powerful chart type.

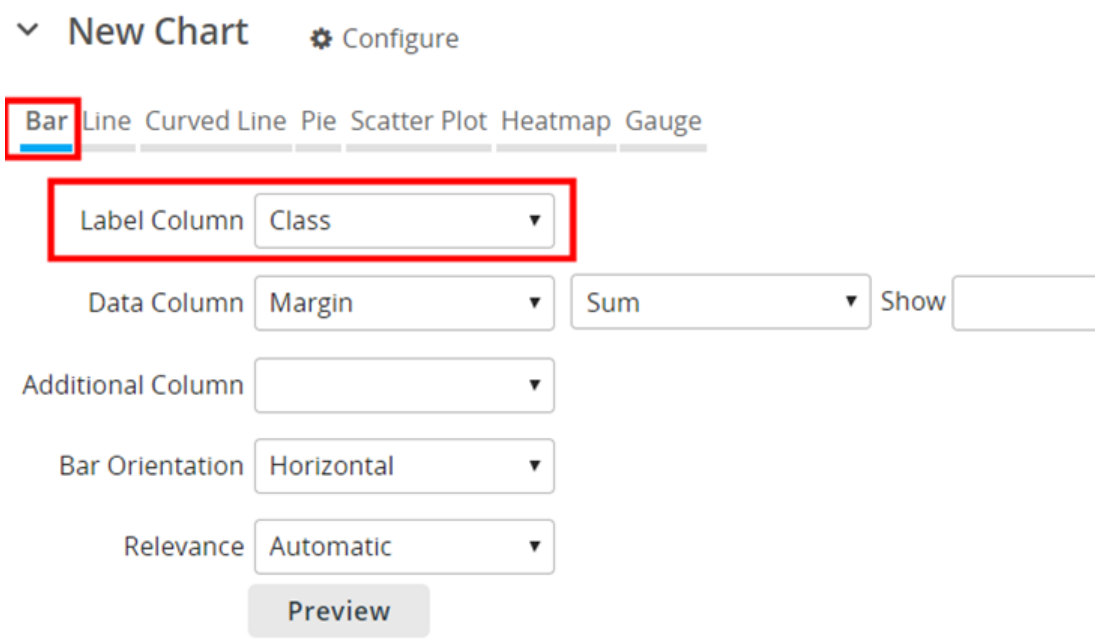
“Label column” is the ‘group by’ dimension (for example to group by Customer, or by Site, or by Item, or by Entity) and a bar is drawn for each group. Only text and date fields are listed in the drop-down list.

“Data Column” is the facts that you want to measure (“measure” in BI). Typically this is a numeric field, but it can also be a text or code field that you want to ‘count’. Think of the sales order number and count the number of distinct sales orders. Right from the “Data Column” you can select the aggregation type (that can be aSum, Average, Minimum, Maximum, Standard Deviation, Count, Distinct Count). When the data column is a text or date type field then only Count and Distinct count are possible. Right from the aggregation type is the “Show” option for having values on the bars. Note that showing the values can give an untidy look when many large numbers are displayed across the bars. Even without the values, users always can hover over the bars to see the values in the tooltip.

The “Additional Column” is optional and it allows you to add a second measure in the chart. For example, if the first Data Column you have the YTD Sales, then you could have the Additional data Column set to YTD Cost of Sales.

The additional column can have its own aggregation type (sum, count…) or it can use the same aggregation as the main data column (in that case, leave the second aggregation type blank).

The additional column can displayed as side-by-side bars or as a line, curved line or scatter plot. When the additional column is a code field that is a partition of the main label dimension with the same aggregation, then stacked bars become an option to select (see next slide).



The “Bar Orientation” allows you to choose horizontal or vertical bars. Horizontal are default because it displays long labels better. But Vertical is a more common form that is easier to read (if the labels are short). Bar orientation disappears when the Additional column is a line or curved line.

“Relevance” makes that only the most relevant values are displayed (for example the top 10 customers). You can turn this off when you want to see all the values. With many labels it’s sometimes better to only show the top 10 or top 20. You can also set this to ‘automatic’ and let the system determine the best fitting for the ‘top’ list.
When relevance is applied, the remaining records that fall outside the top are charted in a bar labeled ‘Others’. As of the September 2018 release there is also an option to hide the Others bar. That is useful if that bar is so large that it dwarfs the other bars and makes those unreadable. In that case, by hiding the Other category the named bars will show at their full size.

Once you have set all the configuration properties for a chart, you click the Preview button and the chart is displayed.

Stacked Bar Chart

Here we have an example of the stacked bars. The Sales by region has Region as the main label dimension and it has the customer name (called Sort name) as an additional column.
When leaving the aggregation type for the customer name blank, the stacked bars option becomes available. And the result is that the region bars are sub-divided in different colors for each customer.

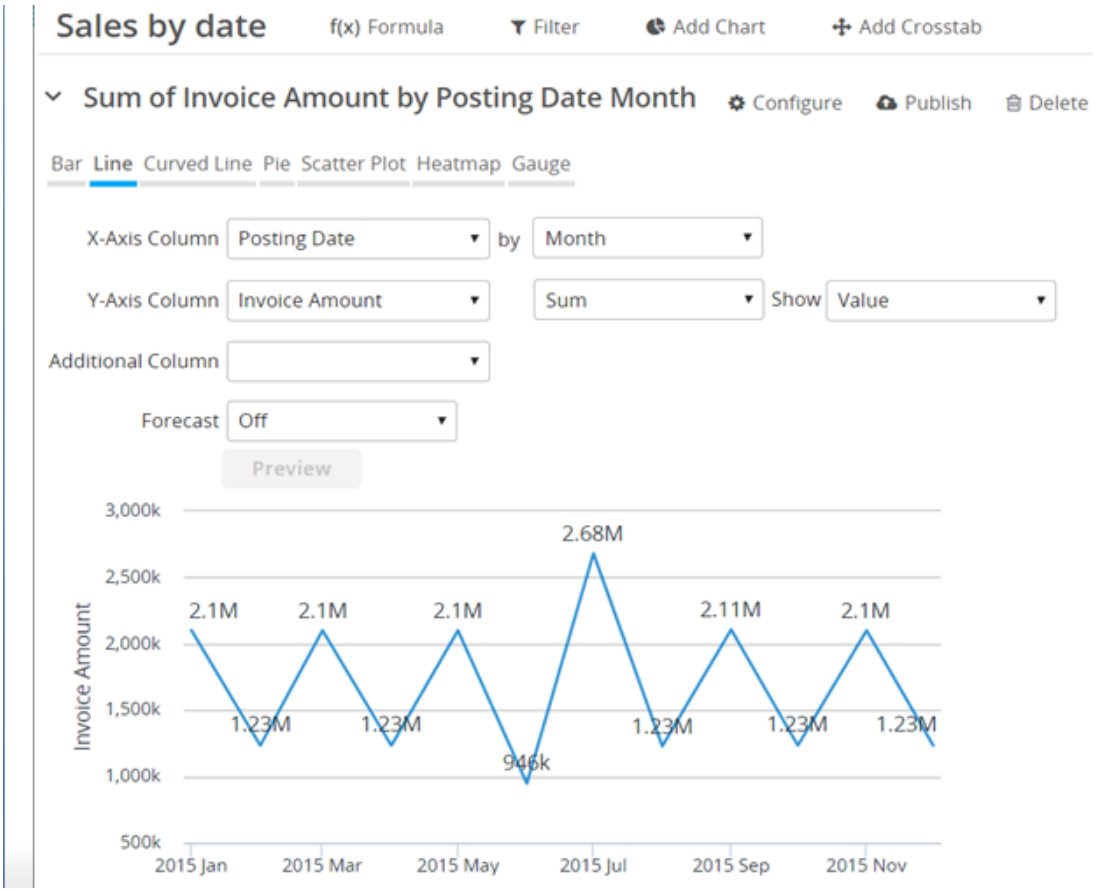
Line Chart

Line charts are used for charting values along a time line. And also for charting two numeric dimensions that have a correlation.

In the configuration of a line chart you have to select an X-axis column and a Y-axis column. For the X-axis only date or numeric fields can be used.

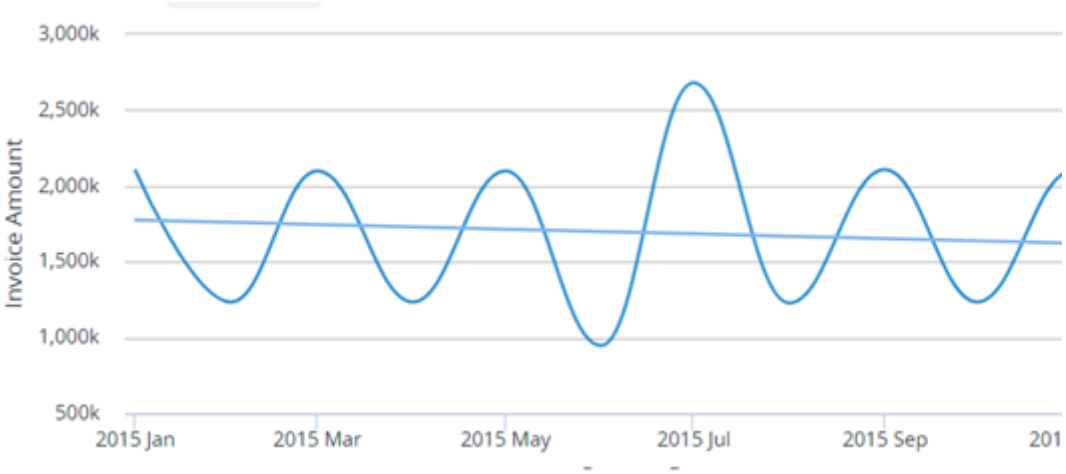
For the Y-axis numeric fields and text fields can be used.
When the Y-axis is a numeric field then you can choose to show aggregated values like sum, average, minimum, maximum, standard deviation.
When the Y-axis is a text field then you can only count records or make a distinct count.

In this example we will chart the totals of the invoiced values over time.
The Posting Date is used as the X-axis. With Date fields you get the choice to show those by year, by quarter, by month, by day and by fiscal quarter. The screenshot shows Month.
For the Y-axis we set the sum of Invoice Amount. So it shows the totals invoiced each month.



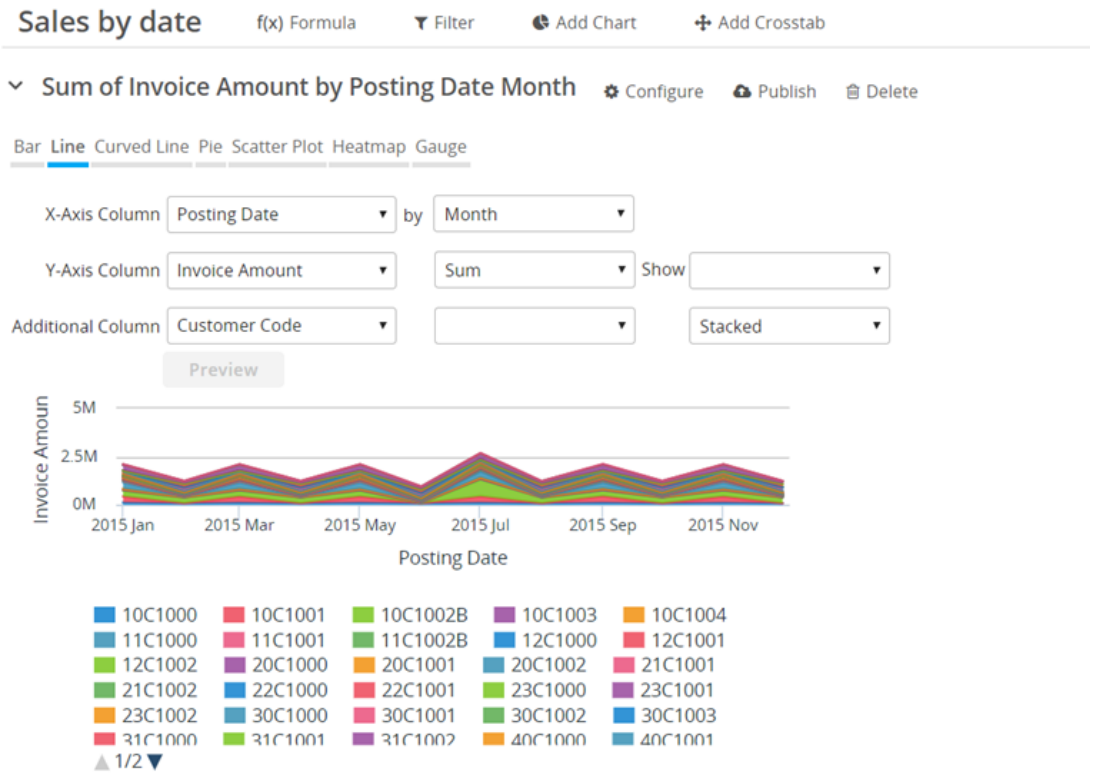
Curved line Chart

This chart type has the same settings a the normal line, but shows a fluent curved line.



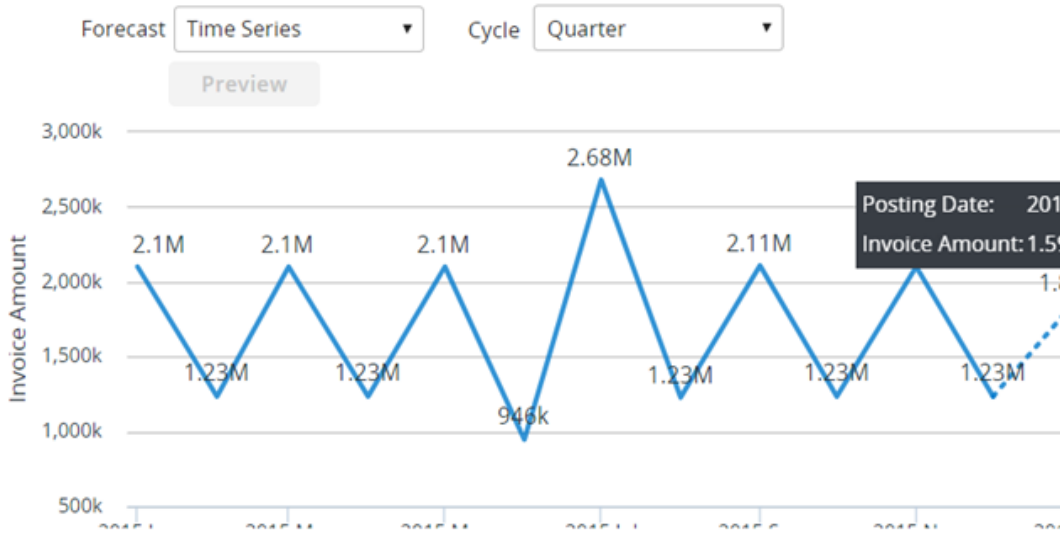
Stacked Line Chart

Also in line charts you can have an additional columns. In this example we added the customer codes so that the invoiced values per customer are displayed in different colors.
You can click on the customer codes in the legend to hide or show certain customers.



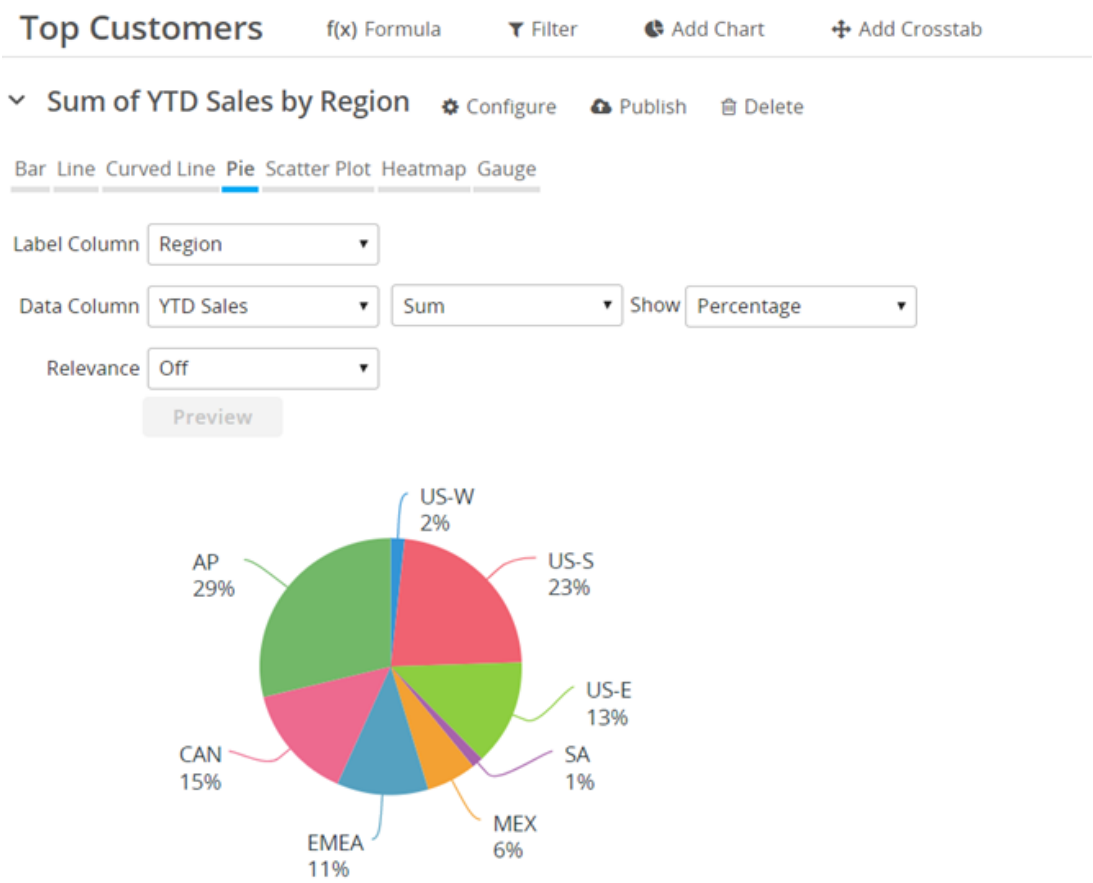
Trend line and forecast

A special property of line charts is the capability to show trend lines or forecasts.
In this example we have extended the sales by customer chart with the forecast for the following months.



Pie Chart

Pie charts are the best choice when you want to compare values for a Label column that has a small number of distinct codes.
As an example we use the Region as the Label column and we make the sum of the YTD Sales by region. We also added the percentage to show as a number for each slice.
Since the number of slices is limited we set the Relevance parameter to OFF so that all regions are displayed.

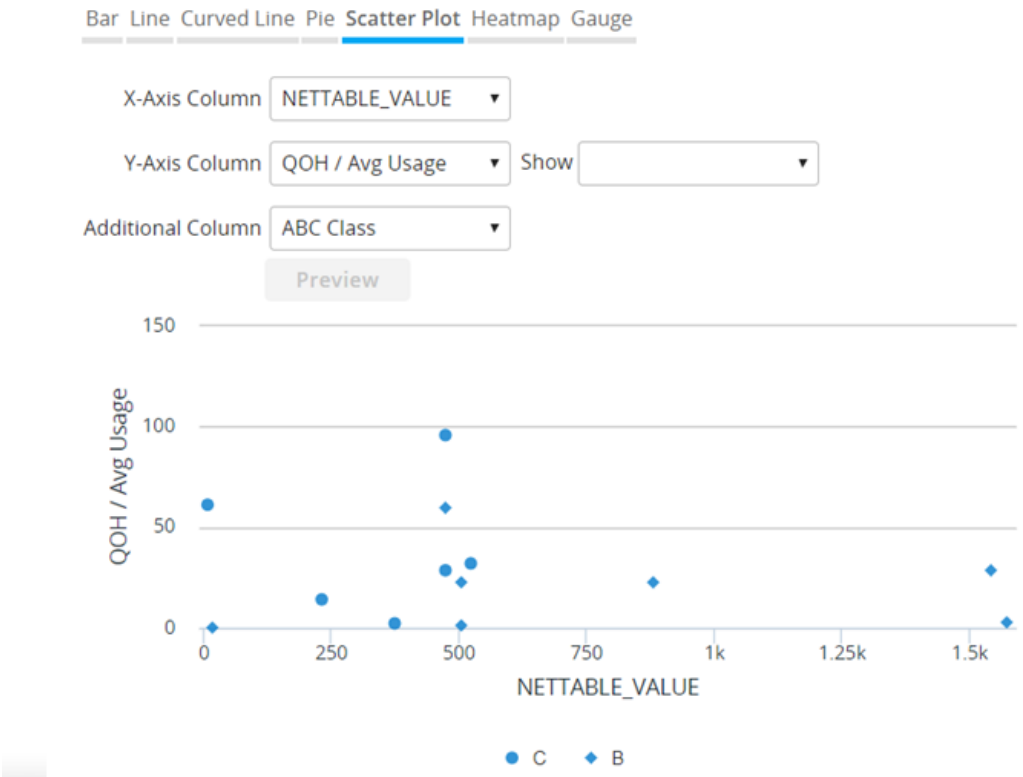


Scatter Plot

Scatter plots allow the visualizing and analyzing of the relationship and correlation between two sets of quantitative data. In most other chart types, you find a 'label' dimension on one of the axes, but for a scatter plot, the dimension is represented by points in the chart, and the measures are found on each of the two axes. For example, the X-axis represents the nettable value and the Y-axis represents the Quantity on Hand over average usage and each dot in the

chart represents an item with the legend showing the ABC class of the items. As you can see this results in interesting but rather scientific type of charts.

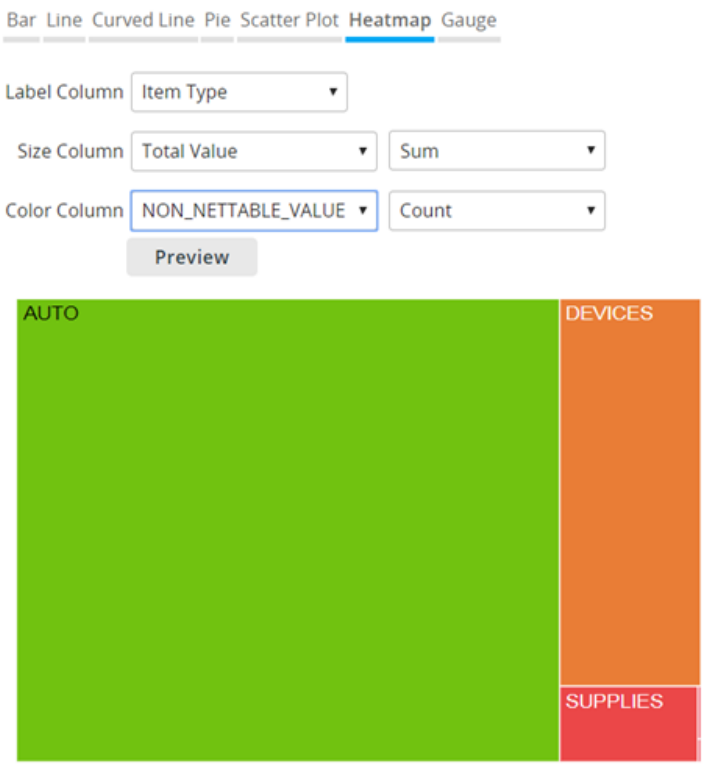
Note: Inventory - Turns by Item Historical Average (QAD)Taking the average qty of sales and issues divided by quantity on hand provides a hybrid Inventory Turns ratio.



Heat map

The heat map also combines the two measures over a selected dimension. Each rectangle represents a group in the Label dimension, in this example the Item Type is the Label dimension.

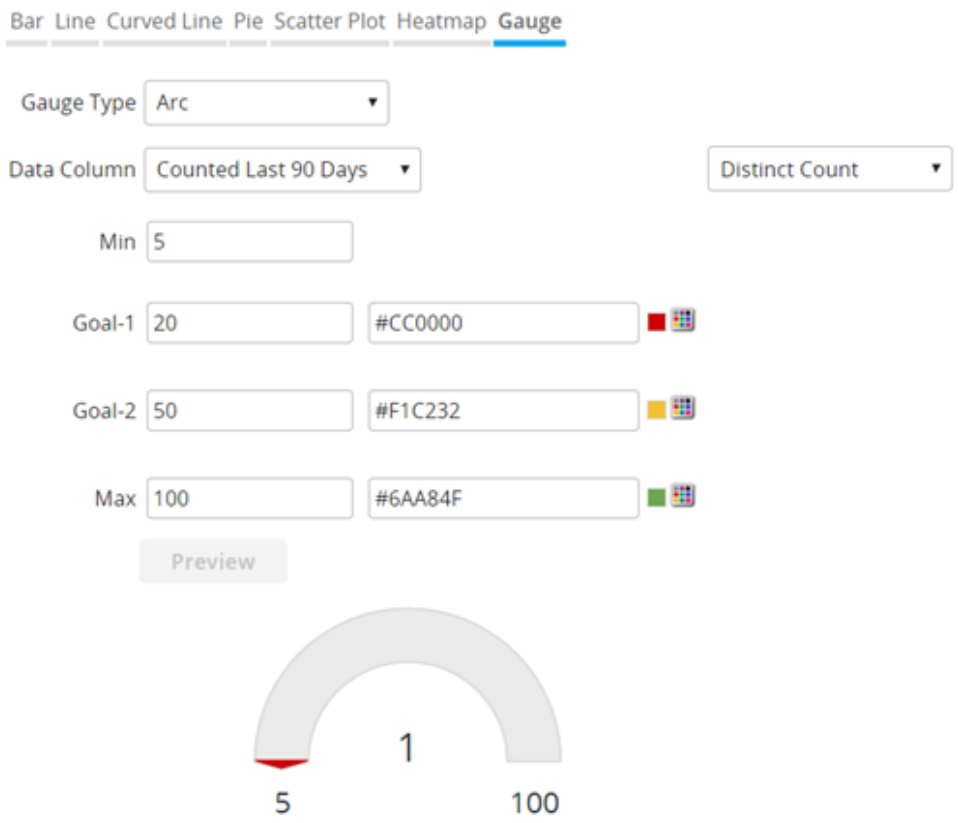
Per Item Type there is one rectangle displayed. The size of the rectangle represents the total value of all the items of this type. The color of the rectangle represents the total non-nettable value of all the items of the type. So the graph is measuring two values at the same time.



Gauge

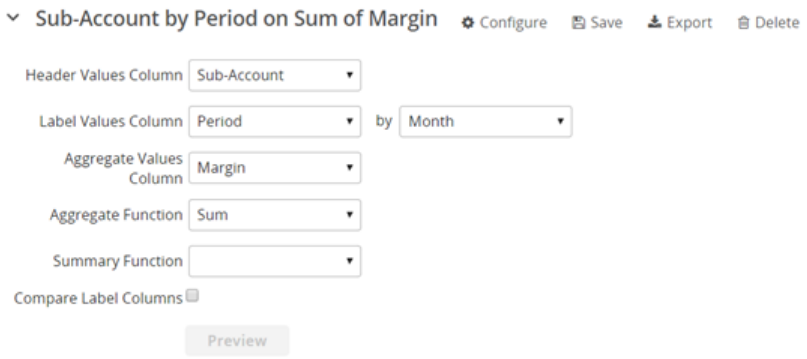
A gauge is typically used to measure a single number against a target. For this visual you start by choosing the gauge type (Arc, Balloon bar, Bullet bar).

In the Data Column you specify what you want to measure. If the data column is numeric, then you can make the sum, average, minimum, maximum or standard deviation. And if the data column is a non-numeric field, then you can count the records (with Count or Distinct count).
Then you specify the Min and Max values of the gauge and optionally the goals (target values) with a color. When the goal is reached, then the arc will get the color associated with that goal.
In this example we count how many items where counted in the last 90 days and we have a first goal of 20 (distinct) items to be counted, and a second goal of 50 (distinct) items to be counted and a maximum of 100.



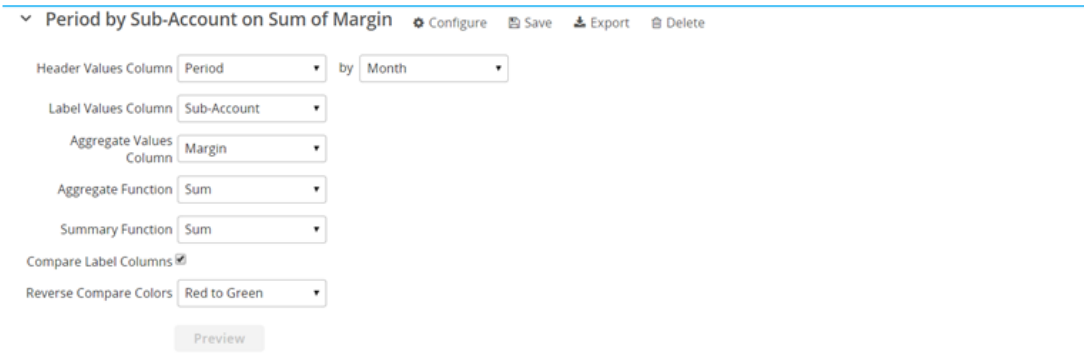
Cross Tabs

Cross tabs, also called pivot tables, are small summary tables that transform flat source data into two dimensional tables with one dimension as columns and the other dimension as rows.
This format not graphical but it's very easy to read, and it's very popular with Excel users.
You can create Cross tab sections by clicking the Cross tab icon at the top of the visuals screen.
In this example we used the Net Profit Margin KPI data and set the Sub-Account dimension as the Header Value Columns and the Periods as the Label Value columns that become rows in the cross tab. We selected the Margin as the Aggregate Values Column. That means that the values in the cross tab are the margin numbers and we made the sum of these by selecting Sum for Aggregate function.



Period	Blank	Cons	Gserv	Mech
2017 January	-662,413.3	-133,102.5	-342,330.2	1,302,598.6
2017 February	-278,897.1	-79,735.7	-135,510.1	601,121.9
2017 March	-662,413.3	-133,102.5	-344,330.2	1,293,007.7
2017 April	-278,933.2	-79,735.7	-135,510.1	675,139.8
2017 June			6,045.5	
2017 July			992.7	
2017 August			6,045.5	
2017 September			992.7	
2017 May			992.7	

The optional Summary function allows you to add overall totals.
The Compare Label Columns will show the difference between the columns as a value and as a percentage.
In this example we see how the margin for each sub-account grew or shrank over the months.



Sub-Account	Total Sum	2017 January	2017 February	2017 March	2017 April	2017 May	2017 June	2017 Ju
	620,923.1	164,752.5	106,979	153,161.7	180,960.8	992.7	6,045.5	
Mech	3,871,868	1,302,598.6	601,121.9↓ -53.85% -701,476.7	1,293,007.7↑ 115.10% 691,885.8	675,139.8↓ -47.79% -617,868			
Blank	-1,882,657	-662,413.3	-278,897.1↑ 57.90% 383,516.2	-662,413.3↓ -137.51% -383,516.2	-278,933.2↑ 57.89% 383,480.1			
Gserv	-942,611.5	-342,330.2	-135,510.1↑ 60.42% 206,820.1	-344,330.2↓ -154.10% -208,820.1	-135,510.1↑ 60.65% 208,820.1	992.7↑ 100.73% 136,502.8	6,045.5↑ 508.99% 5,052.8	9 -83.58% -5
Cons	-425,676.5	-133,102.5	-79,735.7↑ 40.09% 53,366.8	-133,102.5↓ -66.93% -53,366.8	-79,735.7↑ 40.09% 53,366.8			

Formulas

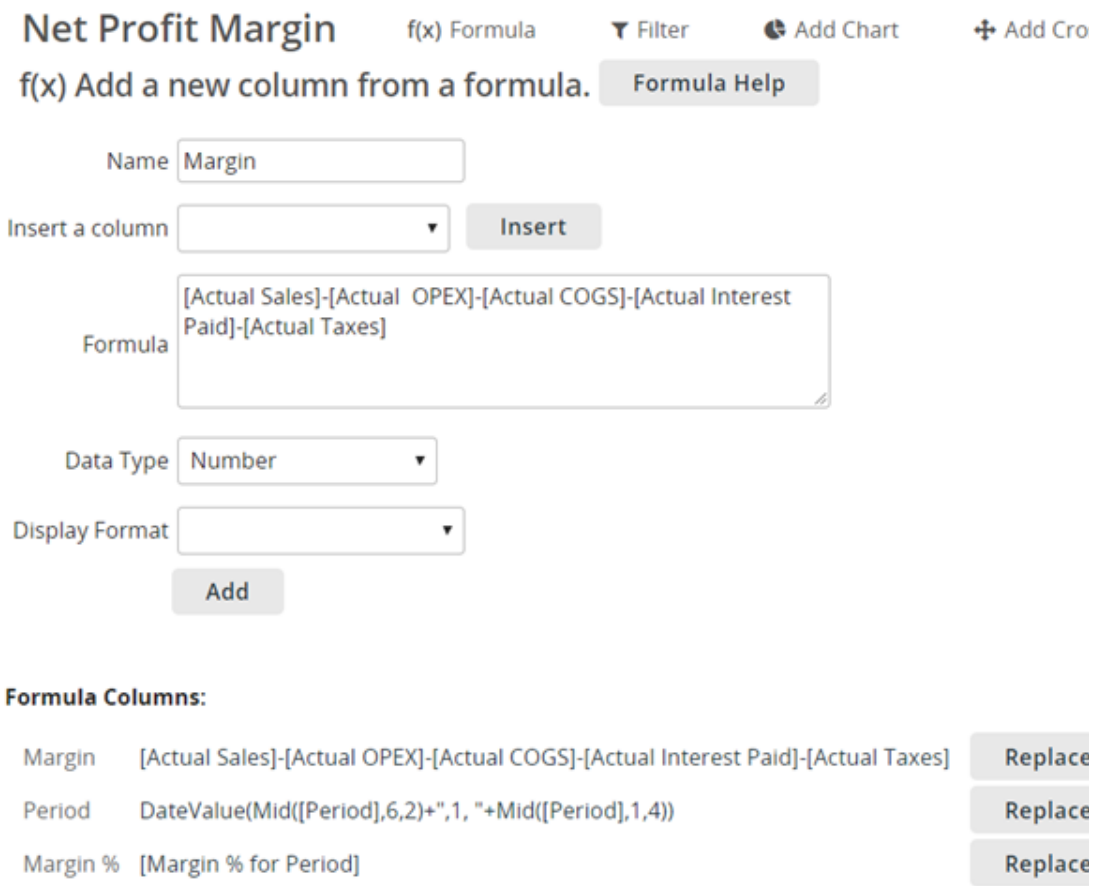
There are many cases where you need additional calculations or transformations of the data that you get from the data source to make it more useful for the KPI visualization.

That can be a simple arithmetic like adding, subtracting, multiplying or dividing columns and the result will be stored in a new column in the data table.

You can open the Formula section by clicking the f(x) Formula icon at the top of the visuals screen.

In this example we will calculate the Margin by subtracting the cost elements from the gross sales. You start by entering a name for the new result column. Then you create the formula.
You can type the complete formula directly in the 'Formula' box or you can use the 'Insert a column' drop-down list to select an existing column and when you click the Insert button, that column name is inserted in the Formula box between square brackets. That is how columns must be addressed in the formula.
The data type indicates what type of data the calculated result should be. For arithmetic that is typically a number. In other cases you will work with dates and want the result to be a date format. You must always pay good attention to the data type, because the charts will react quite differently depending on the data type. For example, a Date type can be x-axis in a line chart, but a text type can not.
The display format is optional and allows you to display numbers or dates in a special format.
When you click the Add button, the newly composed formula is added and a new column is added in the data table at the bottom of the screen. When that column contains question marks, that is an indication that the formula is wrong. When you correct a formula, instead of using the Add button, you must use the Replace button at the bottom of the formula section. For modifying an existing formula listed at the bottom of the formula section but not displayed anymore in the Formula box, you click on the name of the formula at the bottom of the formula section to bring back the formula details in the fields above it.

The best way to find out why a formula gives unexpected results, is to build it up step by step, adding one element of calculation at a time.
Formulas are executed in the order that they are listed and you can use the result of one formula in the next formula.
Formulas also allow you to do powerful transformations of data: you can calculate the difference between dates, you can add a true/false column that is the result of a IF condition, you can transform a text field in a number field, a number field in a text field, you can truncate the length of long labels, you can transform a period in the first or last day of the period, and so on. In fact the Formula is supporting many visual basic functions and syntax. The Formula Help button at the top gives you access to a help page with all available functions.



Formula use cases

Here are some more examples of what formulas can do:

Date calculations: you can create custom time buckets, where each bucket contains data that falls in a certain interval. For example Invoices that are 10 days overdue, 20 days overdue, 30 days overdue.
You can do the same for week or month buckets.
You create true/false columns that tell you if the data falls in the current month so that you can only show the current month's data or the opposite and exclude the current month's data when its expected to be incomplete.

Categorization: you create text columns that represent a category the data falls and use that category column as label column in a bar or pie chart.
We already mentioned that you can make the codes or descriptions shorter and you can also combine two or more columns in a single column.

Examples

abs(DateDiff("d", [Performance Date], [Effective Date]))

/* overdue so of so0082 */
DueDateGroup => DateDiff("d", Date(), [Due Date])
DueDateGroupNumber => IIF([DueDateGroup]<-7, -3, IIF([DueDateGroup]<-1, -2, IIF([DueDateGroup]<0, -1, IIF ([DueDateGroup]=0, 0, IIF([DueDateGroup]=1, 1, IIF([DueDateGroup]<8, 2, IIF([DueDateGroup]<15, 3, IIF ([DueDateGroup]<31, 4, 5))))))))

DueDateGroupNumber => IIF(DateDiff("d", Date(), [Due Date]) <-7, -3, IIF(DateDiff("d", Date(), [Due Date]) <-1, -2, IIF (DateDiff("d", Date(), [Due Date])<0, -1, IIF(DateDiff("d", Date(), [Due Date])=0, 0, IIF(DateDiff("d", Date(), [Due Date])=1, 1, IIF(DateDiff("d", Date(), [Due Date])<8, 2, IIF(DateDiff("d", Date(), [Due Date])<15, 3, IIF(DateDiff("d", Date(), [Due Date])<31, 4, 5))))))))

DueDateGroupLabel => IIF([DueDateGroupNumber] =-3, "Overdue > 7 Days", IIF([DueDateGroupNumber]=-2, "Overdue 2-7 Days", IIF([DueDateGroupNumber]=-1, "Due Yesterday", IIF([DueDateGroupNumber]=0, "Due Today", IIF ([DueDateGroupNumber]=1, "Due Tomorrow", IIF([DueDateGroupNumber]=2, "Due in 2-7 Days", IIF ([DueDateGroupNumber]=3, "Due in 8-14 Days", IIF([DueDateGroupNumber]=4, "Due in 15-30 Days", "Due > 30 Days")))))))

DueDateGroupLabel => IIF(DateDiff("d", Date(), [Due Date]) <-7, "1- Overdue > 7 Days", IIF(DateDiff("d", Date(), [Due Date]) <-1, "2- Overdue 2-7 Days", IIF(DateDiff("d", Date(), [Due Date])<0, "3- Due Yesterday", IIF(DateDiff("d", Date(), [Due Date])=0, "4- Due Today", IIF(DateDiff("d", Date(), [Due Date])=1, "5- Due Tomorrow", IIF(DateDiff("d", Date(), [Due Date])<8, "6- Due in 2-7 Days", IIF(DateDiff("d", Date(), [Due Date])<15, "7- Due in 8-14 Days", IIF(DateDiff("d", Date(), [Due Date])<31, "8- Due in 15-30 Days", "9- Due > 30 Days")))))))

SummaryLabel => IIF([DueDateGroupNumber] <0, "Overdue Orders", "Non-Overdue Orders")

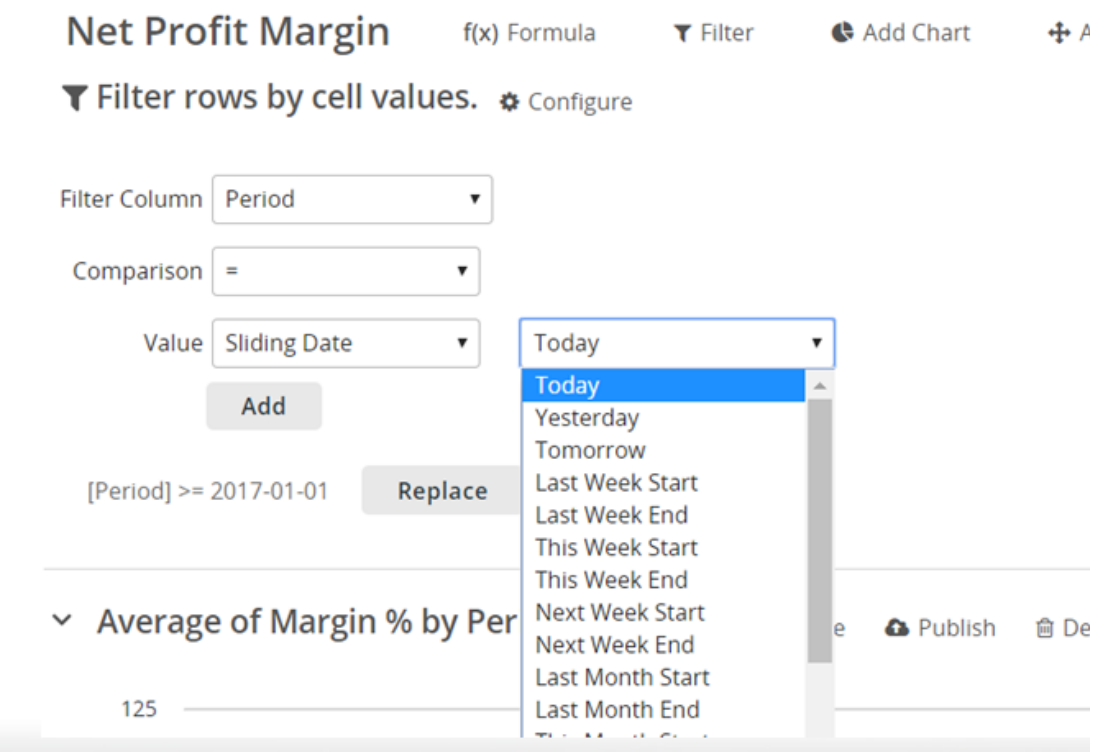
IsPastQuarter => IIF([Quarter] < Year(Date()) + "-Q" + Math.ceil(Month(Date())/3),1,0)

if(DateDiff("d", [Due Date], now()) >=0) {FormatNumber(Math.ceil(DateDiff("d", [Due Date], now()) / 30),0) }

IIF(DateDiff("d", Date(), [Due Date]) <0, “overdue”,”not overdue”)

Filters

In many cases you want to filter the data for a KPI visual.
You can open the Filter section by clicking the Filter icon at the top of the visuals screen. Use the Configure button to add or modify the filter details.
You create a filter by first specifying the Filter Column. That is the data column that you want to apply the filter on. The Comparison contains the operator like equals, or greater than.
There are many possible comparisons. The drop-down lists shows them all and they are pretty easy to understand.
In the Value field you can specify what to compare with. That can be a constant (fixed value) and for date types you can also compare with a sliding date like ‘today’ or ‘this year start’.
For sliding dates there is also a drop-down list with possible values to compare with.
When you have set the filter criteria, you click the Add button and the filter is applied on the data. That counts for both the data table and all the visuals on the screen.
You can add multiple filters that are all applied. Multiple filters using different Filter columns are combined as AND condition (both must be true). Multiple filters using the same Filter column are combined with OR condition (one of them must be true). But you can change an AND in OR by clicking on the button labeled AND; and the other way around.



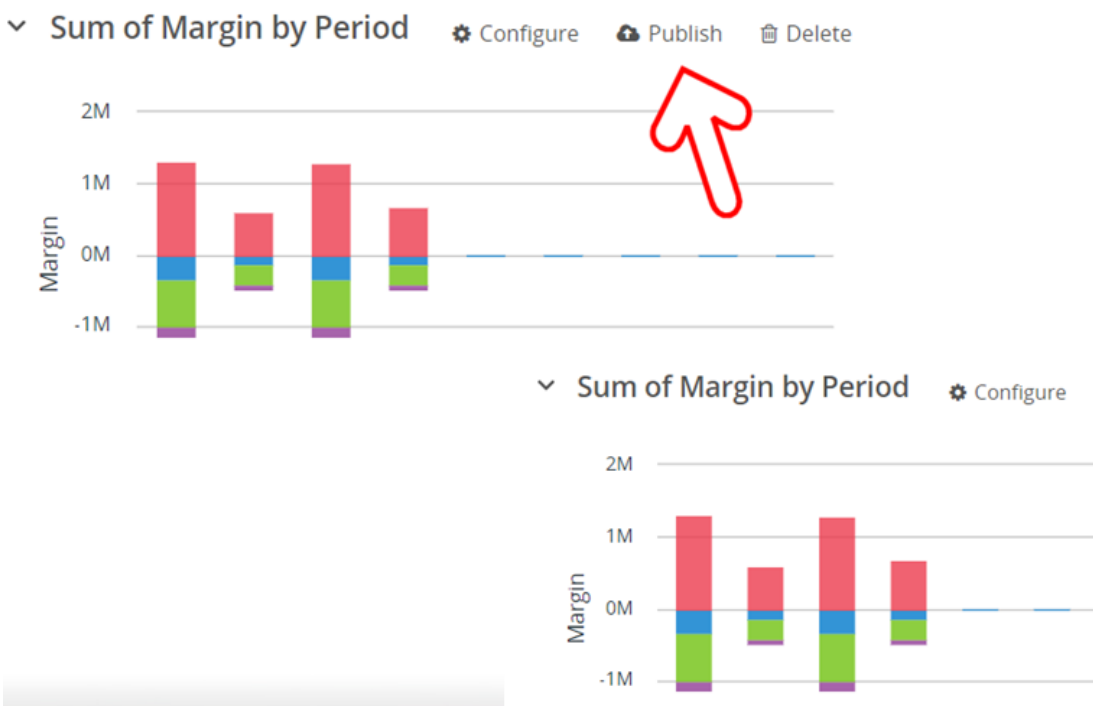
As of the September 2018 release, filters applied on source data columns are passed to the back-end query service and only the records that fulfill the filter conditions are retrieved. This gives an important performance improvement for large datasets. In previous versions the filtering set on a visual was applied after all the data was retrieved from the back-end data source.

Browse Filters in the KPI vs Filters in the visual

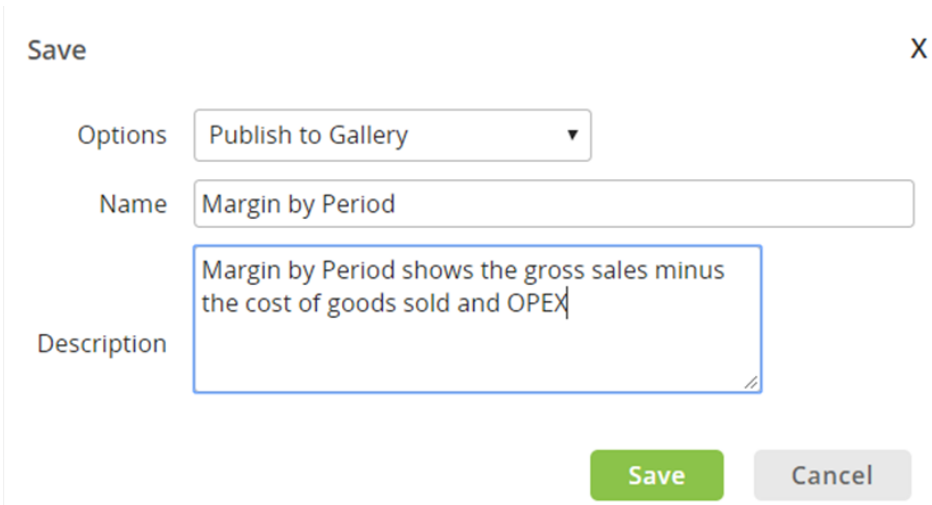
There are two places in action centers where you can set filters on the data. You can set a filter in the KPI maintenance screen in the Configure Browse section and you can set a filter on the Visual as described in the previous paragraphs.
Setting the filter in the browse configuration is more beneficial for performance, but the filter will apply for all visuals of the KPI and can typically only be changed by an administrator or power user who maintains the KPI metadata.
Setting a filter in the visual can be done by an end-user and only applies on the version of the visual that is being changed. When developing KPIs you must thus carefully evaluate which filters can apply for all use cases of the KPI and which filters are needed for a specific visual.

Publishing a visual

Each time you complete a new visual with formulas and filters and all desired settings, you can save the complete page settings by clicking Save at the bottom.
But in most cases you want to make the chart available for use in the Action Centers. That is something you do by clicking the Publish icon just above the chart.
Note that if you have created multiple charts in the Visuals page, you will have to Publish those one by one.
The publish action makes the chart available in a shared Gallery with all KPI visuals in the system and the action center users can later pick visuals from that Gallery.
Since the gallery is shared with many users, only privileged key users will be given access to update the Gallery. In another video about security, I explain the details about action center security.
The icon is called Publish when you create new visuals from the KPI maintenance screen, but you will see an icon labeled Save when you are modifying visuals or creating new visuals from within the action center.
That is because for KPI maintenance has only one option is to Publish the chart to the Gallery, while in the Action Center there is a choice between saving a new or modified chart on the action center or to publish it in the Gallery.



When you Publish a visual to the Gallery a dialog box will open and you can enter a name and a description for the visual. It will be stored in the Gallery with that name.



3 States of a visual

The visual definition can be stored in 3 different places.

1- At the level of the KPI maintenance when you click the Visual button at the bottom you can Save the latest settings. This will create an xml save file on disk on the server.

2- When you publish a visual to the Gallery. Then the setting are added in a shared Gallery.xml file.

3- On the Action Center (dashboard). When you add the visual from the Gallery to the action center, it copies the settings and stores those on the level of the action center (dashboard)

Modifications are independent and that sometimes leads to confusion. But it also offers great flexibility in for example securing the Gallery versions and still allowing users to modify their action center versions.

Please also not that Gallery visuals cannot be edited. Those can only be changed by publishing new visuals or delete old visuals from the gallery (secured)
New version in gallery is always a new duplicate entry => need to delete to old version

Visual data is stored in XML files (in a single directory) Do not tamper with those files!

Securing and sharing action centers

User Types

The users of Action Centers can be categorized in two groups.

End-users

The first group are the end-users of Action Centers; typically this are the managers who consume the information in support of decision making.

Administrators/power users

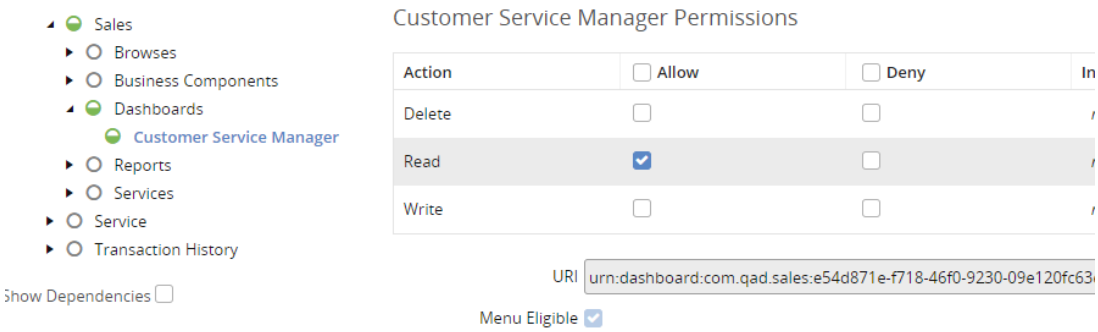
The second type of users are the administrators and power users who can create KPIs, can create KPI visuals, can publish KPI visuals to the shared Gallery and they can create shared role based action centers. For all these objects they also have rights to update them or even delete them.

Shared role based Action Centers

End-users have access to the shared action centers that were created by administrators or by QAD and that are shared with them based on their role. The shared action centers are typically set read-only for them. They can fully analyze them, but not change the settings.

To set permission to access a specific Action Center => urn:dashboard:com.qad.<*APP*>:<*GUID*> (the action center name is visible in the Role Permissions tree)

Example for “Customer Service Manager”



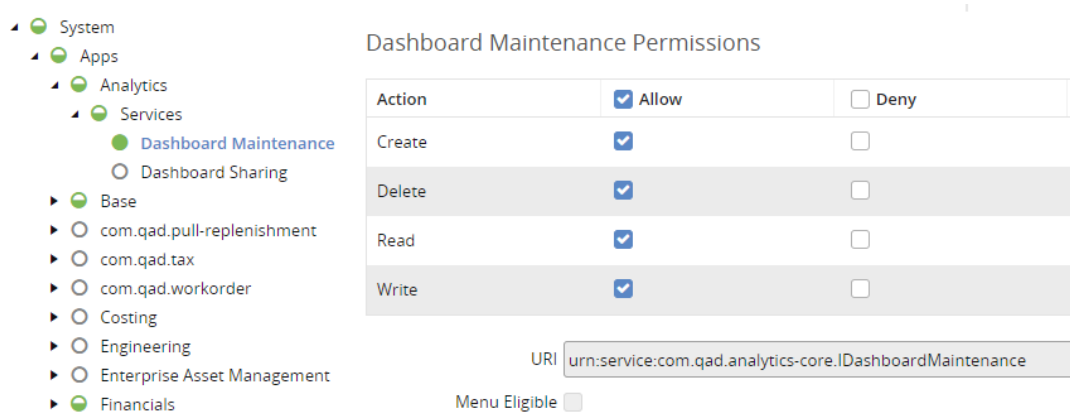
Read access means you can see and access the action center, analyse the KPIs but you cannot add/remove KPIs from the specific action center.

The QAD predefined action centers come out of the box with read permissions for the corresponding QAD roles.

Personal Action Centers

The end-users can also be allowed to create and maintain personal action centers.

This controlled by the permissions for "Dashboard Maintenance" under Analytics urn:service:com.qad.analytics-core.IDashboardMaintenance



Typically you will give access to all four of CRUD actions together (CRUD means ‘Create’, ’Read’, ’Update’, ’Delete’). This permission is also needed to create personal action centers.

End-users can expand KPI visuals from their action center so they can analyze those further with filtering, sorting, grouping and slicing.
If those visuals belong to their personal action center then they can also update the visuals on the action center or make copies. But they typically they will not be allowed to add new visuals to the gallery (see also 'Dashboard Sharing' permissions for admins).

From their action centers, end users always have read access to the shared gallery so that they can select visuals for their personal action centers.

Data source access

Any user running a KPI and needing to see the data must have access to the data source .

In the Role Permissions those resources can be searched by URI.

Operational browses can be found under the appropriate App e.g. "Inventory" => urn:app:com.qad.inventory
or under "Mfg/Pro Infrastructural Foundation" => urn:app:com.qad.mfgcoreplus

- Example Sales by Customer browse = > urn:browse:mfg:sa011

FRW kpis can be found in the KPI container of the Financials app urn:app:com.qad.financials

See list of all data source under Predifined Action Centers

KPI maintenance CRUD permission

To give admins and power users access to the KPI maintenance screen (create/edit/delete/migrate KPIs)

- Read permission for urn:browse:mfg:an002 This is the browse listing all browses in the KPI maintenance on the webUI..
- Read permission for urn:browse:fin:BKPI.SelectKPI This is the browse listing all FRW KPIs in the KPI maintenance on the webUI..
- Read permission for urn:be:com.qad.qra.kpi.IKpiMetadata This is the business entity that is the basis of the KPI maintenance screen.

Action Center and Gallery Create and Sharing permissions

Specific for action center users controlling if they can make updates to the Gallery

- Create, Delete permission for urn:service:com.qad.analytics-core.IDashboardSharing

Specific for action center users controlling if they can create/update/delete action centers

- Create permission for urn:service:com.qad.analytics-core.IDashboardMaintenance

Predefined Action Centers

This page just gives an overview of the KPIs and browses used for the QAD predefined action centers. For a detailed description and business background please check the online help user documentation.

Purchasing Manager (Purchasing App)

KPI	Browse	
Purchase Receipts Not On Time (pre-SM2 versions)	po041	PO Receipt Cost
Purchase Receipts On Time (pre-SM2 versions)	po041	PO Receipt Cost
Purchase Receipts In Full (pre-SM2 versions)	po804	Purchase Order Lines
Purchase Receipts Not In Full (pre-SM2 versions)	po804	Purchase Order Lines
Top Spend by Supplier (pre-SM2 versions)	po804	Purchase Order Lines
Purchasing - Commitments and Historical Spending - (QAD)	po804	Purchase Order Lines
Purchasing - Critical Items with No Qualified Secondary Source - (QAD)	po083	Item Browse
Purchasing - Data Integrity Analysis - (QAD)	po083	Item Browse
Purchasing - Future Planned Purchases - (QAD)	po082	MRP Detail
Purchasing - Inventory Effectiveness - (QAD)	ic761	Inventory Value by Item
Purchasing - On Time and In Full Analysis for All Orders - (QAD)	po080	Inventory Value By Item
Purchasing - On Time and In Full Analysis for Current and Prior Week - (QAD)	po080	Inventory Value By Item
Purchasing - On Time and In Full Analysis for Past 3 Months - (QAD)	po080	Inventory Value By Item
Purchasing - Past Due Analysis - (QAD)	po080	Inventory Value By Item
Purchasing - Source Agreement Analysis - (QAD)	po081	PO081
Purchasing - Two Year Spend Analysis - (QAD)	po804	Purchase Order Lines

Inventory Manager (Inventory App)

KPI	Browse	
Inventory Stockouts by Buyer-Planner	ic761	KPI Inventory Value by Item
Inventory to Expire	ic761	KPI Inventory Value by Item
Inventory Obsolescence Potential	ic761	KPI Inventory Value by Item
Inventory Value	ic761	KPI Inventory Value by Item
Inventory Item Scrap	ic762	Inventory Value by Item
Inventory Cycle Count	ic761	KPI Inventory Value by Item
Inventory Safety Value by Item	ic761	KPI Inventory Value by Item
Inventory by Days on Hand	ic752	Days On Hand Analysis
Inventory Data Health	ic761	KPI Inventory Value by Item

Maintenance Manager (Assetmgt App)

KPI	Browse	
Work Order Backlog	ea023	Maintenance Order
Work Order Compliance	ea023	Maintenance Order
Open Maintenance Request	ea030	Service Request

Downtime by Equipment Type	ea024	All Downtime
Failures by Equipment Type	ea025	All Failures
Material Cost	ea028	Maintenance Cost
Labor Cost	ea026	Labor History
Contractor Cost	ea027	Contractor Cost
Aging Work Order Backlog	ea023	Maintenance Order

Operations (PushProduction App)

KPI	Browse	
Operations - OEE Metrics, Summary by Production Line - (QAD)	wo253	OEE By Site and Production Line - 3 Months
Operations - Work Center OEE by Period - (QAD)	wo258	OEE By Work Center - 13 Weeks
Operations - Work Center OEE, Summary - (QAD)	wo257	OEE By Work Center - 3 Months
Operations - Department OEE Summary - (QAD)	wo255	OEE By Department - 3 Months
Operations - Department OEE by Period - (QAD)	wo256	OEE By Department - 13 Weeks
Operations - Production Line OEE by Period - QAD	wo254	OEE By Site and Production Line - 13 Weeks
Operations - OEE Metrics by Site, Summary - (QAD)	wo251	OEE By Site - 3 Months
Operations - OEE Metrics for 13 Periods - (QAD)	wo252	OEE By Site - 13 Weeks
Operations - Production Line Utilization and Efficiency by Site and 13 Periods - (QAD)	wo252	OEE By Site - 13 Weeks

Production OTIF (PushProduction App)

KPI	Browse	
Production OTIF - Overdue Production Orders - (QAD)	wo003	Work Orders
Production OTIF - Site OTIF (Production Line Data) Summary - (QAD)	wo261	OTIF By Site (Line Summary) - 3 Months
Production OTIF - Site OTIF (Production Line Data) by Period - (QAD)	wo262	OTIF By Site (Line Summary) - 13 Weeks
Production OTIF - Production Line OTIF Summary - (QAD)	wo263	OTIF By Site and Production Line - 3 Months
Production OTIF - Production Line OTIF by Period - (QAD)	wo264	OTIF By Site and Production Line - 13 Weeks
Production OTIF - Site OTIF (All Orders) Summary - (QAD)	wo265	OTIF By Site (Family Summary) - 3 Months
Production OTIF - Site OTIF (All Orders) by Period - (QAD)	wo266	OTIF By Site (Family Summary)- 13 Weeks
Production OTIF - OTIF by Product Family Summary - (QAD)	wo267	OTIF By Site and Product Family - 3 Months
Production OTIF - OTIF by Product Family by Period - (QAD)	wo268	OTIF By Site and Product Family - 13 Weeks
Production OTIF - Master Schedule Analysis (Past Due Open Orders) - (QAD)	wo271	OTIF and Overdue By Production Order
Production OTIF - Master Schedule OTIF % (Current) - (QAD)	wo271	OTIF and Overdue By Production Order
Production OTIF - Department Operation OTIF Summary - (QAD)	wo275	Operation OTIF By Department - 3

		Months
Production OTIF - Department Operations OTIF by Period - (QAD)	wo276	Operation OTIF By Department - 13 Weeks
Production OTIF - Work Center Operations OTIF Summary - (QAD)	wo277	Operation OTIF By Work Center - 3 Months
Production OTIF - Work Center Operations OTIF by Period - (QAD)	wo278	Operation OTIF By Work Center - 13 Weeks
Production OTIF - Master Schedule Analysis - (QAD)	wo271	OTIF and Overdue By Production Order
Production OTIF - Master Schedule OTIF Summary - (QAD)	wo271	OTIF and Overdue By Production Order

Customer Service Manager (Sales App)

KPI	Browse	
Open Orders by Due Date	so082	Sales Order Detail
Orders on Credit Hold	so083	Sales Order Credit
Top Customers	sa011	Sales by Customer
Top Items	sa012	Sales by Item
Late Shipments	so007	Invoice History
Late Payments	an001	Customer Invoices

Chief Financial Officer (Financials App)

KPI	Browse / FRW KPI	
Shipping Metrics	so007	Invoice History
Sales Analysis 2	BDInvoice.SelectDInvoice	Customer Invoices
Net Profit Margin	FRW KPI (no browse)	Net Profit Margin
Cash Flow Analysis	FRW KPI (no browse)	Cash Flow
Days Payable Outstanding	FRW KPI (no browse)	DPO
Days Sales Outstanding	FRW KPI (no browse)	DSO
Revenue Analysis	FRW KPI (no browse)	Sales
Global Revenue Analysis	FRW KPI (no browse)	Sales
OPEX Analysis	FRW KPI (no browse)	Net Profit Margin
Working Capital	FRW KPI (no browse)	Working Capital

Useful browses for Operational Metrics History and Transaction History (tr_hist)

Browse export (.brwx) of browse 'xx777' with Operational Metrics History Data

KPI export (.zip) with Operational Metric 'Invoices History' based on the browse xx777

Browse export (.brwx) of browse 'xx987' with Transaction History Data (tr_hist)

QAD Page explaining the promotion of a .NET UI browse to a webUI browse

Query Service

With the introduction of the Query Service in the September 2018 release Action Centers become capable of processing large volumes of data.

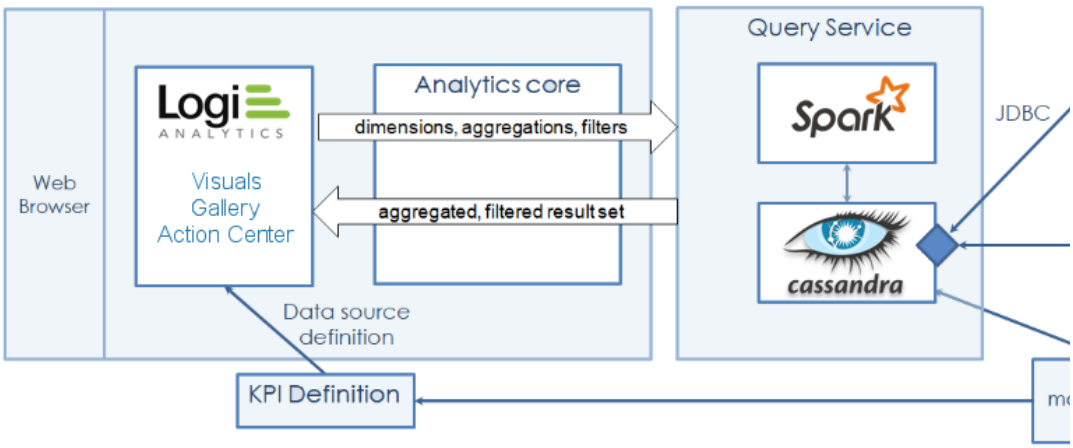
In earlier versions of Action Centers we set a limitation of 5,000 rows retrieved by the browse to avoid performance bottle necks in the rendering of the KPI visuals.

Query service has solved this issue. Regardless of the number of records in a browse result data set, the Query Service retrieves that data in an aggregated format and calculates the totals and averages values grouped by any of the dimensions of the browse. It does that automatically based on required dimension for a KPI visual. Therefor it has no impact on the steps required to create KPIs and Action Centers.

Under the cover, there are two new components that make this possible:

- The Cassandra Data Lake absorbs large volumes of data and stores it in an optimal format for querying. It gets refreshed with the latest ERP transactions seamlessly at scheduled times and on a user's request with a single click.
- Apache Spark Java processes the large data at light speed on a regular web server infrastructure. Spark transforms hundreds of thousands of raw data records into a small summarized data set in a few seconds. In our tests the average time for aggregating 400,000 records was below 5 seconds, and going up to a millions of records it scales up linearly.

The Query Service can process the data of any of the browses in the system, also the browses created by customers.



The query service uses the browse query definition to read the data from the ERP database and stores it in the Cassandra database. This can happen a different points in time: the first time when you create a new KPI it happens on the spot. Later it can be refreshed daily/weekly/monthly (based on the KPI definition). And it will typically also refresh the data when the system is restarted ⭐. And it can also be manually refreshed from the Action Center directly by clicking the refresh icon. The latter is an option that can be enabled/disabled in the KPI maintenance by the administrator.

⭐ there are system configuration properties that control this. See Action Center Technical Reference Guide

More information about the Query Service design can be found here QRA Query Service Designs